The 18th European Conference on Artificial Intelligence

Proceedings

6th International Cognitive Robotics Workshop

July 21-22, 2008
Patras, Greece

Yves Lespérance (Chair)
Gerhard Lakemeyer
Jan Peters
Fiora Pirri
Contents

Preface

Yves Lespérance .................................................. 3

Long Presentations

Towards Autonomous Design of Experiments for Robots

Timo Henne, Alex Juarez, Monica Reggiani, and Erwin Prassler ............... 4

Fred meets Tweety

Antonis Kakas, Loizos Michael, and Rob Miller ................................. 10

Repairing Decision-theoretic Policies Using Goal-oriented Planning

Ch. Mies, A. Ferrein, and G. Lakemeyer ........................................... 16

Designing Goal-Oriented Reactive Behaviours

Peter Novak and Michael Köster .................................................... 24

Representation and Integration: Combining Robot Control, High-Level Planning, and Action Learning

Ronald Petrick, Dirk Kraft, Kira Mourão, Christopher Geib, Nicolas Pugeault, Norbert Krüger, and Mark Steedman .................. 32

A Seamless Navigation Aid for the Blind

Georgios Sakellariou and Aristotelis Zosakis .................................. 42

Notes on a qualitative theory of shadows

Paulo Santos, Hannah M. Dee, and Valquiria Fenelon .......................... 47

Short Presentations

Evolutionary Genetic Programming for Autonomous Robot Motion Control

Fabrizio Abrate, Basilio Bona, and Frederico Tibaldi .......................... 55

Robot Topological Map Generation from Formal Route Instructions

Mohammed Elmogy, Christopher Habel, and Jianwei Zhang ................ 60

Towards an Episodic Memory for Cognitive Robots

Sascha Jockel, Martin Weser, Daniel Westhoff, and Jianwei Zhang .......... 68

A Cognitive Theory of Autodidactic Agency

Loizos Michael .............................................................................. 75

Combining appearance and behavioral cues for the purpose of person recognition

Fiora Pirri and Matia Pizzoli ............................................................ 81

Computational Framework for and the Realization of Cognitive Agents Providing Intelligent Assistance Capabilities

Marcin Skowron, Joerg Irran, and Brigitte Krenn .............................. 88
Organizing Committee

Yves Lespérance (Chair) York University, Toronto, Canada
Gerhard Lakemeyer Aachen University of Technology, Germany
Jan Peters Max Planck Institute, Tübingen, Germany
Fiora Pirri University of Rome “La Sapienza”, Italy

Program Committee

Michael Beetz Technical University of Munich, Germany
Silvia Coradeschi Orebro University, Sweden
Patrick Doherty Linkping University, Sweden
Maria Gini University of Minnesota, Minneapolis, USA
Félix Ingrand LAAS/CNRS, Toulouse, France
Gal A. Kaminka Bar Ilan University, Israel
Jim Little University of British Columbia, Canada
Francois Michaud University of Sherbrooke, Canada
Bernhard Nebel University of Freiburg, Germany
Maurice Pagnucco University of New South Wales, Australia
Joelle Pineau McGill University, Canada
Martin Riedmiller University of Osnabrück, Germany
Alessandro Saffiotti Orebro University, Sweden
Sebastian Sardina RMIT University, Melbourne, Australia
Murray Shanahan Imperial College, London, UK
Stuart C. Shapiro SUNY Buffalo, USA
Mikhail Soutchanski Ryerson University, Toronto, Canada
Peter Stone University of Texas at Austin, USA
Michael Thielscher Dresden University of Technology, Germany
Sven Wachsmuth Bielefeld University, Germany
Jeremy Wyatt University of Birmingham, UK
Preface

Cognitive robotics is concerned with integrating reasoning, perception, and action within a uniform theoretical and implementation framework (using methods drawn from logic, probability and decision theory, reinforcement learning, game theory, etc.). It is quite a young field of research. However, the use of robots and softbots is becoming more and more widespread, with many commercial products on the market. Complex applications and the need for effective interaction with humans are increasing demand for robots that are capable of deliberation and other high-level cognitive functions. Models from cognitive science and techniques from machine learning are being used to enable robots to extend their knowledge and skills. Combining results from mainstream robotics and computer vision with those from knowledge representation and reasoning, machine learning, and cognitive science has been central to research in cognitive robotics.

The International Cognitive Robotics Workshop aims at bringing together researchers involved in all aspects of the theory and implementation of cognitive robotics, to discuss current work and future directions. The current edition of the Workshop is the sixth in a series. The First International Cognitive Robotics Workshop was held in Orlando in 1998 as a AAAI Fall Symposium. Since then the Workshop has been held every two years, in locations alternating between Europe and North America. The second edition was in Berlin in 2000 as an ECAI workshop, the third in Edmonton in 2002 as a AAAI workshop, the fourth in Valencia in 2004 as an ECAI workshop, and the fifth in Boston in 2006 as a AAAI workshop.

The 2008 edition of the Workshop received 16 technical paper and 2 position paper submissions. Of these, 7 submissions were accepted for long presentations and 6 for short presentations. The accepted papers appear in these proceedings. We have an excellent program with papers on cognitive robot architecture, cognitive vision, learning for cognitive robotics, human-robot interaction, and systems addressing specific types of applications. We are also having two invited lectures, one by Anthony G. Cohn entitled “Qualitative Spatio-Temporal Representations and Cognitive Vision” and another by Jan Peters entitled “Motor Skill Learning for Cognitive Robotics”.

I would like to thank my partners in the Organizing Committee, Gerhard Lake-meyer, Jan Peters, and Fiora Pirri, who helped with many aspects of the organization, all members of the Program Committee, who did a great job reviewing the submitted papers in a short time window, as well as the ECAI workshop Co-Chairs Boi Faltings and Ioannis Vlahavas and the Local workshop and tutorials Co-Chairs Pavlos Peppas and Ioannis Hatzilygeroudis for their help and advice.

Yves Lespérance
Rome, June 2008
Towards Autonomous Design of Experiments for Robots

Timo Henne, Alex Juarez, Monica Reggiani and Erwin Prassler

Abstract. In order to understand a real-world environment on a conceptual level, any agent requires the capability for autonomous, open-ended learning. One of the main challenges in Artificial Intelligence is to bias the learning phase sufficiently in order to obviate complexity issues, while at the same time not restricting the agent to a certain environment or to a particular task. In this paper we describe a framework for autonomous design of experiments for a robotic agent, which enables the robot to improve and increase its conceptual knowledge about the environment through open-ended learning by experimentation. We specify our implementation of this framework and describe how its modules can recognize situations in which learning is useful or necessary, gather target-oriented data and provide it to machine learning algorithms, thus reducing the search space for the learning target significantly. We describe the integration of these modules and the real world scenarios in which we tested them.

1 INTRODUCTION

In Artificial Intelligence, numerous learning paradigms have been developed over the past decades. In most cases of embodied and situated agents, the learning goal for the artificial agent is to “map” or classify the environment and the objects therein [32, 27], in order to improve navigation or the execution of some other domain-specific task. Dynamic environments and changing tasks still pose a major challenge for robotic learning in real-world domains. In order to intelligently adapt its task strategies, the agent needs cognitive abilities to more deeply understand its environment and the effects of its actions. In order to approach this challenge within an open-ended learning loop, the XPERO project (http://www.xpero.org) explores the paradigm of learning by experimentation to increase the robot’s conceptual world knowledge autonomously. In this setting, tasks which are selected by an action-selection mechanism are interrupted by a learning loop in those cases where the robot identifies learning as necessary for solving a task or for explaining observations. It is important to note that our approach targets unsupervised learning, since there is no oracle available to the agent, nor does it have access to a reward function providing direct feedback on the quality of its learned model, as e.g. in reinforcement learning approaches.

In the following sections we present our framework for integrating autonomous robotic experimentation into such a learning loop. In section 2 we explain the different modules for Stimulation and Design of Experiments and their interaction. In section 3 we describe our implementation of these modules and how we applied them to a real world scenario to gather target-oriented data for learning conceptual knowledge. There we also indicate how the goal-oriented data generation enables machine learning algorithms to revise the failed prediction model.

2 A FRAMEWORK FOR DESIGN OF EXPERIMENTS

We propose a framework for designing experiments to be executed by a robotic learner which implements the paradigm of learning by experimentation. This framework integrates a Stimulation and a Design of Experiments component which interact by using available knowledge about the environment. The Design component consists of two parts which address the Exploration of the feature space and the actual Experimentation, which is the focus of our framework. It serves the purpose of designing and executing sequences of robot actions (experiments) in order to collect target-oriented data that afford learning new concepts. The output of the Experimentation module is thus intended to provide a machine learning algorithm with data in a format appropriate for learning conceptual knowledge.

![Diagram of the proposed framework for autonomous Design of Experiments.](image)

Figure 1. The proposed framework for autonomous Design of Experiments.

2.1 Available Knowledge

The components for Stimulation and Design of Experiments rely on information available to the autonomous agent. This information

---

1 T. Henne, A. Juarez and E. Prassler are with the Department of Computer Science, University of Applied Sciences Bonn-Rhein-Sieg, Germany; M. Reggiani is with the Dipartimento di Ingegneria e Gestione dei Sistemi Industriali, University of Padua, Italy; email: alex.juarez@fh-bonn-rhein-sieg.de
describes and relates robot actuation and sensing capabilities and knowledge about the environment, which was either learned at some earlier stage or provided as general a priori knowledge.

We define the *available knowledge* as the aggregate of the following information:

- \( A \), the set of *actions* that the robot can execute. Each action \( a_i \in A \) is defined on a set of parameters \( \theta_1, \theta_2, \ldots, \theta_n \) which must be assigned values before the actual execution of the action.
- \( F \), the set of *features* which are extracted from the robot’s sensor data. The robot is capable of transforming and processing the sensory information about the environment into internal representations that we call features. These features range from direct sensory data such as odometry and bumper signals, over object characteristics (e.g. color, size, pose) to more complex constructs such as predicates in first-order logic, e.g. position(object, X, Y, time).
- \( M \), the set of models of the worlds. These models represent the current beliefs of the agent on the effects that its actions have on the environment (e.g. “if a ball is hit with a force \( F \) at time \( t \), it will move with velocity \( v \) in direction \( d \) until it comes to a stop at a time \( t_2 \) with \( t_2 > t_1 \)”). The type of information expressed by these models requires a more abstract representation, such as first-order logic or qualitative models.

Any of these parts of the agent knowledge, but especially the set of models \( M \), are subject to revision and extension along the cognitive evolution process of the agent, facilitated by our proposed framework.

### 2.2 Design of Experiments

The process of discovering new concepts in robotics is still not well-defined in the state-of-the-art research literature. The existing solutions mostly focus on very specific domains. Consequently, no established general procedure exists which could be employed in determining the sequence of actions (experiment) that will successfully lead to an improvement of the agent’s knowledge. According to [30] the process of knowledge abstraction should involve four steps: act, predict, surprise, and refine. The structure of our framework follows these steps that have been implemented in three modules: *Exploration* (Sec. 2.3), *Stimulation* (Sec. 2.4), and *Experimentation* (Sec. 2.5).

The *Exploration* module organizes how the agent uses its current knowledge to *act*, i.e. it selects and executes actions \( A \) to achieve predefined goals or to explore its environment. At the same time the *Stimulation* module continuously observes the robot actions \( A \), extracts information about the environment as features \( F \) from sensor data and *predicts* the behaviour of these features by using the current knowledge, represented by models \( M \). When an unexpected phenomenon (a surprise) is observed, a signal is sent to the *Experimentation* block. The *Experimentation* module collects information about the experienced surprise through the selection of action sequences (experiment). These sequences are designed to provide relevant data to the learning module, thus starting a process of *revising* and *refining* the current model \( M \) in order to improve its predictive capabilities.

### 2.3 Investigating the environment: Exploration

The task of the *Exploration* module is to identify which paradigm will provide relevant information from the environment. As experimental paradigm we define a distinguishable class of experimental situations, i.e. distinctively different ways in which the agent investigates the environment [7]. The initial set of experimental paradigms \( P \) is built from the set of elementary actions \( A \) that the robot can execute. In later stages, the autonomous agent will try to produce new experimental paradigms, e.g. by combining known paradigms [7], also taking into account the cost and complexity of their execution.

Choosing the most suitable paradigm from \( P \) and the combination of its elements is a difficult task. One solution lies in applying a heuristic to choose an appropriate paradigm, taking into account the current knowledge, the costs of the experimental paradigms, and the Exploration goals(s).

In our framework we introduced three initial heuristics suggested by [29]. One heuristic \( H_{\text{goalSeeking}} \) chooses an experimental paradigm known to change a feature in \( F \) with the object of modifying its value with a certain relation to a target value. A second heuristic \( H_{\text{noiseEffect}} \) explores the paradigms that apparently have no effect on the environment. This heuristic aims at validating current beliefs on these paradigms, and tries to produce effects which had not been encountered previously. Finally, the heuristic \( H_{\text{random}} \) explores a randomly selected paradigm with randomly defined parameters. By applying these heuristics, we can guarantee that after a reasonable execution time, the system will have investigated even the paradigms which are not so promising, but that could still contribute to the creation of new models \( M \).

### 2.4 From Exploration to Experimentation: Stimulation

A central question within Learning by Experimentation is when to stop exploring the environment heuristically, and start the Design and Execution of Experiments. We believe that in order to facilitate autonomous, open-ended learning, the trigger of the Experimentation phase should be intrinsic, automatic, and at the same time related to the robot’s experience during the Exploration. In this work we propose the use of a robotic surprise mechanism to stimulate the Design of Experiments.

The application of artificial surprise in various fields such as evolutionary and developmental robotics, social agents, and human-machine interaction have shown the effectiveness and scalability of employing this concept. In the literature we can find examples of the integration of artificial surprise to active vision and adaptive learning [25, 26, 14, 13], as well as approaches to robot exploration of a partially unknown environment [21, 20, 19] and [24, 15]. These approaches share the idea that surprise is the result of an observation that diverges from an expectation.

The surprise mechanism used in this paper combines several elements from the mentioned approaches to artificial surprise and works under the assumption that the knowledge available to the robot can predict and explain any observation derived from the effect of the robot actions on the environment. To achieve this, each action \( a_i \in A \) is associated with one or more models \( m_i \in M \). If an action brings about an observation that diverges from the prediction offered by the associated model, this is considered as surprise.

The robot recognizes events that are candidate to surprise on two different levels of abstraction. The first level is directly related to the sensory input data and simulates a reflex to these events. At this level, the model is an estimation of the underlying probability distribution of the sensor data where such distribution is updated periodically as the robot executes its actions. The second level of abstraction uses available knowledge represented as first-order logic models or qualitative models to attempt an explanation of physical phenomena asso-
located to the execution of an action, for example the rolling of a ball after the robot has pushed it.

Prediction failure recognition at the sensor level is accomplished by estimating a probability distribution $P$ on the sensor data and comparing it to a model distribution $D$, which is associated with the originating action and constantly adapted using sensor information. While estimating the probability distribution two cases can occur: a) there is knowledge given a priori about the model distribution $D$, in which case the estimation of $P$ is done using histogram techniques; and b) there is no available knowledge about the model distribution $D$ in which case it is assumed that the sensor data follows a Gaussian distribution. The distribution $P$ is then locally estimated using a window of variable size and using maximum likelihood. In order to improve the accuracy of the model distribution as the robot executes its actions, Bayesian update is applied periodically to $D$.

A surprise measure is obtained by applying the Kullback-Leibler Divergence $KLD(P, D)$ to both probability distributions, in a similar approach to the one proposed in [14, 13]. The generic formulation of the KLD is given by formulas 1 and 2 for the discrete and continuous case. A derivation of KLD for the case of two Gaussian distributions is also shown in formula 3. This is especially useful in the case where the estimate of $P$ and $D$ is assumed to be Gaussian.

$$KLD(P, D) = \sum_x P(x) \log \frac{P(x)}{D(x)}$$ (1)

$$KLD(P, D) = \int_{-\infty}^{\infty} P(x) \log \frac{P(x)}{D(x)} dx$$ (2)

$$KLD(P, D) = \frac{1}{2} \left( \log \left( \frac{\sigma^2_P}{\sigma^2_D} \right) + \frac{\sigma^2_P}{\sigma^2_D} + \frac{\mu_D - \mu_P}{\sigma_D}^2 - 1 \right)$$ (3)

A large value of the metric produced by such measure indicates that an observation could not be explained by the current model distribution, in other words, the kind of surprise triggered by a disconfirmed passive expectation.

As mentioned before, the second level that recognizes prediction failures utilizes more complex knowledge using different knowledge representations, e.g. first order logic models or qualitative models. The abstraction from sensor data to first order logic representation is inspired by the ideas from [28, 10] and associates robot actions, predicates and sensor data with the intervention of a human expert. This solution will change into an automated mechanism as our framework evolves. In the case of qualitative models, the abstraction is performed by applying a temporal abstraction based on a sliding smoothing window that obtains the best fitting line for the sensor data, while controlling spurious noise by means of applying a hysteresis mechanism [17, 16]. This yields a state-based representation of increasing, decreasing or steady feature values, which is associated with a specific robot action.

The two mechanisms described are executed in parallel and independent from each other. They are, however, related by a subsumption principle based on the idea that more complex knowledge might be able to explain an event that seems surprising at the sensor level. Therefore, a surprise triggered from an unexpected change in the robot sensor data can be suppressed if there exists a model (e.g. a qualitative model) that can explain such observation. For example, an object that starts moving as a result of a robot action (e.g. PushObject) will cause a surprise at sensor level, caused by the perception of such object motion. If a qualitative model associated with the robot action exists such that it can correctly predict the perceived object motion (e.g. a ball rolling after being pushed by the robot), the perception can then be explained as it occurs and the surprise is subsumed.

Before the execution of an action, the models predicting its effects are loaded into memory. During execution, the sensor data is converted into the corresponding representation and compared online with these models. If the observation shows a divergence from the expected effect, a signal indicating a prediction failure is produced. This surprise can be characterized as a disconfirmed active expectation.

2.5 Experimentation

The Experimentation module receives a surprise signal from the Stimulation module whenever an observation diverges from the prediction. This signal contains information about the initial state of the environment as perceived by the robot, the experimental paradigm and the parameter values which generated the surprise, and the prediction rule which failed.

Due to the presence of noise in both perception and actuation, the occurrence of one surprising event is not enough to classify the paradigm as deserving attention. Therefore, the information from the Stimulation module is stored in a database for comparison with future surprises. For this, the agent must identify those features of the initial states and those distinctive parameters of the experimental paradigms which were relevant to the prediction failure, in order to avoid storing too much redundant or irrelevant data.

Early attempts to form equivalence classes in collected data can be found in [9] using k-means clustering algorithm and Support Vector Machines to define affordance relations and in [22]. While these approaches attempt to directly identify the final relations that will be part of robot knowledge, our goal here is instead to provide a heuristic that can drive the agent in the Experimentation phase. The correct identification of the initial situation and paradigm can reduce the search space for learning algorithms significantly, which is critical for the task of learning high-level concepts, such as models in first-order logic. Errors in this identification process will most probably result in an ineffective learning phase, but the overall correctness of the framework is not affected.

An additional improvement of the framework, and part of our future work, is a mechanism to define the importance of the stored surprises. As suggested in [31], Exploration may be achieved by selecting actions and/or states which have been selected less frequently or less recently.

The importance of a surprise can therefore be inversely related either to its age or to its recency, i.e. the time during which a surprise did not occur. In the first case, the importance of visiting a particular state tends to vanish over time, while the latter case favors older surprises.

3 FRAMEWORK IMPLEMENTATION

Starting from the description presented in Section 2, we developed a first implementation of the proposed framework. As several activities are required to run concurrently, we built a service-based infrastructure based on the ICE middleware [12]. Each module (Exploration, Stimulation, and Experimentation) was implemented as a distributed service that can publish data for the interested services (subscribers) and collect information from connected modules. Additionally a FeatureExtraction service gathers data from the different publishers and
provides them to a learning algorithm, while a Manager service monitors the state of the overall system to enable and disable the different services as required.

The implemented system was designed to be easily adaptable to new experimental setups. We built a library with immutable parts (the framework algorithms) and clearly defined the interfaces where the description of the available knowledge and the interface with the real robot sensors and actuators can be provided, depending on the current setup. Thus, different robots and sensor types can be integrated without affecting the overall framework.

### 3.1 Application to a real scenario

To validate the framework we used the library described above to support the collection of data within the showcase outlined in the XPERO project.

This showcase features a robot located in an almost empty room with boxes blocking the room’s exit. Although for a human programmer the solution to this task is straightforward, this scenario still presents a major challenge for the currently available unsupervised learning paradigms. Autonomously learning expressive knowledge to solve this task requires a cognitive evolution of theories, i.e. evolving from simple notions to increasingly complex ones.

In this work, we focused on the first two concepts that the agent should learn in the XPERO evolution of theories. For this the robot is situated in a free space with static and movable objects (Figure 2). The notions which can be learned in these subscenarios include the notion of static vs. movable objects and the notion of obstacles, i.e. that under certain circumstances objects prevent a robot from moving to certain places or in certain directions.

We chose to encode the set of actions $\mathcal{A}$ with their parameters $\theta_i$, the features $\mathcal{F}$, and the prediction rules defining the model $\mathcal{M}$ in simple XML syntax. This facilitates both the initial process of encoding the starting knowledge and the autonomous revision of the world knowledge with learned insights. Four elementary actions were implemented to define the initial set of paradigms $\mathcal{P}$. Tables 1 and 2 present these actions and their parameters, respectively.

<table>
<thead>
<tr>
<th>ActionId</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>goInContact objectId</td>
</tr>
<tr>
<td>$a_2$</td>
<td>pushObject objectId, pushDistance</td>
</tr>
<tr>
<td>$a_3$</td>
<td>moveForward distance angle xCoord, yCoord, orientation</td>
</tr>
<tr>
<td>$a_4$</td>
<td>rotate angle</td>
</tr>
<tr>
<td>$a_5$</td>
<td>goTo xCoord, yCoord, orientation</td>
</tr>
</tbody>
</table>

**Table 1.** Actions $\mathcal{A}$.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>objects in the environment</td>
</tr>
<tr>
<td>pushDistance</td>
<td>[minPushDist, maxPushDist]</td>
</tr>
<tr>
<td>distance</td>
<td>[minDist, maxDist]</td>
</tr>
<tr>
<td>xCoord</td>
<td>[-10.0, 10.0]</td>
</tr>
<tr>
<td>yCoord</td>
<td>[-10.0, 10.0]</td>
</tr>
<tr>
<td>orientation</td>
<td>[-π, π]</td>
</tr>
</tbody>
</table>

**Table 2.** Parameters.

The predictive rules in $\mathcal{M}$ available to the robot were encoded in first order logic. For the subscenarios two of them were used, given below in their Prolog notations and their associated actions:

1. $a_2$: move(Object, Start, Dist, End) :- approxEqual(Start, End).
2. $a_3$: move(Robot, Start, Dist, End) :- approxAdd(Start, Dist, End).

Their meaning is straightforward:

1. When the robot executes $a_2$ on a certain object, its model predicts that the end position of the object will be approximately equal to its initial position. This knowledge cannot explain cases when the robot tries to push objects that can actually be moved, so here a prediction failure is sent to the Experimentation module, including the object involved, the action that was executed and the prediction rule that failed.
2. For the execution of action $a_3$, the model predicts that the robot end position will be approximately equal to the sum of the robot start position and the distance parameter of the action. This prediction fails for cases when the robot bumps into non-movable objects on its path. Again, a surprise signal is generated which will prompt the Experimentation module to again identify relevant features that distinguish this case from the ones where the prediction rule does not fail.

The setting we used for the real world experiments was an Eddy robot (see [3]) located in a free space with four colored boxes (see figure 2), of which two were movable and two not movable for the robot. An overhead camera provided the object IDs and localisation information about the robot and the objects by means of color blob tracking.

![Figure 2. The real environment used in our experiments.](image)

Upon receiving a surprise signal, the Experimentation module autonomously designs an experiment by selecting an appropriate paradigm, defining the initial states for the environment, and choosing the paradigm parameters that efficiently cover the experimental domain as shown in [8]. Once the experiment has been designed, a planner produces the sequence of actions that the robot will perform to execute the experiment. Defining a plan for a robot is a hard problem, and it is not entirely clear which planning techniques serve the problem best. We are currently using neoclassical planners that assume to have deterministic robot actions and a complete description of the world. We mainly evaluated SHOP2 [23], a HTN (Hierarchical Task Networks) planner already successfully applied in several robotic applications [11, 1, 2]. The main drawback in using classical planners is that the handling of exceptions is often left to the developer. In most cases an execution engine needs to be implemented,
which is able to perform additional recovery actions during the execution of the plan.

We have also investigated an alternative solution based on non-classical planners [4, 5], a promising area of research but with few available planners, often simply at a prototype level. The goal of these planners is to deal with partial or non-observability of the state, non-deterministic domains and complex goals, and they are intended to solve a large amount of realistic practical planning problems. Their applicability is nevertheless limited by the quantity and quality of sensory information they are able to deal with. Therefore planning tools that we have tested produce solutions only in simple cases and force us to rely on classical planners whenever their application is not possible.

As a simple example, consider the case where the robot executes action a2 (pushObject) while interacting with a movable object, encountering a surprise and triggering the Design of Experiments. The framework was able to correctly identify the action generating the surprise and to design a new experiment to explore the action parameters, i.e. the object in the environment and the distance for the pushing actions, each time starting from a new robot pose. An experimental trace logged during an experimentation phase is shown in Figure 3(b). The execution of the experiments gathers the data necessary to learn a new model, which is able to correctly explain the observations made by the robot when trying to push an object, regardless of it being movable or not. Figure 3(a) depicts the simulated execution path of the plan covering several experiments by a robot in the environment previously described.

As intended by our framework design, we provided the specifically targeted data generated in our experiments in a real environment to HYPER, a machine learning tool for inductive logic programming (ILP) [6]. In order to discover the desired concepts in the form of predicates, HYPER was extended to facilitate predicate invention. However, when dealing with data generated by unspecific robot actions, HYPER is not able to derive any concept, since the amount of data and possibly significant variables inevitably led to combinatorial explosion problems. Here our framework proved as a useful bias for revising the prediction model under question, since it focuses on generating data for the action whose prediction model failed and produces data in a predicate form, thus limiting the number of variables to be investigated by the ILP algorithm. With these data, HYPER was able to learn the concepts of movable objects and obstacles [18], which could not be achieved with data from unspecific robot actions.

4 CONCLUSION AND FUTURE WORK

In this paper, we presented ongoing work on a framework for integrating targeted data generation for robotic learning by experimentation. We explained how the different modules Stimulation, Exploration and Experimentation work together to enable an intrinsically motivated, reasonable and autonomous switch from task execution to experimentation. Subsequently we showed how the data collection in the Experimentation phase is guided by the heuristic applied in the Exploration phase, and by the robotic surprise from the Stimulation module. We described how the implemented framework library allows for a simple exchange of robots, sensors and scenarios. By applying our framework to a real world scenario, we were able to show its feasibility and demonstrate how purposeful data generation takes place which enables a learning algorithm to discover conceptual knowledge.

Our current work focusses on exploring other learning algorithms and evaluate the effect of both the quality of the experiment design, and the number of the experiments performed, on the prediction accuracy of the revised model. Furthermore, we are further developing the automation of generating and evaluating experimental paradigms, and exploring other knowledge representations such as qualitative models, and test our framework with different scenarios.

ACKNOWLEDGEMENTS

This work has been funded by the European Commission’s Sixth Framework Programme under contract no. 029427 as part of the Specific Targeted Research Project XPERO ("Robotic Learning by Experimentation").

REFERENCES

Fred meets Tweety
Antonis Kakas\(^1\) and Loizos Michael\(^2\) and Rob Miller\(^3\)

Abstract. We propose a framework that brings together two major forms of default reasoning in Artificial Intelligence: applying default property classification rules in static domains, and default persistence of properties in temporal domains. Particular attention is paid to the problem of qualification, central in default reasoning and in any attempt to integrate different forms of this type of reasoning. We examine previous semantics developed independently for the two separate forms of default reasoning, and illustrate how these naturally lead to the solution that we propose in integrating the two. The resulting integration gives rise to domains where four different types of knowledge interact and qualify each other in an intricate manner. Through a series of examples we show how this knowledge qualification leads to intuitive conclusions. We prove that our framework of integration is elaboration tolerant: extending a consistent domain with additional information generates a consistent domain with additional information. Section 1 examines the qualification problem as studied in the two major forms of default reasoning. It depends on whether Tweety can fly or not! If all we know about Tweety is that it is a bird, we then expect to see it flying, but if we also know that it is a penguin we will not expect to see it flying, even if we hear a loud noise produced by the act of firing. What can we conclude if after the act of shooting we observe that Tweety is still on the ground? That Tweety is not a typical bird, or that the gun did not make a loud noise when fired, or even that the gun was not loaded at the time of shooting? Can we indeed conclude anything at all after such an unexpected observation?

In this problem of “Fred meets Tweety” we need to bring together two major forms of default reasoning that have been extensively studied on their own in Artificial Intelligence, but have rarely been addressed in the same formalism. These are default property classification as applied to inheritance systems [5, 10], and default persistence central to temporal reasoning in theories of Reasoning about Action and Change (RAC) [4, 9, 11]. How can a formalism synthesize the reasoning encompassed within each of these two forms of default reasoning?

Central to these two (and indeed all) forms of default reasoning is the qualification problem: default conclusions are qualified by information that can block the application of the default inference. One aspect of the qualification problem is to express within the theory the knowledge required to properly qualify and block the default inference under exceptional situations. This endogenous form of qualification is implicit in the theory, driven by auxiliary observations that enable the known qualifying information to be applied. For example, known exceptional classes in the case of default property inheritance, or known action laws (and their ramifications) in the case of default persistence, qualify respectively these two forms of default reasoning.

But this task of completely representing within a given theory the qualification knowledge is impractical and indeed undesirable, as we want to jump to default conclusions based on a minimal set of information available. We, therefore, also need to allow for default conclusions to be qualified unexpectedly from observed information that is directly (or explicitly) contrary to them. In this exogenous form of qualification the theory itself cannot account for the qualification of the default conclusion, but our observations tell us explicitly that this is so and we attribute the qualification to some unknown reason.

Recent work [6, 12] has shown the importance for RAC theories to properly account for these two forms of qualification, so that an exogenous qualification is employed only when observations cannot be accounted for by an endogenous qualification of the causal laws and default persistence. In our problem of integrating the default reasoning of property classification into RAC, this means that we need to ensure that the two theories properly qualify each other endogenously, so that the genuine cases of exogenous qualification can be correctly recognized. In particular, we study how a static default theory expressing known default relationships between fluents can endogenously qualify the reasoning about actions and change, so that the application of causal laws and default persistence is properly adjusted by this static theory. In the Fred meets Tweety scenario described above, for example, the normal default that “penguins cannot fly” would act as an implicit qualification for the causal law that “a loud noise causes birds to fly”, but not so when either Tweety is not known to be a penguin, or it is known to be a super-penguin (super-penguins being an exception to the default that penguins cannot fly).

More generally, we study how four different types of information present in such an integrated framework of RAC interact and qualify each other: (i) information generated by default persistence, (ii) action laws that qualify default persistence, (iii) static default laws of fluent relationships that can qualify these action laws, and (iv) observations that can qualify any of these. This hierarchy of information comes full circle, as the bottom layer of default persistence of observations (which carry the primary role of qualification) can also qualify the static theory. Hence, in our proposed integrated framework, temporal projection with the observations help to determine the admissible states of the static default theory. In turn, admissible states qualify the actions laws and the temporal projection they generate.

Section 2 examines the qualification problem as studied in the two separate domains and its form for the proposed integration. Section 3 gives the formal semantics of the integration framework and the central result that ensures its elaboration tolerance. Section 4 briefly dis-

\(^1\) University of Cyprus, P. O. Box 20537, CY-1678, Cyprus. e-mail: antonis@ucy.ac.cy
\(^2\) Harvard University, Cambridge, MA 02138, U.S.A. e-mail: loizos@eecs.harvard.edu
\(^3\) University College London, London WC1E 6BT, U.K. e-mail: rsm@ucl.ac.uk
cusses related and future work.

2 Knowledge Qualification

Through a series of examples, we present in this section the issues that arise when examining the qualification of knowledge, and place in context the various problems and solutions considered so far. We remark that we generally use the term qualification in a broader sense than that used in the context of Reasoning about Action and Change.

Here and throughout the paper we employ the syntax of the action description language $\mathcal{ME}$ [6] for temporal domain descriptions, and a pseudo-syntax based on that of propositional logic for representing static theories describing default or strict domain constraints. Strict static knowledge is represented in propositional logic. Default static knowledge is represented in terms of default rules of the form $\phi \rightsquigarrow \psi$, where $\phi, \psi$ are propositional formulas. In this pseudo-syntax we specify the relative strength between two default rules by statements of the form “rule (i) overrides rule (j)”. Formulas which contain variables are a shorthand representation of all formulas obtained by substituting the variables over a finite domain of constants.

We do not reproduce here the formal syntax for these theories. In particular, the formal semantics of our approach, given in the next section, will not depend on the specific form of the static theories, and different frameworks such as Default Logic [10] or argumentation [1] can be used. In this section it is sufficient for the reader to use the informal reading of the theories for their semantics.

One of the first knowledge qualification problems formally studied in A.I. relates to the Frame Problem (see, e.g., [11]) of how the causal change properly qualifies the default persistence; see Figure 1(a). In the archetypical Yale Shooting Problem domain [4], a turkey named Fred is initially alive, and one asks whether it is still alive after loading a gun, waiting, and then shooting Fred. The lapse of time cannot cause the gun to become unloaded. Default persistence is qualified only by known events and known causal laws linked to these events.

The consideration of richer domains gave rise to the Ramification Problem (see, e.g., [7]) of how indirect action effects are generated and qualify persistence; see Figure 1(b). Static knowledge expressing relationships (or domain constraints) between different properties was introduced to encode these indirect effects. Then, in early solutions to the Ramification Problem a direct action effect would cause this static knowledge to be violated, unless a minimal set of indirect effects were also assumed in order to maintain consistency [7, 8]. Thus, given the static knowledge that “dead birds do not walk”, the shooting action causing Fred to be dead would also indirectly cause Fred to stop walking, thus qualifying the persistence of Fred walking.

Subsequent work examined default causal knowledge, bringing to focus the Qualification Problem (see, e.g., [12]) of how such default causal knowledge is qualified by domain constraints; see Figure 1(c). In some solutions to the Qualification Problem, static knowledge within the domain description was identified as the knowledge that endogenously qualified causal knowledge, as opposed to as an aid to causal knowledge in qualifying persistence [6]. The Ramification Problem was now addressed by the explicit addition of causal laws, and the development of a richer semantics to account for their interaction. The following example domain illustrates a typical case.

Fix a model implying “GunBroken holds-at 1”. Then we reason that the static theory (of domain constraints) qualifies the direct effect of the action “Shoot(Fred)” on “FiredAt(Fred)”, and hence it also prevents the indirect effect “¬Walks(Fred)” from being triggered. Thus, the default persistence of Fred walking is not qualified, and we conclude that Fred keeps walking. If, on the other hand, a model implies “¬GunBroken holds-at 1”, then neither causal law is qualified by the static theory. Note that the effect “¬Alive(Fred)” is not qualified despite the observation “Walks(Fred) holds-at 1”; the causal knowledge “¬Alive(Fred) causes ¬Walks(Fred)” provides an escape route to this qualification. Hence, the default persistence of “Walks(Fred)” is qualified, and Fred is not walking after time-point 2. Models derived according to either of the two cases are valid.

Perhaps the next natural step was realizing that observations after action occurrences also qualify causal change when the two conflict, a problem known as the Exogenous Qualification Problem (see, e.g., [6]; see Figure 1(d)). Consider, for example, the previous domain extended by the observation “¬FiredAt(Fred) holds-at 4”. Even though the effect of the “Shoot(Fred)” is not, as we have seen, necessarily qualified by the static theory alone, the explicit observation that the action’s direct effect is not produced leads us to conclude that it was necessarily qualified. The interaction with the endogenous qualification of the causal laws by the static theory comes from the fact that “GunBroken” together with the static theory qualifies the action law, and provides, thus, an explanation of the observed action failure. So, if we wish to minimize the unknown exogenous cases of qualification, we would conclude that “GunBroken” holds, as this is the only known way to endogenously account for the observed failure.

Independently of the study of qualification in a temporal setting, another qualification problem was examined in the context of Default Static Theories [10] that consider how observed facts qualify default static knowledge; see Figure 1(f). In the typical domain, represented below, one asks whether a bird named Tweety has the ability to fly, when the only extra given knowledge is that Tweety is a bird.

\[
\begin{align*}
\text{static theory:} \\
\text{Bird(Tweety)} & \rightrightarrows \neg\text{CanFly(x)} \\
\text{Penguin(x)} & \rightarrow \text{Bird(x)} \\
\text{Bird(x)} & \rightrightarrows \text{CanFly(x)} \\
\text{rule (1) overrides rule (3)}
\end{align*}
\]

In the absence of any explicit information on whether Tweety has the ability to fly, the theory predicts “\text{CanFly(Tweety)}”. Once extended with the fact “\text{Penguin(Tweety)}”, however, “\text{CanFly(Tweety)}” is retracted. The same happens if instead of “\text{Penguin(Tweety)}”, the fact “\text{CanFly(Tweety)}” is added. In either case the static theory is qualified, and yields to explicit facts or stronger evidence.

2.1 Putting Fred and Tweety in the Same Scene

In this paper we investigate temporal domains that incorporate (possibly) default static theories. The technical challenge lies in understanding how the four types of knowledge in a domain, three of which may now be default, interact and qualify each other; see Figure 1(e).

We view observations as part of the non-defeasible part in static default theories, thus primarily taking the role of qualifying the static knowledge, which then in turn will qualify the causal knowledge as described above. Due to the temporal aspect of a domain, however, a point-wise interpretation of observations as facts in the static default theory is insufficient, even in domains with no causal laws and, thus, strict persistence. Consider a temporal domain with the observations “\text{Penguin(Tweety) holds-at 1}” and “\text{Bird(Tweety) holds-at 4}”,

\[
\begin{align*}
\text{Shoot(Fred) occurs-at 2} \\
\text{FiredAt(Fred) holds-at 1} \\
\text{Alive(Fred) holds-at 1} \\
\text{Walks(Fred) holds-at 4} \\
\text{Bird(Tweety) holds-at 1} \\
\text{CanFly(Tweety) holds-at 4} \\
\end{align*}
\]
and a static theory as in the Tweety example above. By viewing each time-point in isolation, we can only conclude that “CanFly(Tweety)” holds at time-point 4, but not at time-point 1. This cannot be extended into a temporal model without violating the (strict) persistence. Instead, “Penguin(Tweety) holds-at 1” should persist everywhere, as if “Penguin(Tweety)” was observed at every time-point. These virtual (or assumed) observations then qualify the static theory at every time-point, implying “¬CanFly(Tweety)”. Analogously, if the observation “CanFly(Tweety) holds-at 7” is included in the domain, the observation persists everywhere and qualifies the default conclusion of the static theory that the penguin Tweety cannot fly.

Assume, now, that observations and persistence have appropriately qualified the static theory at each time-point T, so that the theory’s default extensions (models) determine the set of admissible states at T. Through these sets of admissible states, the qualified static knowledge then qualifies the change that the theory attempts to generate through its causal knowledge. Given a time point T, it is natural that causal knowledge will be qualified by admissible states as determined immediately after T. This is illustrated in the next domain.

\[
\begin{align*}
\text{ClapHands} & \text{ causes Noise} \\
\text{Noise} & \text{ causes Fly(x)} \\
\text{Noise} & \text{ causes } \neg\text{Noise} \\
\text{Spell(x)} & \text{ causes CanFly(x)} \\
\text{Penguin(Tweety)} & \text{ holds-at 1} \\
\text{ClapHands} & \text{ occurs-at 3} \\
\text{Spell(Tweety)} & \text{ occurs-at 5} \\
\text{ClapHands} & \text{ occurs-at 7}
\end{align*}
\]

The default persistence of “Penguin(Tweety) holds-at 1” implies that “¬CanFly(Tweety)” holds in each set of admissible states up to time-point 5. In particular, this conclusion holds immediately after “ClapHands occurs-at 3”, and qualifies through the static theory the causal generation of “Fly(Tweety)” by the action “ClapHands”.

Intuitively, we expect “Spell(Tweety) occurs-at 5” to override the static theory’s default conclusion “¬CanFly(Tweety)” from holding at time-points following time-point 5. Note, however, that up to now we have assumed that the static default theory is stronger than the causal knowledge, and that it qualifies any change implied by the latter. But this is not the case now, since we wish to specify that some causal information is stronger than the static default theory. How, then, can we ensure that the causal generation of “CanFly(Tweety)” by “Spell(Tweety)” will not be qualified in this particular case?

This requirement is accommodated by including the particular causal law of interest “Spell(x) causes CanFly(x)” as a default rule “Spell(x) \rightarrow CanFly(x)” in the static theory, and giving this rule priority over other default rules of the static theory with the contrary conclusion. The action occurrence “Spell(Tweety)” is also automatically included as a fact in the default theory, so that together with the default rule they imply “CanFly(Tweety)”. This conclusion holds in the set of admissible states associated with the time-point at which the action “Spell(Tweety)” occurred, namely time-point 5, which then allows the action’s effect “CanFly(Tweety)” to override the static theory’s usual default conclusion “¬CanFly(Tweety)”.

Such “strong” actions5 (like “Spell(x)” take the world out of the normal default state (where penguins cannot fly) into an exceptional, from the point of view of the static theory, state (where Tweety can fly). The rest of the default conclusions of the static theory still apply in this exceptional state (following time-point 5), conditioned on the exception (that Tweety can fly) that the “strong” action has brought about. This exception holds until some later action occurrence (of “UndoSpell(Tweety)” brings the world back into its normal state. In our domain, then, the action “ClapHands occurs-at 7” is not qualified, and Tweety (a penguin able to fly) flies after time-point 7.

Consider now replacing “Spell(Tweety) occurs-at 5” in the domain above with the observation “Fly(Tweety) holds-at 5”. By persistence, this observation qualifies the static theory so that “Fly(Tweety)” holds in each set of admissible states at time-points strictly after 3. Note that it is not known how the static theory is qualified, but only that it is somehow exogenously qualified. This does not hold for time-points up to including time-point 3, since the occurrence of the action “ClapHands” at time-point 3 can now account for the change from “¬Fly(Tweety)” by qualifying its persistence, as the static theory does not now qualify “ClapHands occurs-at 3”. Note that the interpretation of “Fly(Tweety) holds-at 5” is that Tweety flies for some exogenous reason (e.g., it is on a plane). If an action at time-point 6 were to cause Tweety to stop flying, then this would release the static theory’s default conclusion that penguins do not fly, so that the subsequent action “ClapHands occurs-at 7” would be qualified and would not cause Tweety to fly again.

A somewhat orthogonal question to the one of when causal knowledge is qualified by the static theory, is that of how this qualification happens. Assume we wish to know if Fred is alive after firing at it. In the following domain one concludes that Fred is dead from time-point 2 onwards, and also that Tweety is flying. What happens, however, if one observes “¬Fly(Tweety) holds-at 4”? Can one still conclude that Fred is dead? Interestingly enough, the answer depends on why Tweety did not fly after Fred was shot! The observation by it-

---

4 We remind the reader that our goal here is not to provide semantics for static theories, and that using an informal reading suffices for their semantics.

5 “Strong” actions are domain-dependent, and it is the domain designer’s task to identify them and to extend the static theory with appropriate extra rules.
self does not explain why the causal laws that would normally cause Tweety to fly were qualified.

\[
\text{Shoot}(x) \text{ causes FiredAt}(x) \quad \text{FiredAt}(x) \text{ causes } \neg \text{Alive}(x) \quad \text{Noise}\text{ causes Fly}(x) \quad \text{Noise causes } \neg \text{Noise} \quad \text{Alive(Fred) holds-at } 1 \quad \text{Turkey(Fred) holds-at } 1 \quad \text{Bird(Tweety) holds-at } 1 \quad \text{Shoot(Fred) occurs-at } 2
\]

An exogenous explanation would be that Tweety is a penguin, and “Fly(Tweety)” is qualified from being caused. An exogenous explanation would be that Tweety could not fly due to exceptional circumstances (e.g., an injury). In either case we would presumably conclude that Fred is dead. However, Tweety might not have flown because the shooting action failed to cause a noise, or even because the shooting action failed altogether. Different conclusions on Fred’s status might be reached depending on the explanation.

3 Formal Semantics of Integration

Due to the cyclical nature of the qualifications amongst different types of knowledge, we develop the formal semantics in two steps, starting from the temporal semantics. Thus, we will start by assuming that the static theory is somehow qualified, and do not, for now, examine how this is achieved. This effectively breaks the cycle of qualifications, and reduces Figure 1(e) to Figure 1(c). We will then base our semantics on that of FL [6], from which we borrow the syntax.

A state is a complete and consistent set of positive or negative fluent literals in our problem domain language. A state change is a pair of states, comprised of an initial state, and a resulting state.

Definition 1 (Causal Node) A causal node (or simply node) is a tuple \(N = (S, B, P)\), where \(S\) is a state, \(B\) is a set of action constants, and \(P\) is an active process log. Let \(A\) be a set of state changes. A pair \((\langle S_1, B_1, P_1 \rangle, \langle S_2, B_2, P_2 \rangle)\) of causal nodes is an admissible change under \(A\) iff \(\langle S_1, S_2 \rangle\) is a state change in \(A\).

Consider the domain description \(D^*\) of the last example in the previous section, which will serve as a running example in this section. Intuitively, then, one possible causal node associated with time-point 2 in \(D^*\) is \(N_0 = (S_0', B_0', \emptyset)\), where \(B_0' = \{\text{Shoot(Fred)}\}\), and the literals \(\neg \text{FiredAt(Fred)}, \text{Alive(Fred)}, \text{Turkey(Fred)}, \neg \text{Noise}, \text{Bird(Tweety)}\) are amongst those satisfied by (or belonging to) \(S_0'\).

A process proc(L) is triggered at a causal node \((S, B, P)\) w.r.t. \(D\) iff the body \(C\) of a causal law “\(C\ causes L\)” holds in \(S \cup B\). When the literal \(L\) is positive \(F\) (resp., negative \(\neg F\)), the triggered process proc(F) = \(|F|\) (resp., proc(\neg F) = \(|\neg F|\)) is initiating (resp., terminating).

All processes \(P_1\) triggered at a causal node \((S, B, P)\) become part of the active process log, and the process successor of \((S, B, P)\) is the (unique) causal node \((S', B', P \cup P_1)\). In the example \(D^*\) above, the processes \(\text{FiredAt(Fred)}\) and \([\text{Noise}\) are (the only ones) triggered at \(N_0\) w.r.t. \(D^*\), since the action constant in the bodies of the causal laws “\(\text{Shoot}(x)\) causes \text{FiredAt}(x)\)” and “\(\text{Shoot}(x)\) causes \text{Noise}\)” belongs in \(B_0'\) when \(x = \text{Fred}\). Thus, the process successor of \(N_0\) w.r.t. \(D^*\) is \(N_1 = (S_1', B_1', \emptyset, \{\text{FiredAt(Fred)}, \neg \text{Noise}\})\).

Processes in the active process log get resolved. A causal node \((S', B', P')\) is a resolvent of a causal node \(N = (S, B, P)\) iff either \(i) S' = S \land P' = P \land \emptyset, \lor (ii) P' \subset P, \land S' \text{ differs from } S \text{ on exactly those fluents in } P \setminus P', \) and is such that it satisfies \(F\) (resp., \(\neg F\)) when an initiating (resp., terminating) process for \(F\) is in \(P \setminus P'\). Any non-empty subset of the processes can be resolved in a single step, so that multiple resolvers might be obtained. This captures the possibly asynchronous resolution of processes — unresolved processes remain in the process log and get resolved later. In our example, \(N_2 = (S_2', B_2', \{\text{Noise}\})\) is one of the resolvents of \(N_1\), where \(S_2'\) differs from \(S_0'\) only in that it satisfies \(\text{FiredAt(Fred)}\).

Definition 2 (Causal Chain) Let \(D\) be a domain description, \(A\) a set of state changes, and \(N_0\) a causal node. A causal chain rooted at \(N_0\) w.r.t. \(D\) is a (finite) sequence \(N_0, N_1, \ldots, N_n\) of causal nodes such that for each \(k: 0 \leq k \leq n - 1\), \(N_1k+1\) is a process successor of \(N_{2k}\) w.r.t. \(D\) and \(N_{2k+2}\) is a resolvent of \(N_{2k+1}\), and such that every resolvent of the process successor of \(N_n\), has the same state as \(N_{2n}\). A causal chain \(N_0, N_1, \ldots, N_n\) is admissible under \(A\) up to \(N_{2n}\) iff the pair \((N_{2(1-1)}, N_{2j})\) of causal nodes is an admissible change under \(A\) for every \(j: 1 \leq j \leq n\), and either (i) \(j = n\); or (ii) \((N_{2j}, N_{2(k+1)})\) is not an admissible change under \(A\). In the former case the causal chain is fully admissible under \(A\).

Causal chains capture, thus, the triggering and resolution of (indirect) effects, until the state stabilizes. One causal chain rooted at \(N_0\) w.r.t. \(D^*\) is \(N_0, \ldots, N_n\), where: \(P_2 = \{\text{Noise}, \neg \text{Alive(Fred)}\}; S_2'\) differs from \(S_2\) only in that it satisfies \(\text{Noise}, \neg \text{Alive(Fred)}\); \(S_2'\) differs from \(S_2\) only in that it satisfies \(\neg \text{Noise}, \text{Fly(Fred)}, \text{Fly(Tweety)}\). The causal chain does not continue further; the process successor \(N_2\) of \(N_0\) contains in its process log \(P_2\) only the process \(\text{Alive(Fred)}\), and all resolvants of \(N_2\) have the same state as \(N_2\).

Each causal chain corresponds to a possible evolution path of the state of affairs at a fixed time point, as implied by a domain’s causal knowledge. The static knowledge determines, through the notion of admissible change that it defines, whether a change between consecutive states in an evolution path is indeed allowed. If all possible evolution paths contain a non-admissible change, then the static theory suggests that the causal knowledge of the domain is flawed, and that the evolution of the state of affairs has stopped at an unknown point before reaching a non-admissible change (Condition (ii) below).

Definition 3 (Proper Causal Descendant) Let \(D\) be a domain description, \(A\) a set of state changes, and \(N_0, N_2\) two causal nodes. \(N\) is a proper causal descendant of \(N_0\) w.r.t. \(D, A\) iff either:

(i) there exists a causal chain \(N_0, \ldots, N_2\) rooted at \(N_0\) w.r.t. \(D\) that is fully admissible under \(A\) such that \(N = N_2\); or

(ii) there exists no causal chain rooted at \(N_0\) w.r.t. \(D\) that is fully admissible under \(A\), and there exists \(k: 0 \leq k \leq n - 1\) and a causal chain \(N_0, \ldots, N_n\) rooted at \(N_0\) w.r.t. \(D\) that is admissible under \(A\) up to \(N_{2n}\) such that \(N = N_{2j}\) for some \(j: 0 \leq j \leq k\).

It can be verified that the causal node \(N_0\) defined earlier is contained in each causal chain rooted at \(N_0\) w.r.t. \(D^*\), with \(S_0\) satisfying, amongst others, the literals \(\text{Fly(Fred)}\) and \(\text{Turkey(Fred)}\). Intuitively, the set \(A\) of state changes that corresponds to the static theory of \(D^*\) includes no state change with a resulting state that simultaneously satisfies \(\text{Fly}(x)\) and \(\text{Turkey}(x)\). Hence, no pair \((N^*, N^b)\) is an admissible change under \(A\), and, thus, no causal chain rooted at \(N_0\) is fully admissible under \(A\). So, Condition (ii) of Definition 3 is used.

We define now the temporal projection component of the semantics. Let \(\Pi\) be the set of time-points, and \(\Phi\) the set of positive or negative fluent literals in the language. We assume an initial time-point \(T_{in} \triangleq \min(\Pi)\), but do not assume discreteness or total ordering. Let \(\overline{I}\) denote the negation of \(I \in \Phi\); thus, if \(L = \neg F\), then \(\overline{L} = F\).
An interpretation $H$ is a mapping of each fluent at each time-point to a truth-value. The state $S(H, T)$ at $T$ w.r.t. $H$ is the restriction of $H$ to the time-point $T$. The event base $B(D, T)$ at $T$ w.r.t. $D$ is the set of action constants $\{ A \mid "A\ occurs-at-T" \in D \}$. An admissibility requirement $\alpha$ maps each time-point to a set of state changes.

A state $S$ is stable in $H$ at $T$ w.r.t. $D$ under $\alpha$ if there exists a proper causal descendant $\langle S, G, P \rangle$ of $\langle S, \emptyset, \emptyset \rangle$ w.r.t. $D$ under $\alpha(T)$. So, stable states do not spontaneously change, and take into account the causal knowledge and the admissibility requirements — the effects of any processes that could have been triggered have already been taken into account, so that no other change is "pending". We ask that the initial state at $T_i\in$ in a temporal model of $D$ satisfies this requirement. The change that occurs at each time-point $T$ is determined by a proper causal descendant $\langle S, G, P \rangle$ of $\langle S(H, T), B(D, T), \emptyset \rangle$ w.r.t. $D$ under $\alpha(T)$. The change $S \setminus S(H, T)$ that is brought about in the state of affairs is a change set of $H$ at $T$ w.r.t. $D$ under $\alpha$.

Definition 4 (Externally Qualified Model) Let $D$ be a domain description, $H$ an interpretation, $c : \Pi \rightarrow 2^\Phi$ a mapping, and $\alpha_{st}, \alpha_{ch}$ two admissibility requirements. $H$ is an externally qualified model of $D$ under $(\alpha_{st}, \alpha_{ch})$ supported by $c$ iff the following hold:

1. $S(H, T_i)$ is stable w.r.t. $D$ under $\alpha_{st}(T_i)$;
2. for every $T \in \Pi$, $c(T)$ is a change set of $H$ at $T$ w.r.t. $D$ under $\alpha_{ch}$;
3. for every $L \in \Phi$, and every $T_1, T_2 \in \Pi$ s.t. $T_1 < T_2$, $T_2 \leq \alpha_{ch}(T)$,
   - (i) if $H$ satisfies $L$ at $T_1$, and there does not exist $T_2 \in \Pi$ s.t. $T_1 \leq T_2 \leq T_3 \leq \alpha_{ch}(T)$, then $H$ satisfies $L$ at $T_3$;
   - (ii) if $L \in c(T_1)$, and there does not exist $T_2 \in \Pi$ s.t. $T_1 < T_2 < T_3$ and $L \in c(T_2)$, then $H$ satisfies $L$ at $T_3$.

Hence, the world is initially in an admissible state of the static default theory (Condition (1)), and it changes in an admissible manner (Condition (2)) so that: literals not caused to change persist (Condition (3.i)), and caused change is realized (Condition (3.ii)).

3.1 Defining Admissibility w.r.t. a Static Theory

The static theory determines the admissibility requirements $\alpha_{st}, \alpha_{ch}$ after being qualified by the combined effect of observations and persistence. We model this effect through virtual observations, assumed to be part of a domain description $D$ despite not being explicitly observed. Adding a set $(Q)$ of such observation in $D$ is a virtual extension of $D$ (by $Q$). If $D_1, D_2$ are virtual extensions of $D$ by $Q_1, Q_2$, respectively, and $Q_1 \subseteq Q_2$, then $D_1$ is preferred over $D_2$.

The domain description $D_1 = D^*$ is a virtual extension of $D^*$ by $Q_1^1 = \emptyset$. The domain description $D_2$ obtained from $D^*$ by adding the observations in $Q_2^2 = \{ \"-Fly(Tweety) holds-at-T\" | T > 2 \}$, is a virtual extension of $D^*$ by $Q_2$. Clearly, $D_1$ is preferred over $D_2$.

Note that virtual observations are not meant to capture abnormal situations. Instead, a virtual observation at $T_{vert}$ is simply interpreted as the persistence to $T_{vert}$ of a known observation at $T_{obs}$, providing a means for the known observation at $T_{obs}$ to qualify the static theory at $T_{vert}$. The minimization of virtual observations guarantees that known observations persist only as needed to achieve this effect.

Definition 5 (Internally Qualified Model) An internally qualified model $M$ of a domain description $D'$ is an externally qualified model of $D'$ under $(\alpha_{st}, \alpha_{ch})$ supported by $c$, iff for every $T \in \Pi$,

1. $\langle S_1, S_2 \rangle \in \alpha_{ch}(T)$ iff $S_1, S_2$ are models of the static theory in $D'$ given as non-defeasible facts the literals observed in $D'$ at $T$;
2. $\langle S_1, S_2 \rangle \in \alpha_{ch}(T)$ iff $S_2$ is a model of the static theory in $D'$ given as non-defeasible facts (i) the literals observed in $D'$ at $T'$ for every $T' \in \{ T', T + \varepsilon \}$, for some $\varepsilon > 0$; (ii) the literals satisfied by both $S_1$ and $S_2$; and (iii) the action constants in $B(D', T)$.

A static theory’s models map the theory’s propositional symbols to truth-assignments that are compatible with the theory’s default extensions. The semantics of these models is treated as a black-box, about which we only assume that a consistent set of input facts is satisfied by all (if any) models of the static theory; this rather benign assumption holds in typical default static theory semantics (e.g., [1, 10]).

Due to the existence of causal laws that may override the static knowledge, we distinguish between two admissibility requirements: (1) $\alpha_{st}$ ensures static admissibility at the initial state of affairs, and (2) $\alpha_{ch}$ ensures admissible change thereafter, taking into account previously caused exceptions (Condition (2.ii)). The two can be reduced to one if causal laws never override the static knowledge.

Definition 6 (Model) A model $M$ of a domain description $D$ is an internally qualified model of a virtual extension $D'$ of $D$ such that there exists no virtual extension $D''$ of $D$ that has an internally qualified model, and such that $D''$ is preferred over $D'$.

The virtual extension $D_1$ of $D^*$ has an internally qualified model, where from time-point 2 onwards Fred is dead and not flying, while Tweety is flying. The virtual extension $D_2$ of $D^*$ also has an internally qualified model, where Tweety is not flying. The preference of $D_1$ over $D_2$, and over any other virtual extension of $D^*$, implies that the internally qualified models of $D_1$ are also the models of $D^*$.

Note, in particular, that since virtual extensions of a domain are expected to have internally (and hence externally) qualified models, virtual observations in these virtual extensions are forced to respect the default persistence, as per Condition (3.ii) of Definition 4.

The central role of observations in our semantics, as the knowledge that bootstraps reasoning, is consistent with Figure 1(e). Indeed, since every other type of knowledge is amenable to qualification, the following strong elaboration tolerance result can be established.

Theorem 1 (Elaboration Tolerance Theorem) Let $D$ be a consistent domain, $D'$ a domain with no observations, and $D \cup D'$ their union, where the static theories of $D$ and $D'$ are merged together to form the single static theory of $D \cup D'$. We assume that the static theory of $D \cup D'$ is consistent. Then, $D \cup D'$ is a consistent domain.

Proof (sketch): Let $D_{ext}$ be the virtual extension of $D$ by $\{ \"L holds-at-T\" | S(M, T) satises L \}$, for $M$ a model of $D$. $M$ can be shown to be an internally qualified model of $D_{ext} \cup D'$, which is a virtual extension of $D \cup D'$. This implies the claim. □

4 Concluding Remarks

We have proposed an integrated formalism for reasoning with both default static and default causal knowledge, two problems that have been extensively studied in isolation from each other. Our proposed solution applies to domains where the static knowledge is "stronger" than the causal knowledge, and where it is appropriate for the former
to qualify excessive change caused by the latter. Of course, these assumptions might not be appropriate for every domain. Our semantics already allows for “strong” causal laws to override static knowledge.

Our agenda for future research includes further investigation of such “strong” causal knowledge (constituting a different configuration in Figure 1.(c)), and of how “strong” static knowledge can generate extra (rather than block) causal change. We would also like to develop computational models corresponding to the theoretical framework presented here, using, for example, ideas from argumentation.

Although we are not aware of any previous work explicitly introducing Fred to Tweety, much work has been done on the use of default reasoning in inferring causal change. Of particular note in the context of the qualification problem are [3, 12]. An interesting approach to distinguishing between default and non-default causal rules in the context of the Language C+ is given in [2].

REFERENCES


Repairing Decision-theoretic Policies Using Goal-oriented Planning

Ch. Mies 1 and A. Ferrein 2 and G. Lakemeyer 2

Abstract. In this paper we address the problem of how decision-theoretic policies can be repaired. This work is motivated by observations made in robotic soccer where decision-theoretic policies become invalid due to small deviations during execution; and plan repair might pay off rather than replanning from scratch. Our policies are generated with READYLOG, a derivative of Golog based on the situation calculus, which combines programming and planning for agents in dynamic domains. When an invalid policy was detected, the world state is transformed into a PDDL description and a state-of-the-art PDDL planner is deployed to calculate the repair plan.

1 Introduction

Using decision-theoretic (DT) planning for the behavior specification of a mobile robot offers some flexibility over hard-coded behavior programs. The reason is that decision-theoretic planning follows a more declarative approach rather than exactly describing what the robot should do in a particular world situation. The programmer equips the robot with a specification of the domain, a specification of the actions and their effects, and the planning algorithm chooses the optimal actions according to a background optimization theory which assigns a reward to world states. With this reward, certain world states are preferred over others and goal-directed behavior emerges. The theory behind DT planning is the theory of Markov Decision Processes (e.g. [2, 3, 28]). The MDP model allows for stochastic actions, the solution of such an MDP leads to a behavior policy which optimizes the expected cumulated reward over the states which were traversed during planning. DT planning has been deployed for the decision-making in a variety of robotic applications. An interesting approach is the integration of DT planning techniques into other existing robot programming languages. One of these approaches is the language DTGolog proposed in [5], which marries DT planning with the well-known robot programming framework Golog [24, 29]. They follow the idea to combine decision-theoretic planning with explicit robot programming. The programmer has the control over the planning algorithm for example by restricting the state space for the search for a policy. These techniques have been successfully deployed in dynamic real-time domains as well. In [17, 15], the Golog dialect READYLOG is proposed which is well-suited to formulate the behavior of soccer robots in the RoboCup. Besides support for continuous change and probabilistic projections, also DT planning as proposed in [5] is integrated into their READYLOG interpreter. READYLOG was used for controlling soccer robots in the past [12, 17], but turned out useful also for the abstract behavior specification of soccer agents [13]. In dynamic domains like robotic soccer it turns out that policies become easily invalid during execution due to failing actions. The observation is that slight deviation in the execution can make the policy fail. Consider for example, when a ball should be intercepted and the robot does not have the ball in its gripper afterwards, or a move action where the robot deviated slightly from the target position. The result is, however, that the remainder policy cannot be executed anymore as preconditions for subsequent actions are violated. Nevertheless, often simple action sequences like goto(x, y); turn(θ); intercept-ball are the solution to the problem. In this paper we describe an approach, how policies can be repaired. The rest of this paper is organized as follows. In Section 2 we briefly elucidate the formalisms we use in our approach, namely the language READYLOG and the language PDDL. Section 3 addresses the execution monitoring for detecting when a policy has become inapplicable. In Section 4 we show our transformation of the world description between the situation calculus and PDDL as well as the repairing itself. Section 5 presents first results of the plan repair algorithm in simulated soccer as well as in the well-known Wumpus world. We conclude with Section 6 discussing some related work there.

2 Planning Background

2.1 DT Planning in Readylog

READYLOG [16, 17], a variant of Golog, is based on Reiter’s variant of the situation calculus [29, 25], a second-order language for reasoning about actions and their effects. Changes in the world are only due to actions so that a situation is completely described by the history of actions starting in some initial situation. Properties of the world are described by fluents, which are situation-dependent predicates and functions. For each fluent the user defines a successor state axiom specifying precisely which value the fluent takes on after performing an action. These, together with precondition axioms for each action, axioms for the initial situation, foundational and unique names axioms, form a so-called basic action theory [29].
ReadyLog integrates several extensions made to Golog like loops and conditionals [24], recursive procedures, but also less standard constructs like the nondeterministic choice of actions as well as extensions exist for dealing with continuous change [22] and concurrency [10], allowing for exogenous and sensing actions [8] and probabilistic projections into the future [21], or decision-theoretic planning [5] employing Markov Decision Processes (MDPs), into one agent programming framework [16, 17, 4]. For specifying the behaviors of an agent or robot the following constructs exist: (1) sequence \( (a; b) \), (2) nondeterministic choice between actions \( (a|b) \), (3) solve an MDP \( \text{solve}(p, h) \), \( p \) is a Golog program, \( h \) is the MDP’s solution horizon, (4) test actions \( (?c) \), (5) event-interrupt \( \text{waitFor}(c) \), (6) conditionals \( (\text{if}(c; a_1, a_2)) \), (7) loops \( \text{while}(c; a_1) \), (8) condition-bounded execution \( \text{withCtrl}(c, a_1) \), (9) concurrent execution of programs \( \text{proc}(p_1, p_2) \), (10) stochastic actions, (11) probabilistic (offline) projection \( \text{proj}(c; a_1) \), and (12) procedures \( \text{proc}([\text{name}(\text{parameters})], \text{body}) \) naming the most important ones.

ReadyLog uses a transition semantics as proposed by [10] with CONGOLG. The program is interpreted in a step-by-step fashion where a transition relation defines the transformation from one step to another. In this transition semantics a program is interpreted from one configuration \( \langle \sigma, s \rangle \), a program \( \sigma \) in a situation \( s \), to another configuration \( \langle \delta, s' \rangle \) which results after executing the first action of \( \sigma \), where \( \delta \) is the remaining program and \( s' \) the situation resulting of the execution of the first action of \( \sigma \). The one-step transition function \( \text{Trans} \) defines the successor configuration for each program construct. In addition, another predicate \( \text{Final} \) is needed to characterize final configurations, which are those where a program is allowed to legally terminate.

To illustrate the transition semantics, let us consider the definition of \( \text{Trans} \) for some of the language constructs:

1. \( \text{Trans}(\text{nil}, s, \delta, s') \equiv \text{false} \)
2. \( \text{Trans}(\alpha, s, \delta, s') \equiv \text{Poss}(\alpha, s) \land \delta = \text{nil} \land s' = \text{do}(\alpha, s) \)
3. \( \text{Trans}(\langle \delta_1; \delta_2 \rangle, s, \delta, s') \equiv \text{Final}(\delta_1, s) \land \text{Trans}(\delta_2, s, \delta, s') \lor \exists \delta'. \delta = (\delta' \land \delta_2) \land \text{Trans}(\delta_1, s, \delta', s) \)
4. \( \text{Trans}(\sigma_1 || \sigma_2, s, \delta, s') \equiv \exists \gamma. \delta = (\gamma || \sigma_2) \land \text{Trans}(\sigma_1, s, \gamma, s') \lor \exists \gamma. \delta(\sigma_1 || \gamma) \land \text{Trans}(\sigma_2, s, \gamma, s') \)

1. Here \( \text{nil} \) is the empty program, which does not admit any further transitions.
2. For a primitive action \( \alpha \) we first test if its precondition holds. The successor configuration is \( \langle \text{nil}, \text{do}(\alpha, s) \rangle \), that is, executing \( \alpha \) leads to a new situation \( \text{do}(\alpha, s) \) with the \( \text{nil} \) program remaining.
3. The next definition concerns an action sequence \( [\delta_1; \delta_2] \), where it is checked whether the first program is already final and a transition exists for the second program \( \delta_2 \), otherwise a transition of \( \delta_1 \) is taken.
4. \( \sigma_1 || \sigma_2 \) denotes that \( \sigma_1 \) and \( \sigma_2 \) can be executed concurrently. Here the definition of \( \text{Trans} \) makes sure that one of the two programs is allowed to make a transition without specifying which. This corresponds to the usual interleaved semantics of concurrency.

We only sketched the transition semantics here. For a concise overview of the transition semantics we refer the interested reader for example to [10, 9]. Note that the transition semantics allows for a natural integration of sensing and online execution of programs.

In this paper we focus on repairing DT policies. Therefore, we discuss in little more detail, how decision-theoretic planning is conducted in the ReadyLog framework. From an input program, which leaves several choices open, the forward-search value iteration algorithm, which was originally proposed in [5], computes an optimal policy. We call the input programs plan skeletons. Fig. 1 illustrates the planning. For each stochastic action outcome, the algorithm inserts a new branch and annotates the tree with the probability of occurrence of this outcome. When the whole tree of possible actions is spanning, the value at each branch point is calculated as shown in the figure. The different outcomes of stochastic actions are called nature’s choice points and are all represented in the final policy, which is a tree branching over nature’s choices. Agent choice points in the plan skeleton are optimized away when calculating the policy. At each agent choice point the best choice is taken according to the value of the respective branch.

An optimal policy \( \pi \) is then executed with ReadyLog’s runtime system. We refer to [16] for further details. Important for this work is to know that the planning process is done off-line, based on models of the world, while the execution of the policy naturally is on-line. Discrepancies between the model and the real execution might occur. To detect these differences, in [16] it was proposed to introduce special markers into the policy. With these markers one can easily detect, when a policy becomes invalid and can no longer be executed. We want to stress that these discrepancies between the MDP model in ReadyLog and the real execution trace occurs because our models are in general not precise enough. We cannot foresee and model all possible outcomes that can happen. Against this backdrop, we are using fully observable MDPs as an idealized model.

We will show in Section 3 how the concept of markers works formally and how it can be detected when a policy has become invalid during execution.

2.2 PDDL and SGPlan

The Planning Domain Definition Language (PDDL) is a family of formal standard languages to define planning domains. It was first proposed by McDermott in 1998 [26]. It is used as the description language for the bi-annual International Plan-
ning Competitions (IPC) and was since then further developed to meet the requirements of the planning competitions. In this work, we basically use PDDL2.2. The most important language features of this PDDL-version are the following: (1) basic actions; (2) objects with types; (3) conditional action effects; (4) fluents; (5) metrics.

The basic actions have a STRIPS style, and are used to define the possible actions. An action definition consists of a precondition formula stating at which world states the action is applicable, the effect formula defining the action’s effect to the world when applied, and the action parameters, which is a list of all used variables. For pruning the search tree, it is important that objects have certain types. Only objects of the correct type may be unified with the variables or predicates. There exists a default super-type object in PDDL. Conditional action effects are a huge help in defining actions. The construct \(\text{when} : \varphi_{\text{pre}} \varphi_{\text{effect}}\) defines such an effect. If \(\varphi_{\text{pre}}\) holds when the action is applied, the effect \(\varphi_{\text{effect}}\) is added. Nothing is changed otherwise. In PDDL planning systems, such an action \(a\) is usually transformed into two actions \(a_1\) and \(a_2\), where the precondition of \(a_1\) is the one of \(a\) combined with \(\varphi_{\text{pre}}\) or its negation, respectively. The effect is combined with \(\varphi_{\text{effect}}\) if \(\varphi_{\text{pre}}\) was added to the precondition, e.g. the if-condition was fulfilled. Unlike READYLOG, fluents only may take numerical values. Due to the object-oriented approach of PDDL, which especially maintains a finite state space, it is not possible to have numerical values as arguments of predicates. Fluents are defined with the keyword function. There exist five ways to change a fluent’s value by applying one of the following actions: increase, decrease, scale-up and scale-down for relative assignments and assign for absolute assignment. Metrics are used to measure the quality of a plan. A metric is defined in the problem definition and is a function over fluents. It can either be maximized or minimized. A PDDL world state is a collection of predicates that are true at certain time points. Using the closed world assumption all predicates not listed in a world state are assumed to be false. The first important time point is the initial state. When PDDL-subsets without time representation (as PDDL2.2) are considered, other important time points are set after actions are applied. More details about PDDL can be found in [18, 14, 19].

For our system, we used the metric PDDL2.2 planner SGPLAN, which won the last two IPC’s in the metric domain, which is the one of our interest. The basic architecture of SGPLAN follows a hierarchical planning approach and decomposes the overall goal into several sub-goals. These are ordered such that no sub-goal interferes with another one. Then the sub-goals are planned on. In many cases, not all actions are needed and so SGPLAN can prune the search tree by not considering unused actions resulting in a substantially smaller search space. The sub-plans, fulfilling the sub-goals, are then used to build a plan solving the entire planning problem. A more detailed description of SGPLAN is given in [6] and [23].

3 Policy Execution Monitoring

As we pointed out, READYLOG distinguishes between an off-line mode, in which the interpreter is switched during plan generation, and an on-line mode for executing the calculated policies and interacting with the environment. In our previous work [16] it was shown how discrepancies between planning and execution could be detected by introducing markers into the policy which keep track of the model assumptions made during plan generation. The policy was discarded in cases where the model assumption did not hold at execution time and re-planning was initiated. Now, we extend this idea and show how the plan repair procedure is invoked during plan execution.

3.1 Marking Possible Failures during Planning

As we have stated in Section 2, Readylog uses a special operator \(\text{solve}(h, f, p)\) for a program \(p\), a reward function \(f\), and a finite horizon \(h\), which initiates decision-theoretic planning in the on-line transition semantics (cf. also [15]).

\[
\text{Trans}(\text{solve}(h, f, p), s, \delta, s') \equiv \exists \pi, v, pr. \text{BestDo}(p, s, h, \pi, v, pr, f) \land s' = s \land \delta = \text{applyPol}(\pi).
\]

The predicate \(\text{BestDo}\) first calculates the policy for the whole program \(p\). The policy \(\pi\) is then scheduled for on-line execution as the remaining program. For generating a policy, we use models of how actions change the world. In the case of the robot’s position, for example, policy generation works with an abstract model of the robot’s movements so that robot positions in future states can simply be computed without having to rely on actual sensing. Making use of such models during plan generation requires that we monitor whether \(\pi\) remains valid during execution as discrepancies between the model and the real-world situation might arise. Monitoring is handled within special \(\text{applyPol}\) transitions, whose definition we omit here. Note that the \(\text{solve}\) statement never reaches a final configuration as further transitions are needed to execute the calculated policy. To keep track of the model assumptions we made during planning, we introduce special markers into the policy. Hence, in the definition of \(\text{BestDo}\) we have to store the truth values of logical formulas. For conditionals this means:

\[
\text{BestDo}(\text{if } \varphi \text{ then } p_1 \text{ else } p_2 \text{ endif}; p, s, h, \pi, v, pr) \overset{\text{def}}{=} \\
\varphi[s] \land \exists \pi_1. \text{BestDo}(p_1; p, s, h, \pi_1, v, pr) \land \\
\pi = M(\varphi, \text{true}); \pi_1 \lor \\
\neg \varphi[s] \land \exists \pi_2. \text{BestDo}(p_2; p, s, h, \pi_2, v, pr) \land \\
\pi = M(\varphi, \text{false}); \pi_2
\]

Thus, for conditionals we introduce a marker into the policy that keeps track of the truth value of the condition. We prefix the generated policy with a marker \(M(\varphi, \text{true})\) in case \(\varphi\) turned out to be true in \(s\) and \(M(\varphi, \text{false})\) if it is false. While-loops are treated in a similar way. The treatment of a test action \(\varphi?\) is even simpler, since only the case where \(\varphi\) is true matters. If \(\varphi\) is false, the current branch of the policy is terminated, which is indicated by the \(\text{nil}\) policy, i.e. the empty policy.

\[
\text{BestDo}(\varphi?; p, s, h, \pi, v, pr) \overset{\text{def}}{=} \\
\varphi[s] \land \exists \pi'. \text{BestDo}(p, s, h, \pi', v, pr) \land \\
\pi = M(\varphi, \text{true}); \pi' \lor \\
\neg \varphi[s] \land \pi = \text{nil} \land pr = 0 \land v = \text{reward}(s)
\]
With keeping track of the model assumption in conditions this way, it is easy to detect when a policy became invalid due to discrepancies between plan and execution time.

Another source of a failure is when an unexpected action outcome occurs. Previously, these cases were not treated explicitly. In the case where none of the action outcomes occurred, the resulting policy simply was the empty policy. As the empty policy directly terminates, policy execution directly terminated. It was possible to exploit this behavior, as we were not interested in repairing a policy. We simply discarded it. However now, when we try to repair a policy, we need to account for this. Thus in the following, we introduce the marker $\mathcal{M}(\text{unexp})$ into the policy for the case that an action had an outcome which was not foreseen and thus was not part of the optimal policy.

In the case of a stochastic action $a$, the predicate $BestDoAux$ with the set of all outcomes for this stochastic action is expanded. $choice'(a) \overset{df}{=} \{n_1, \ldots, n_k\}$ is an abbreviation for the outcomes of the stochastic action $a$.

$$BestDo(a; p, s, h, \pi, v, pr) \overset{df}{=} \exists \pi', v'.BestDoAux(choice'(a), a, p, s, h, \pi', v', pr) \land \pi' = a; senseEffect(a); \pi' \land v = reward(s) + v'$$

The resulting policy is $a; senseEffect(a); \pi'$. The pseudo action $senseEffect$ is introduced to fulfill the requirement of full observability. For similar reasons, predicates $senseCond(\varphi, n)$ need to be defined for each action outcome to find out which outcome condition occurred when executing the stochastic action (see e.g. [15] for a thorough discussion). The remainder policy $\pi'$ branches over the possible outcomes and the agent must be enabled to sense the state it is in after having executed this action. The remainder policy is evaluated using the predicate $BestDoAux$. The predicate $BestDoAux$ for the (base) case that there is one outcome is defined as

$$BestDoAux(\{n_k\}, a, \delta, s, h, \pi, v, pr) \overset{df}{=} \neg Poss(n_k, s) \land \pi = \text{nil} \land v = 0 \land pr = 0 \lor Poss(n_k, s) \land senseCond(n_k, \varphi_k) \land (\exists \pi', v', pr').BestDo(\delta, do(n_k, s), h, \pi', v', pr') \land \pi = \text{if } \varphi_k \text{ then } value(v, n_k); \pi' \land \text{else } \mathcal{M}(\text{unexp}) \text{ endif} \land v = v' \cdot \text{prob}(n_k, a, s) \land pr = pr' \cdot \text{prob}(n_k, a, s)$$

In case none of the stochastic action outcome of $choice'(a)$ applied, we insert the marker $\mathcal{M}(\text{unexp})$ into the policy. As the repair strategy for unexpected outcomes we decided to choose the most promising outcome in terms of the value of the remainder policy. To know which branch of the conditional has the highest value, we additionally store the value of each branch in the policy and the outcome action leading to that branch. For the other cases, $BestDoAux$ is defined as

$$BestDoAux(\{n_1, \ldots, n_k\}, a, p, s, h, \pi, v, pr) \overset{df}{=} \neg Poss(n_1, s) \land BestDoAux(\{n_2, \ldots, n_k\}, p, s, h, \pi, v, pr) \lor Poss(n_1, s) \land (\exists \pi', v', pr').BestDoAux(\{n_2, \ldots, n_k\}, p, s, h, \pi', v', pr') \land \exists \pi_1, v_1, pr_1.BestDo(p, do(n_1, s), h - 1, \pi_1, v_1, pr_1) \land senseCond(n_1, \varphi_1) \land \pi = \text{if } \varphi_1 \text{ then } value(v_1, n_1); \pi_1 \text{ else } \pi' \text{ endif} \land v = v_1 + v_1 \cdot \text{prob}(n_1, a, s) \land pr = pr_1 + p_1 \cdot \text{prob}(n_1, a, s)$$

Note that according to the MDP we employ when calculating our policy, an unexpected outcome may not occur, as in a fully observable MDP, which we assume here, this cannot happen. On the other hand, in practice, this situation can happen. With introducing the unexpected outcome marker, we are able to react to such design flaws. We assume in these cases that the action outcome with the highest value is the most promising one, as it describes a good compromise between the probability that nature will choose this action and the value of the remainder policy. Our repair strategy now tries to create the situation where the remainder policy for the outcome with the highest value becomes executable again.

### 3.2 Monitoring Execution Flaws

In the following we show, how a policy is executed, and how plan repair is initiated when marker conditions fail. First, we regard the conditional marker.

$$Trans(applyPol(\mathcal{M}(\varphi, v), \pi, s, \delta, s')) \equiv s = s' \land (\varphi[s] \land v = \text{true} \lor \neg \varphi[s] \land v = \text{false}) \land \delta = applyPol(\pi) \lor (\varphi[s] \land v = \text{false} \lor \neg \varphi[s] \land v = \text{true}) \land (\exists \pi'.\text{Repair}_1(s, \varphi, v, \pi') \land \delta = applyPol(\pi'; \pi) \lor \forall \pi, \neg \text{Repair}_1(s, \varphi, v, \pi') \land \delta = \text{nil})$$

If this marker does not fail, we go on with the execution of the policy, as shown in the first disjunction in the $Trans$ predicate above. Otherwise, we initiate to calculate a repair plan with the predicate $\text{Repair}_1$. The arguments of $\text{Repair}_1$ is the current situation $s$, the condition $\varphi$, and $\varphi'$'s true value $v$ from planning time. The result is the repair policy $\pi'$ which needs to be executed before the original policy $\pi$. If no such repair policy exists, the execution of the policy is terminated.

If our policy contains a conditional which was inserted when a stochastic action was interpreted, we have to check which outcome occurred. This, as such would not be further mentionable as this case is usually the same as for ordinary conditions in the transition semantics. Though, as we need to deal with the marker for unexpected outcomes $\mathcal{M}(\text{unexp})$, we need some special treatment of conditionals which occur inside a policy. While checking which outcome condition occurred in reality, we keep track of the branch with the highest value $n_i$, as well as its outcome action $n_i$ and the corresponding remainder policy $\pi_i$. To do so, we need a new predicate
TransAux which extends Trans with additional parameters.

\[
\text{Trans}(\text{applyPol}(\text{if } \varphi \text{ then value}(v, n); \pi_1 \text{ else } \pi_2 \text{ endif}), s, \delta, s') \equiv \\
\exists \pi, m. \text{TransAux}(\text{applyPol}(\text{if } \varphi \text{ then value}(v, n); \pi_1 \text{ else } \pi_2 \text{ endif}), s, \delta, s', 0, m, \pi).
\]

\text{TransAux} now "checks" if the condition \(\varphi\) holds, otherwise, the next outcome is investigated.

\[
\text{TransAux}(\text{applyPol}(\text{if } \varphi \text{ then value}(v, n); \pi_1 \text{ else } \pi_2 \text{ endif}), s, \delta, s', v, n, \pi_c, \pi_e) \equiv \\
\varphi[s] \land \delta = \text{applyPol}(\pi_1) \land s = s' \lor \\
\neg \varphi[s] \land \exists v', n', \pi'.'
\]

\[
\text{TransAux}(\text{applyPol}(\pi_2), s, \delta, s', v', n', \pi') \land \\
(v > v_c \land v' = v \land n' = n \land \pi' = \pi_1 \lor \\
v \leq v_c \land v' = v \land n' = n_c \land \pi' = \pi_e)
\]

We further keep track, which outcome had the highest value.

If then all outcomes failed, the remaining branch in the policy is the unexpected outcome marker. As \(v_c, n_c,\) and \(\pi_c\) contain the value, the outcome action and the remainder policy of the branch with the highest value, we can use this information for reconstructing the desired world state for our plan repair. If then all outcomes failed, the remaining branch in the policy is the unexpected outcome marker. Note that the \(v_c, n_c,\) and \(\pi_c\) contain the value, the outcome action and the remainder policy of the policy with the highest value.

\[
\text{TransAux}(\text{applyPol}(\mathcal{M}(\text{unexp})), s, \delta, s', v_c, n_c, \pi_c) \equiv \\
v_c = 0 \land \delta = \text{nil} \lor \\
v_c \neq 0 \land (\exists \pi'. \text{Repair}_2(s, n_c, \pi') \land \delta = \text{applyPol}(\pi'; \pi_c) \lor \\
\forall \pi'. \neg \text{Repair}_2(s, n_c, \pi') \land \delta = \text{nil})
\]

Similar as for condition markers, the predicate \(\text{Repair}_2\) initiates a plan repair. The arguments for \(\text{Repair}_2\) is the current situation, and the outcome action leading to the best policy branch. The state \(s' = do(n_c, s)\) is the desired world state for the planner.

With the above predicates we have a means to decide when it is possible to repair a policy. Sometimes, it seems to be wise not to repair a policy but to re-plan it from scratch. Consider, for example, a robot accomplishing a policy for attacking the opponent goal. His policy will include actions like \text{intercept-ball}, \text{dribble}, and \text{shoot}. During the journey towards the opponent goal it might be that the robot loses the ball. With a simple intercept action control over the ball could be regained and the remainder of the policy becomes executable again. On the other hand, if the robot cannot shoot because of a hardware failure and due to that the policy to score may become invalid as some preconditions of the shoot action are violated, a plan repair seems senseless. As, in general, it is hard to decide automatically in which cases it might be useful to try and repair or to cancel the execution of the policy, we lay it into the hand of the system designer to decide under which condition a policy shall be re-planned from scratch. To this end, we introduce the construct \(\text{guardEx}(\psi, p)\). As long as the condition \(\psi\) holds, we continue with executing and possibly repairing our policy. If \(\psi\) does not hold, we cancel the execution of the current policy. The program \(p\) may either contain a \text{solve} statement, or an already calculated policy \(\pi\).

\[
\text{Trans}(\text{guardEx}(\psi, \text{solve}(h, f, p)), s, \delta, s') \equiv \\
\exists \delta''. \text{Trans}(\text{solve}(h, f, p), s, \delta'', s') \land \delta = \text{guardEx}(\phi, \delta'')
\]

Note, that \(\delta'' = \text{applyPol}(\pi)\) is the result of calculating the policy \(\pi\) with the solve statement. For the case that the argument of \(\text{guardEx}\) is already a policy,

\[
\text{Trans}(\text{guardEx}(\psi, \text{applyPol}(\pi)), s, \delta, s') \equiv \\
\psi[s] \land \exists \delta'. \text{Trans}(\delta, s, \delta, s') \land \delta' = \text{guardEx}(\psi, \delta') \lor \\
\neg \psi[s] \land s = s' \land \delta = \text{nil}
\]

Thus, as long as the condition holds, the policy is executed. If \(\psi\) fails, we immediately terminate the execution of the policy.

4 Repairing Policies

In the previous section we showed how we could determine whether or not a policy is corrupt, i.e. it cannot be executed until its end for some reasons, and shall be repaired. Now assume a policy \(\pi\) that has become invalid due to a violation of the condition \(\varphi\) in \(\mathcal{M}(\varphi, \text{false})\). To repair the policy we have to conduct the following steps:

1. **Calculate the desired world state**: The desired world state is the world state in which the remainder policy becomes executable again. This state serves as the goal description in the next step.

2. **Translate state and goal description to PDDL**: Next, we translate the desired world state into the goal description of PDDL. Furthermore, in order to be able to perform goal-directed planning, we need an initial state description; we therefore translate the current state to a PDDL description and use it as the initial state.

3. **Plan step**: The initial and goal state description are transmitted to an external planner via file system, and the planning process is initiated. In our current implementation we make use of SGP3. Note that, as we are using the abstract PDDL description, we can plug any other PDDL planner to our system easily.

4. **Re-translate the repair plan**: When a plan has been generated, we have to re-translate it to a READYLOG policy which can be executed then.

**Desired Reference World State** We need to calculate a world state that serves as goal description for the PDDL planner. This state is called the desired reference world state denoted by \(s^*\). It has to be a unique state since PDDL is not able to deal with multiple goal states. We have to distinguish if the repair is initiated by \(\text{Repair}_1(s, \varphi, v, \pi)\) or \(\text{Repair}_2(s, n_c, \pi)\). In the former case, we need to find a repair plan leading from situation \(s\) to a situation \(s'\) in which \(\varphi \equiv n\) holds. We currently restrict \(\varphi\) to simple conditions, like testing fluent values. For these simple cases it is more likely to find a simple repair plan. In the latter case, the last executed (stochastic) action resulted in an unexpected outcome. Then \(s\) is not complete since the remainder of the failed policy might need an effect
of the stochastic actions. Since PDDL needs exactly one goal state, we have to determine one of the possible effects. Since the highest-valued outcome during planning gives a compromise between occurrence probability and value of that outcome, we decided to choose that one as a heuristic. Therefore, we simply extend \( s \) by that outcome: \( s^* = do(n_c, s) \).

**Translating Golog Situations to PDDL States** In the next step of our repair algorithm, we have to construct a PDDL description of our planning problem. As initial PDDL state, we have to transform the current world state. The PDDL goal state is the translation of the desired reference world state defined in the last section. The resulting PDDL plan can then be translated back to READYLOG. If it is executed before the remainder of the current policy, it is executable again. The state description has to consist of as less atoms as possible as the run-time of PDDL-planners is, in general, directly related to the number of ground actions. Therefore, we have to restrict our translation to the important and salient parts of our situation calculus world description.

- **Restrict to salient fluents.** The goal state needs only refer to fluents that occur in the policy. Thus for the PDDL state translation, we can ignore all fluents that are not mentioned by the READYLOG policy.
- **Restrict fluent domains:** The idea is to translate only that values that are important to the remainder of the policy. We introduced this kind of pruning for more complex domains like simulated robot soccer. For example, a policy for a pass between player \( p_1 \) and player \( p_2 \) does not necessarily need to know anything about the positions of the other players.

Therefore, we only translate \( p_1 \)'s and \( p_2 \)'s positions.

**Translating Back Plans to Policies** After having calculated a PDDL repair plan with the external planner, it has to be executed before the remainder of the failed policy. Since PDDL is a deterministic language, no nondeterministic actions can occur in the plan. Thus, each PDDL action is translated to a sequence of READYLOG actions with specified outcome. For example, the PDDL action `grab_gold` from the Wumpus domain is translated to the READYLOG action description `grab_gold; select_outcome(2)` denoting that the gold has been grabbed successfully. `select_outcome(n)` is an action which simply identifies which outcome of a stochastic action's outcome needs to be chosen. For a deterministic action, we do not have to select a specific outcome. This sequence is then translated to a transition policy that is executed before the remainder of the failed policy. The translation process is again straight-forward. The art is, though, to find a set of PDDL actions that can be mapped to READYLOG. Note that the resulting policy does not allow any branching over nature's choices, i.e. all stochastic actions or events must result in the outcome that is specified in the PDDL plan sequence by the `select_outcome(n)` statement. Assuming a PDDL action is mapped to the sequence `a; select_outcome(i)`, the resulting transition policy \( \pi_{trans} \) has to represent this sequence exactly.

### 5 First Empirical Results

For proving the concept of the presented plan-repair scheme, we applied it to the toy domain WumpusWorld, where an agent has to hunt a creature, the Wumpus, in a maze environment with pits and traps while searching for a pile of gold. The domain is modeled in a stochastic way, that is, the basic actions like move or grab gold have stochastic effects, they can succeed or fail with certain probabilities. Moreover, exogenous events can occur. For example, the agent may lose the gold with or move to a not-modeled direction with 30 percent. In these cases, plan repair is invoked. The plans the agent performs are decision-theoretic policies with a plan horizon of three, meaning that the agents plans ahead the next three actions. In this paper we propose to connect READYLOG to an external PDDL planner. The first important question is whether or not plan repair pays off in the given application domain, as it was shown that, in general, plan repair is at least as complex as planning from scratch [27]. Therefore, we compared the run-times of planning from scratch (DT) each time a policy became invalid with an iterative deepening depth-first brute-force planner (BF) in READYLOG itself (this is very similar to the comparison made in [7]). Next, we compared this to the PDDL repair scheme for the first row in this paper to get information about the extra computational overhead to transfer the world states between READYLOG and the PDDL planner.

The results are shown in Tab. 1. We defined three evaluation setups STANDARD, SMALL and HORIZON. The scenarios where we applied the PDDL repair mechanism are suffixed with `pddl`, the ones with brute-force planning are suffixed with `BF` and the ones without repairing with `NR` (no repair). The first column represents the number of successful runs (out of 100 runs in total) and their average run-time. A run is successful when the agent arrives at the target cell with the gold in its hands. The second column contains the number of generated policies together with their average computation time; the number of calls of the repairing mechanism as well as their average computation times are presented in the last column. All run-times are measured in real elapsed seconds. Note that there is no repairing data available if the plan repair was not performed (the `NR` cases). Therefore, the number of generated plans for these scenarios roughly equals the number of repairs plus plans of the other cases. For the first row of the table, this means that each of the 91 successful runs took on average 19.53 seconds. In total, 740 policies have been generated, each of which took 1.23 second on average to be computed. 722 times the repair routine was called due to failing plans. On average, it took 0.99 seconds to establish the successful repair plan. Note that the number of repairs can exceed the number of plans. This is the case when a repaired plan fails again. The scenario STANDARD is the starting point for our evaluation. The DT planning depth is 3 in that setup. Thus, the lengths of the DT plans and the repair plans are roughly equal. The average computation time for PDDL repair (column 3) is slightly lower than the DT planning time (column 1). The average run-time of STANDARD\_PDDL is higher than the one of STANDARD\_DT. The advantage is taken by the overhead of the repair mechanism which is mostly determined by the policy execution simulation. The setup SMALL equals STANDARD except for the smaller size of the WumpusWorld. It contains 54 cells, 8 walls and 2 holes whereas STANDARD contains 180 cells, 16 walls and 2 holes. This leads to a significantly faster PDDL repairing w.r.t. STANDARD\_PDDL. The gap between repairing and DT re-planning is large enough to be

<table>
<thead>
<tr>
<th>Scenario</th>
<th>SUCCESS</th>
<th>POLICIES</th>
<th>DT</th>
<th>BF</th>
<th>NR</th>
</tr>
</thead>
<tbody>
<tr>
<td>STANDARD</td>
<td>91</td>
<td>740</td>
<td>19.53</td>
<td>1.23</td>
<td>-</td>
</tr>
<tr>
<td>SMALL</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HORIZON</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
faster than the pure DT planning approach SMALL_NR, which does not change in comparison to STANDARD_NR. The times of SMALL_BF are also very similar to the ones of STANDARD_BF. With the last setup, HORIZON, we wanted to check the influence of an increased plan horizon to the overall run-times. It has an increased horizon of 4 instead of 3 as in STANDARD and SMALL. This leads to a significantly higher DT planning time since our method has an exponential run-time w.r.t. the horizon. Therefore, HORIZON_PDDL is much faster than its pure DT planning companion. Again, the times of the brute-force approach are not influenced by this scenario variation. The times for plan repair for PDDL and BF equal those with horizon 3. As one can see from these results, the brute-force method is superior to PDDL in all WumpusWorld scenarios. The reason lies in the simple structure of the scenario. The computation time of the brute-force method depends on the number of ground actions. There exists only 9 ground actions (4 move actions, 4 shoot actions and grab_gold). Another issue is the very preliminary interface between READYLOG and the PDDL planner. We make use of the file system to communicate the world states between PDDL and READYLOG, the brute-force planner is integrated into READYLOG.

There is a connection between the simplicity of the action description and the run-time of the brute-force method. To check this, we applied the method to a more complex and realistic domain, the simulated soccer domain. Two teams of 11 agents play soccer against each other. The available actions are move, pass, dribble, score and intercept. Each action takes arguments. Thus, the number of ground actions (where the argument variables are substituted by values) is exponential in the size of the domain. This complexity enables the PDDL repairing to dominate the brute-force repairing due to SG-Plan’s superior search heuristics. Our scenario is as follows: an agent has to intercept the ball and to dribble to the opponent’s goal. Thereby, the dribble path is planned, i.e. the field is divided into rectangular cells and the agent has to dribble from one cell to an adjacent one. We then artificially cause the policy execution to fail after some while in order to force policy repair. For a DT planning horizon of 2, i.e. the policy consists of one intercept and one dribble action, DT planning takes 0.027 and PDDL repairing 0.138 seconds on average. If the horizon is 3, the DT planning time increases to 0.302 seconds whereas the PDDL repairing time does not change. Thus, repairing with PDDL is faster than re-planning in this scenario. This confirms the evaluation of HORIZON. But in contrast to WumpusWorld, the BF repairing takes 0.305 seconds on average and is slower than the PDDL method. This gives more evidence to our observation that the performance of BF is related to the complexity of the action description. Our results show that plan repair in principle is useful in dynamic domains. We take these preliminary results as the starting point for future investigations.

6 Conclusions

In this paper we sketched our plan repair framework, where invalid DT policies generated by READYLOG are repaired using an external PDDL planner. To this end, it must be detected when a policy becomes invalid, and how the world state must look like in order to continue the remainder policy. Then, we map our READYLOG world state to a PDDL description and calculate a repair plan with an external PDDL planner. In our current implementation we use SGPLAN.

Although it was shown by Nebel and Koehler [27] that plan repair is at least as hard as planning from scratch, it turns out that plan repair works in practice under certain circumstances. In our case, the assumption is that only slight deviations in the execution make our policy invalid. Thus, only simple repair plans are needed to reach a state where the remainder policy can be executed again. A similar approach was proposed in [20], which integrate monitoring into a Golog-like language with a transition semantics similar to that of READYLOG. Their system also performs a monitoring step after each action execution and performs repair actions before the remainder of the failed policy in order to restore its executability.

In [4], contingency plans are specified to provide action patterns when previously created plans fail. A similar idea, to combine GOLOG with an external planning system based on a PDDL description, was proposed in [7]. For the future work, we plan to combine their results with ours.

Clearly, our experimental results at the current stage resemble only a first proof of concept. Further and extended testing needs to be conducted. This particularly means to compare our method not only with brute-force GOLOG planners but also with other state-of-the-art planners in several

Table 1. Results of WumpusWorld.

<table>
<thead>
<tr>
<th>Setup</th>
<th>Run time</th>
<th>Plan. time</th>
<th>Repair time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># runs</td>
<td>av. [s]</td>
<td># plans</td>
</tr>
<tr>
<td>STANDARD_PDDL</td>
<td>91</td>
<td>19.53</td>
<td>740</td>
</tr>
<tr>
<td>STANDARD_BF</td>
<td>87</td>
<td>13.95</td>
<td>713</td>
</tr>
<tr>
<td>STANDARD_NR</td>
<td>93</td>
<td>17.26</td>
<td>1213</td>
</tr>
<tr>
<td>SMALL_PDDL</td>
<td>87</td>
<td>14.62</td>
<td>713</td>
</tr>
<tr>
<td>SMALL_BF</td>
<td>89</td>
<td>13.57</td>
<td>728</td>
</tr>
<tr>
<td>SMALL_NR</td>
<td>95</td>
<td>16.58</td>
<td>1251</td>
</tr>
<tr>
<td>HORIZON_PDDL</td>
<td>90</td>
<td>58.71</td>
<td>612</td>
</tr>
<tr>
<td>HORIZON_BF</td>
<td>89</td>
<td>53.13</td>
<td>606</td>
</tr>
<tr>
<td>HORIZON_NR</td>
<td>91</td>
<td>92.85</td>
<td>921</td>
</tr>
</tbody>
</table>
other domain, similar as is done in [11]. Finally, we have to compare our approach with other state-of-the-art POMDP planners. We assume fully observable MDPs as an idealistic model for generating our policies. As there might be discrepancies between the action theory leading to a policy and the real execution, the problem that policies can fail, occurs. As such is our approach driven from the practical experience of programming robots and agents with the logic-based language Readylog and the observation that often only small errors in the execution lead to a complete failure of the optimal policy. Here, we have to stress again that Readylog only calculates partial policies, not taking the whole state space into account. For our future work, we need to take other models like POMDPs into account as well.

Acknowledgments

This work was supported by the German National Science Foundation (DFG) in the Priority Program 1125, Cooperating Teams of Mobile Robots in Dynamic Environments. We would like to thank the anonymous reviewers for their helpful comments.

REFERENCES

Designing Goal-Oriented Reactive Behaviours

Peter Novák and Michael Köster

Abstract. State-of-the-art rule-based agent programming languages, similar to AgentSpeak(L), provide theoretically solid tools for implementing cognitive agents. However, not too much attention was devoted to low-level design issues for development of non-trivial agents in them.

In this paper we discuss some design considerations we faced while implementing Jazzbot, a softbot embodied in a simulated 3D environment, implemented in a rule-based framework of Behavioural State Machines. Finally, we also make an attempt to lift our experiences to a set of informal design guidelines useful for design and implementation of agents with heterogeneous knowledge bases in rule-based agent-oriented programming languages.

1 INTRODUCTION

While the landscape of programming languages for cognitive agents is thriving (for state-of-the-art surveys see e.g. [5], or [6]), a little has been published on their application beyond small-scale example agents. In our research project we are interested in implementing embodied non-trivial agents exploiting power of heterogeneous knowledge representation (KR) techniques. To this end we recently proposed the framework of Behavioural State Machines (BSM) [16] with Jazzýk [17], an associated programming language. The BSM framework takes a rather liberal software engineering stance to programming cognitive agents. It provides a concise and flexible theoretical computational model allowing integration of heterogeneous knowledge bases (KB) into an agent system, while not prescribing a fixed scheme of interactions between them.

Most of the state-of-the-art logic based programming languages for cognitive agents, such as [7, 9, 10], tackle the problem of programming cognitive agents by introducing first class concepts with an underlying semantics of a chosen set of relations between mental attitudes of an agent. Unlike these, our approach is radically different, more based on a liberal software engineering stance. Instead of choosing a set of supported relationships among agent’s knowledge bases storing its mental attitudes, we allow an agent to have an arbitrary number of KBs and provide a modular and flexible generic programming language facilitating interactions between them. It is the task of the programmer to maintain a discipline in applying various types of interactions between KBs specific to cognitive agents, such as goal adoption/dropping, triggering reactive behaviours etc., in agent programs. To support this, we try to propose a set of higher level syntactical constructs, agent programming design patterns, which can differ between various application domains. These should provide at least a partial, semi-formal semantics so that a programmer can rely on their specification.

The liberal nature of the BSM framework allows us to experiment with integration of various KR technologies in a single agent system. In [17], we introduce the Jazzbot project: our specific aim is to develop a BDI inspired softbot roaming in a simulated 3D environment of a first-person shooter computer game. Jazzbot uses non-monotonic reasoning, in particular Answer Set Programming [3], as the core KR technology for representing its beliefs about its environment, itself and peer bots.

In this paper, we discuss our considerations and experiences on the way towards proposing a consistent set of agent programming design patterns for development of BDI inspired cognitive agents. Our approach is application driven, i.e. we try to propose such constructs on the ground of real implementation experience with non-trivial applications. As a side effect, during the development of Jazzbot demo application, we have got an opportunity to rethink design of BDI inspired agents in rule-based programming languages. On the background of the Jazzbot project we present here a collection of design considerations (Section 3) we faced, together with an attempt to lift our solutions and implementation techniques to a set of more general methodological design guidelines (Section 4). However, before coming to the main discussion of the paper, we first briefly introduce the framework of Behavioural State Machines (2). A brief summary in Section 5 concludes the paper.

2 BEHAVIOURAL STATE MACHINES

In [16] we recently introduced the programming framework of Behavioural State Machines (BSM), with an associated implemented programming language Jazzýk [17]. The BSM framework defines a new and unique agent-oriented programming language due to the clear distinction it makes between the knowledge representation and behavioural layers within an agent. It thus provides a programming framework that clearly separates the programming concerns of how to represent an agent’s knowledge about, for example, its environment and how to encode its behaviours. In the following we briefly introduce the framework of Behavioural State Machines. A more rigorous description can be found in the original publications [16, 17].

Mental states of BSM agents are collections of one or more so-called knowledge representation modules, typically denoted by \( M \), each of which represents a part of the agent’s knowledge base. Transitions between such states result from applying mental state transformers (mst), typically denoted by \( \tau \). The various types of mst determine the behaviour that an agent can generate. A BSM agent \( A \) consists of a set of KR modules \( M_1, \ldots, M_n \) and a mental state transformer \( \tau \), i.e., \( A = (M_1, \ldots, M_n, \tau) \). The mst \( \tau \) is also called an agent program.

A KR module of a BSM agent can be seen as a database of statements drawn from a specific KR language. KR modules may be used to represent and maintain various attitudes of an agent such as its
knowledge about its environment, or its goals, intentions, obligations, etc. Unlike other agent oriented languages, the BSM framework abstracts from a particular purpose a KR module can be made to serve. Agents can have any number of such KR modules and an agent designer can ascribe any appropriate purpose to these modules (such as a belief, or a goal base). Formally, a KR module \( M = (S, L, Q, U) \) is characterized by a set of states \( S \) the module can be in, a KR language \( L \) and two sets of query and update operators denoted \( Q \) and \( U \) respectively. A query operator \( \models \in \mathcal{Q} \) is a function evaluating truth value of a query formula from the KR language against the current state of the KR module, i.e. \( \models : S \times L \rightarrow \{\top, \bot\} \).

An update operator \( \circ \in U \) is a mapping \( \circ : S \times L \rightarrow S \) facilitating transitions between agent’s mental states induced by applying update formulae from the KR language: updating the current mental state \( \sigma \) by an update operator \( \circ \) and a formula \( \phi \) results in a new state \( \sigma’ = \sigma \circ \phi \). In a BSM agent \( A = (M_1, \ldots, M_n, \tau) \) we additionally require that the KR languages (and consequently the set of query and update operators) of any two modules are disjoint, i.e. \( L_i \cap L_j = \emptyset \).

**Syntax** A primitive query \( \phi = (\models \varphi) \) of a query operator \( \models \in \mathcal{Q} \) and a formula \( \varphi \in L \) of the same KR module. Arbitrary queries can be composed by means of conjunction \( \land \), disjunction \( \lor \) and negation \( \neg \).

Mental state transformers, syntactical counterparts to KR module updates, enable transitions from one state to another. A primitive mst \( \rho = \circ \phi \), constructed from an update operator \( \circ \in U \) and a formula \( \phi \in L \), is an update on the state of the corresponding KR module of a mental state. Conditional mst’s are of the form \( \phi \rightarrow \tau \), where \( \phi \) is a query and \( \tau \) is a mst. Such a conditional mst allows to make the application of mst \( \tau \) conditional on the evaluation of query \( \phi \). Mst’s can be combined by means of the choice | and the sequence o syntactic constructs.

**Definition 1 (mental state transformer)** Let \( M_1, \ldots, M_n \) be KR modules of the form \((S_i, L_i, Q_i, U_i)\). The set of mental state transformers is defined as:

1. \( \text{skip} \) is a primitive mst,
2. if \( \circ \in U_i \) and \( \psi \in L_i \), then \( \circ \psi \) is a primitive mst,
3. if \( \phi \) is a query, and \( \tau \) is a mst, then \( \phi \rightarrow \tau \) is a conditional mst,
4. if \( \tau \) and \( \tau’ \) are msts, then \( \tau \tau’ \) is an mst (choice) and \( \tau \circ \tau’ \) is an mst (sequence).

**Semantics** The BSM semantics is defined using a semantic calculus similar to that used for Gurevich’s Abstract State Machines [8]. This formalism provides a functional, rather than an operational, view on mental state transformers. The yields calculus, introduced below, specifies an update associated with executing an mst. It formally defines the meaning of the state transformation induced by executing an mst in a mental state.

Formally, a mental state \( \sigma \) of a BSM agent \( A = (M_1, \ldots, M_n, \tau) \) consists of the corresponding states \( (\sigma_1, \ldots, \sigma_n) \) of its KR modules. To specify the semantics of a BSM agent, first we need to define how queries are evaluated and how a state is modified by applying updates to it. A primitive query \( \models \models \varphi \) in a state \( \sigma = (\sigma_1, \ldots, \sigma_n) \) evaluates the formula \( \varphi \in L \) using the query operator \( \models \models \in Q \) in the current state \( \sigma_i \) of the corresponding KR module \((S_i, L_i, Q_i, U_i)\). That is, \( \sigma_i \models (\models \models \varphi) \) holds in a mental state \( \sigma \) if \( \sigma_i \models \models \varphi \), otherwise we have \( \sigma_i \not\models (\models \models \varphi) \). Given the usual meaning of Boolean operators, it is straightforward to extend the query evaluation to compound query formulae. Note that a query \( \models \models \varphi \) does not change the current mental state \( \sigma \).

The semantics of a mental state transformer is an update set: a set of (possibly sequences of) updates. The same notation \( \circ \psi \) (skip) is used to denote a simple update as well as the corresponding primitive mst. It should be clear from the context which of the two is intended. Sequential application of updates is denoted by \( \bullet \), i.e. \( \rho_1 \bullet \rho_2 \) is an update resulting from applying \( \rho_1 \) first and then applying \( \rho_2 \).

**Definition 2 (applying an update)** The result of applying an update \( \rho = \circ \psi \) to a state \( \sigma = (\sigma_1, \ldots, \sigma_n) \) of a BSM agent \( A = (M_1, \ldots, M_n, \tau) \), denoted by \( \sigma \circ \rho \), is a new state \( \sigma’ = (\sigma_1, \ldots, \sigma_n) \) where \( \sigma_i’ = \sigma_i \circ \psi_i \) for each \( \sigma_i \), \( \circ \) and \( \psi \) correspond to one and the same \( M_i \) of \( A \). Applying the special update skip to a state results in the same mental state \( \sigma = \sigma \circ \text{skip} \).

The result of applying an update of the form \( \rho_1 \circ \rho_2 \) to a state \( \sigma \), i.e. \( \sigma \circ (\rho_1 \circ \rho_2) \), is the new state \( \sigma \circ \rho_1 \circ \rho_2 \).

Note, that since we assume disjoint sets of query/update operators for different KR modules, a formula \( \circ \psi \) unambiguously corresponds to a single KR module.

The meaning of a mental state transformer in state \( \sigma \), formally defined by the yields predicate below, is the update it yields in that mental state. For the purpose of this paper, we introduce a slightly simplified, more convenient definition of the yields calculus originally published in [16, 17].

**Definition 3 (yields calculus)** A mental state transformer \( \tau \) yields an update \( \rho \) in a state \( \sigma \), i.e. \( \text{yields}(\tau, \sigma, \rho) \) is derivable in the following calculus:

\[
\begin{align*}
\text{yield}(\text{skip}, \sigma) &= \text{skip} \\
\text{yield}(\circ \psi, \sigma, \circ \psi) &= \circ \psi \\
\text{yield}(\circ \rho, \sigma, \circ \rho) &= \circ \rho \\
\text{yield}(\rho_1 \circ \rho_2, \sigma, \rho_1 \circ \rho_2) &= \rho_1 \circ \rho_2 \\
\text{yield}(\tau, \sigma, \rho_1) &= \text{yield}(\tau \circ \rho_1, \sigma, \tau \circ \rho_1) \\
\text{yield}(\tau, \sigma, \rho_2) &= \text{yield}(\tau \circ \rho_2, \sigma, \tau \circ \rho_2) \\
\text{yield}(\tau, \sigma, \rho_1 \circ \rho_2) &= \text{yield}(\tau \circ (\rho_1 \circ \rho_2), \sigma, \tau \circ (\rho_1 \circ \rho_2))
\end{align*}
\]

The mst yields the update skip. Similarly, a primitive update \( \circ \psi \) yields the corresponding update \( \circ \psi \). In case the condition of a conditional mst \( \phi \rightarrow \tau \) is satisfied in the current mental state, the calculus yields one of the updates corresponding to the right hand side mst \( \tau \), otherwise the skip update is yielded. A non-deterministic choice mst yields an update corresponding to either of its members and finally a sequential mst yields a sequence of updates corresponding to the first mst of the sequence and an update yielded by the second member of the sequence in a state resulting from application of the first update to the current mental state.

The collection of all the updates yielded w.r.t. the Definition 3 comprises an update set of an agent program \( \tau \) in the current mental state \( \sigma \). The semantics of the agent \( A = (M_1, \ldots, M_n, \tau) \) is then defined as a set of all, possibly infinite, computation runs \( \sigma_1, \sigma_2, \ldots, \sigma_i, \ldots \), the agent can take during its lifetime, s.t. for each pair \( \sigma_i, \sigma_{i+1} \), there exists an update \( \rho \) which is yielded by \( \tau \) in \( \sigma_i \) (i.e. \( \text{yields}(\tau, \sigma_i, \rho) \) and \( \sigma_{i+1} = \sigma_i \circ \rho \)).

2.1 Jazzyk

Jazzyk is an interpreter of the Jazzyk programming language implementing the computational model of the BSM framework. In the examples later in this paper, we use a more readable notation mixing
the syntax of Jazzyk with that of the BSM mst’s introduced above. When $\phi$ then $\tau$ encodes a conditional $\text{mst } \phi \rightarrow \tau$. Symbols $;$ and $,$ stand for choice and sequence $\circ$ operators respectively. To facilitate operator precedence, mental state transformers can be grouped into compound structures, blocks, using curly braces $\{ \ldots \}$.

To better support source code modularity and re-usability, Jazzyk interpreter integrates GNU M4\(^2\), a state-of-the-art macro processor. Macros are a powerful tool for structuring and modularizing and encapsulating the source code and writing code templates. GNU M4 macros are defined using a statement $\text{define}(\text{identifiers}, \text{<body>})$ and expanded whenever a macro identifier is instantiated in the source code. Before feeding the Jazzyk agent program to the language interpreter, first all the macros are expanded.

For further details on the Jazzyk programming language and the macro processor integration with the Jazzyk interpreter, consult [17]. Examples throughout this paper will use macros implementing parts of the Jazzbot agent program as standalone mental state transformers.

### 3 DESIGN & IMPLEMENTATION

The architecture of agents as Behavioural State Machines splits the agent program into two distinct layers: the knowledge representation layer and the behavioural layer. While the concern dealt with in the KR layer is modeling agent’s beliefs about its environment and its own mental attitudes, the BSM computational model facilitates implementation of agents’ behaviours. The two are coupled by invocation of query and update operators of KR modules.

In the following we discuss considerations and issues we faced when developing a BDI inspired cognitive agent in the BSM framework. We accompany our discussion with examples adapted from the Jazzbot project implementation.

#### 3.1 Jazzbot

To demonstrate the applicability of the framework of Behavioural State Machines and the Jazzyk language, we implemented Jazzbot, a virtual agent embodied in a simulated 3D environment of a first-person shooter computer game Nexuiz\(^3\).

In [17], we introduce the architectural details of the Jazzbot project. Jazzbot is a goal-driven agent. It features four KR modules representing belief base, goal base, and an interface to its virtual body in a Nexuiz environment respectively. While the goal base consists of a single knowledge base realized as an ASP logic program, the belief base is composed of two modules: Answer Set Programming\(^4\) based one and a Ruby\(^4\) module for representing the map of the bot’s environment. The interface to the environment is facilitated by a Nexuiz game client module. The Figure 1 depicts the structure of the Jazzbot application.

Jazzbot’s behaviours are implemented as a Jazzyk program. Jazzbot can fulfill e.g. search and deliver tasks in the simulated environment, it avoids obstacles and walls. Figure 1 depicts the architecture of Jazzbot and features an example Jazzyk code chunk implementing a simple behaviour of picking up an object by a mere walk through it and then keeping notice about it in its ASP belief base. Note that all the used KR modules are compatible with each other, since they share the domain of character strings. Hence all the variables used in Jazzbot’s programs are meant to be character string variables.

\(^2\)http://www.gnu.org/software/m4/
\(^3\)http://www.allentrep.org/nexuiz/
\(^4\)http://www.ruby-lang.org/

Below we describe our considerations while designing and implementing the Jazzbot softbot. For closer details on the architectural design of the Jazzbot’s components consult [17].

#### 3.2 Knowledge representation layer

Our aim is to demonstrate the flexibility of the BSM framework in a BDI-inspired agent system. As described already above, Jazzbot features two independent knowledge bases: a belief base and a goal base. Additionally the embodiment of the bot requires an interface to its body (and thus to the environment). Jazzbot thus features KR modules labeled beliefs ($B$), goals ($G$) and body ($E$). While the first two are implemented as logic programming based knowledge bases in Answer Set Programming\(^3\), the last one is realized as a connector to a Nexuiz game server. Formally, the body is represented by a KR module $E = (\mathcal{L}_{\text{Nexuiz}}, \{\models_E\}, \{\models_E\})$. It uses a special purpose language $\mathcal{L}_{\text{Nexuiz}}$ for query/update formulae and two query/update operators $\models_E$, $\models_E$ accepting formulae from this language and evaluating them against the simulated environment represented by the game server.

**Belief base**

Jazzbot’s belief base $B$ contains a logic program describing agent’s beliefs about its environment and itself. It is supposed to closely reflect agent’s perceptions of the environment, i.e. updates of belief base correspond to agent’s perceptions, while queries can also include higher level consequences of primitive perceptions w.r.t. the logic program in $B$.

In Jazzbot we exploit the power of non-monotonic reasoning for capturing relations and interactions between various beliefs an agent can hold. Formally, the Jazzbot’s belief base $B = (\text{AnsProlog}^*, \{\models_B\}, \{\models_B, \models_B\})$ uses AnsProlog\(^*\) [3], the language of ASP logic programs and features an entailment query operator $\models_E$ evaluating a query formula $\varphi$ true iff it holds in all answer sets of the logic program representing the actual belief base. The updates $\models_B$ and $\models_B$ correspond to trivial assert and retract of a formula respectively\(^5\).

\(^5\)In the long run we consider more complex belief revision operators realizing extension and contraction operators similar to those used in Dynamic Logic Programming [14].
wander around the environment. Survival requires the bot to explore their interactions and derive non-trivial goals, or subgoals from the adopted goals (primitive facts), the agent can thus also reason about special subscript for its operators. Besides holding a set of currently

The Listing 1 shows a logic program representing a part of the agent’s initial belief base. The Jazzbot’s belief base facilitates reasoning about objects the bot believes to posses, its health status and other players. The bot perceives its health status as a numeric value, from which it derives its own state in the game. It is also able to perceive objects and other players in the environment and it reasons about them as well. For example, it considers a player it actually sees as a friend, if it does not feel threatened by him. In an extended example, the bot could also reason about its roles in the game and keep long-term beliefs about other players.

To represent the bot’s information about the topology of its environment, we employ a module implemented in an object oriented scripting language Ruby. Since a description of it’s functionality is not essential for the purposes of this paper, we do not provide a closer description of its internal functionality.

**Goal base** Goals are usually meant to provide a declarative description of situations (states) an agent desires to be in (goals to-be), or activities it desires to perform (goal to-do). Each goal triggers a certain behaviour of the agent designed to satisfy it. Under some conditions w.r.t. the state of agent’s beliefs, the agent adopts a goal and according to its type, it might eventually drop it, i.e. a goal also comes with an associated commitment strategy.

Because of the separation of concerns between the agent’s belief and goal bases in the BSM framework, the interactions between belief base conditions and goals have to be expressed explicitly in the form of internal behaviours (causal updates). Therefore, rather than providing a concrete fixed logic-based semantics for goals and their associated commitment strategies, we propose viewing these as mere context holders, or behaviour drivers. The main purpose of a goal is then to implicitly represent a condition on beliefs which is to be achieved (or avoided) and enable execution of a behaviour designed to eventually satisfy this condition in the future. In this view, goals of an agent are only loosely coupled with its beliefs.

Formally, the Jazzbot’s goal base $G = (\text{AnsProlog}^*, \triangleright\triangleright, \{\triangleright\triangleright, \triangleright\triangleright, \triangleright\triangleright\})$ is technically equivalent to the belief base $B$, however because the BSM framework requires disjoint KR modules, we use a special subscript for its operators. Besides holding a set of currently adopted goals (primitive facts), the agent can thus also reason about their interactions and derive non-trivial goals, or subgoals from the more primitive ones.

The Listing 2 provides a logic program encoding a part of agent’s initial goal base. Initially the bot has two maintenance goal: to survive and to be happy as well as a single achievement goal to get the box identified as box(42). To satisfy the goal to be happy, the bot activates behaviours which are triggered by tasks to communicate and wander around the environment. Survival requires the bot to explore its environment as well as to seek safety and energy sources.

In the course of its lifetime, the bot might adopt goals regarding getting objects from the environment. Unless some exceptional conditions are met, e.g. for whatever reason the bot desperately looks for a medikit object, the goal to get an object from the environment activates also goals to find, pick and deliver the object. The order of execution of the three subgoals will be specified implicitly by encoding of the bot’s individual behaviours. Searching for the medikit object is defined using specialized rules, as this special object does not have to be delivered anywhere.

Similarly to the bot’s belief base, the goal base contains a logic program for reasoning about agent’s goals. In the case a larger program is contained in the bot’s goal base, this raises a question which parts of the particular goal base language (literals used in the logic program) are to be interpreted as behaviour triggers and which serve only for reasoning about interactions between goals. To solve this problem, we divide the goal base language into two parts: declarative goals handling and handling of tasks. The goal base then facilitates a breakdown of declarative achievement goals, such as achieve(get(box(42))), into tasks, behaviour triggers, such as task(search(X)). This way, we moved the reasoning about goal interactions completely into the goal base, instead of handling it inside the agent program, as it is done in other agent programming languages, such as Jason [7]. In the following we will show, that this technique results in implementation of behaviours, execution of which is triggered by merely checking the associate trigger literal in the goal base.

### 3.3 Behavioural layer

The choice of agent’s KR modules and their ascribed purposes drives the implementation of agents behaviours. These can be either exogenous - resulting in selecting an action (or a sequence of actions) to be executed in the agents environment, or endogenous - implementing interactions between agent’s knowledge bases. Analysis of information flows between agent’s KR modules straightforwardly leads to identification of the individual compound behaviours.

---

**Listing 1** Jazzbot’s belief base implementation in AnsProlog*.

```prolog
% Initially the bot does not hold the box %
% The bot can later hold other objects as well %
¬hold(box(42)).

% Reasoning about the health status %
alive ←¬ health(X), X < 0.
dead ← ¬ health(X), X =< 0.
attacked ← ¬ health(X), X =< 90.
wounded ← ¬ health(X), X =< 50.

% Reasoning about friendliness of other players %
friend(id) ← see(player(id)), not attacked, player(id).
enemy(id) ← see(player(id)), not friend(id), player(id).

player(1..5).
```

**Listing 2** Jazzbot’s goal base implementation in AnsProlog*.

```prolog
% Initially the bot has two maintenance goals and %
% a single achievement goal. %
maintain(happy).
maintain(survive).
achieve(get(box(42))).

% Subgoals of the goal maintain(happy) %
task(communicate) ← maintain(happy).
task(wander) ← maintain(happy).

% Subgoals of the goal maintain(survive) %
task(safety) ← maintain(survive).
task(energy) ← maintain(survive).

% Subgoals of the goal achieve(get(Object)) %
task(search(X)) ← achieve(get(X)), not achieve(get(medikit)), item(X).
task(pick(X)) ← achieve(get(X)), not achieve(get(medikit)), item(X).
task(deliver(X)) ← achieve(get(X)), not achieve(get(medikit)), item(X).
% Specialized subgoals of the goal achieve(get(medikit)) %
task(search(medikit)) ← achieve(get(medikit)).
task(pick(medikit)) ← achieve(get(medikit)).
% Ressurect after being killed %
task(reborn) ← achieve(reborn).
% Definition of items %
item(medikit).
item(X) ← box(X), box(1..50).
```
The Jazzbot’s KR layer consists of the following KR modules: beliefs, goals and the bot’s body. This gives rise to three information flows in the agent system: body → beliefs → goals → body. In accord with the usual understanding of BDI model of rationality [18], these represent respectively the following principles: 1) an agent senses its environment and reflects its perceptions in its beliefs (perceptions); 2) because it believes to be in a certain situation, it updates its goals (commitment strategies); and finally 3) to achieve the adopted goals, it acts in the environment (action selection).

In the following we discuss these individual behaviors in detail.

Perceptions Jazzbot agent roams around the simulated 3D environment and keeps track of its perceptions regarding its surroundings and a state of its own body. The actual state of beliefs is supposed to reflect its perceptions about the world and relations between them. In general, provided a sensory information \( \varphi \) from the agent’s environment (body), following the information flow notation used above, we can encode a corresponding update \( \psi \) of an agent’s belief base as a conditional mental state transformer

\[
\models E \varphi \rightarrow \otimes B \psi
\]

A set of such conditional mst’s captures the relations between various perceptions and their belief counterparts. Now, according to the chosen model of perception, a designer can form a proper BSM mst \( \tau_{\text{perc}} \) from this set by joining the mst’s by either non-deterministic choice \( | \) or sequence \( \circ \) operators. In a more advanced setting, beyond the scope of this paper, the designer can even choose to further structure them using nested conditional mst’s.

Listing 3 Implementation of Jazzbot’s perceptions handling.

```java
define('PERCEIVE',
{    // Check the health sensor
    when \models F [\{ body health X \}] then
    {        // Before updating with the new value, retract the old one
        when \models B [\{ health(Y) \}] then \models B [\{ health(Y) \}]
        \models B [\{ health(X) \}]
    },
    // Check whether the bot still sees an object it remembers
    when \models B [\{ see(Id, Type) \} and not \models F [\{ see(Id, Type) \}]
    then \models B [\{ see(Id, Type) \}],
    // Check the camera sensor
    when \models F [\{ eye see Id Type \}]
    then \models B [\{ see(Id, Type) \}],
    ...
})
```

The Listing 3 shows an example encoding an mst implementing the bot’s perception of its own health in the game as well as recognition of objects in the vicinity of the bot. Note, that the Jazzbot checks its sensors in a sequence. Perception can be considered a safe sequential behaviour, as checking sensors takes only a little time and since the bot does not act in the environment, this behaviour can always finish without an interruption.

Goal commitment strategies A specific goal can be adopted because an agent believes its adoption is appropriate. Similarly, because of a certain state of beliefs, the agent might decide to drop a goal: for instance, a goal can be satisfied, or believed to be impossible to achieve, etc. Informally, a set of such internal behaviours related to a single goal implements a commitment strategy associated with it. Moreover, a designer can implement different commitment strategies w.r.t. various goals. Commitment strategies thus realize the second component of the information flow between the agent’s knowledge bases: given a condition on agent’s beliefs \( \varphi \), the agent updates its goal base by a goal formula \( \psi \). The corresponding mst loosely follows the scheme

\[
\models G \varphi \rightarrow \otimes G \psi
\] (3)

Here, the entailment operator \( \models G \) represents the belief base entailment operator and \( \otimes G \) is a corresponding goal base update operator (in the Jazzbot setting \( \otimes G \in \{ \models B, \models C \} \)). Similarly to perceptions, the agent designer can join and structure the mst’s realizing commitment strategies of individual goals the agent can adopt during its lifetime using BSM composition operators. \( \tau_{\text{cs}} \) will denote the resulting mst.

Listing 4 Implementation of Jazzbot’s goal commitment strategies handling.

```java
define('HANDLE_GOALS',
{
    // Adoption and dropping of the goal to get medikit +/-
    when \models B [\{ wounded \}]
    then \models B [\{ get(medikit) \}]
    \models B [\{ get(medikit) \}],
    when \models B [\{ get(medikit) \}]
    and not \models B [\{ wounded \}]
    then \models B [\{ get(medikit) \}],

    // When the bot receives a user command, it obeys +/-
    when \models B [\{ command(get(X)) \}]
    then \models B [\{ achieve(get(X)) \}],
    when \models B [\{ achieve(get(X)) \}]
    and \models B [\{ holds(X) \}]
    then \models B [\{ achieve(get(X)) \}],

    // When the bot finds out it was killed, it resurrects in the game +/-
    when \models B [\{ dead \}]
    then \models B [\{ achieve(reborn) \}],
    when \models B [\{ alive \}]
    then \models B [\{ achieve(reborn) \}],
    ...
})
```

The Listing 4 provides an example of implementation of commitment strategies w.r.t. bot’s achievement goals. When the bot believes it was wounded in the game, it adopts a goal to get the medikit object to refresh its health. Sometime after it starts to believe that it found it, it drops the goal. Similarly, the bot implements custom handling for each achievement goal it can deal with. It is responsive to user commands and is able to get an item on request and finally when it detects that it was terminated in the game, it adopts a goal to get back into it. Adopted goals, subsequently trigger behaviours which should achieve them. Note, that the individual mst’s implementing goal commitment strategies are joined together using the non-deterministic choice operator. This way, the bot is allowed to adopt, or drop a goal only once per an execution cycle.

Goal oriented behaviours: action selection The core task of an agent, is to perform behaviours (actions) in the environment. In the setting introduced above, behaviours have a purpose: satisfaction of adopted goals. We speak therefore about goal oriented behaviours. In general, following the information flow notation, they amount to choosing an appropriate action \( \psi \) for achieving a goal \( \varphi \), the agent currently pursues:

\[
\models G \varphi \rightarrow \otimes G \psi
\]

with \( \models G \) representing an entailment operator on the goal base and \( \otimes G \) being the body/environment update operator.
In our experience, we quickly found that the agent’s core behaviour, the action selection mechanism, often requires a more complex structuring than the, rather reactive, scheme 3 prescribes. First, there can potentially be several behaviours supposed to achieve the same goal in possibly different contexts, and second, in different contexts, a single behaviour might be appropriate for achieving several different goals. We therefore extend the scheme 3 above as follows:

$$\phi_G \land \phi_B \rightarrow \tau$$  \hspace{1cm} (4)$$

$\phi_G$ represents a pursued goal query, $\phi_B$ is a belief context guard, and $\tau$ is a possibly compound behaviour query associated with (some of the) goal(s) represented by $\phi_G$. To support source code modularity, the Jazzbot interpreter integrates a powerful macro preprocessor. Thus in different contexts re-usable mst’s can be wrapped into named macros and simply expanded at places in the code, where they are applied.

Various behaviours of an agent are combined in various ways in different contexts and situations. In certain contexts (e.g. emergency situations), where a tight behaviour control is required, script-like compound behaviours (also called ballistic [1]) are more appropriate. However, more often we want the agent to interleave behaviours associated with orthogonal, not interfering, goals. The design choices are therefore application specific and left to the designer. We denote the compound mental state transformer implementing agent’s action selection as $\tau_{act}$.

**Listing 5** Implementation of Jazzbot’s behaviour selection.

```c
define(‘ACT’);
{  
  \(/\) Behaviours for getting an item +/-  
  \(/\) The bot searches for an item, only when it does not have it +/-  
  when $\models_G ([\text{task}(\text{search}(X))])$ and not $\models_B ([\text{hold}(X)])$  
  then SEARCH(X) ;  
  \(/\) When a searched item is found, it picks it +/-  
  when $\models_G ([\text{task}(\text{pick}(X))])$ and $\models_B ([\text{see}(X)])$  
  then PICK(X) ;  
  \(/\) When the bot finally holds the item, it deliver it +/-  
  when $\models_G ([\text{task}(\text{deliver}(X))])$ and $\models_B ([\text{hold}(X)])$  
  then DELIVER(X) ;  
  \(/\) Simple behaviour triggers without guard conditions +/-  
  when $\models_G ([\text{task}(\text{reborn})])$ then REINCARNATE ;  
  when $\models_G ([\text{task}(\text{wander})])$ then WALK ;  
  when $\models_G ([\text{task}(\text{safe})])$ then RUN_AWAY ;  
  when $\models_G ([\text{task}(\text{comm})])$ then SOCIALIZE ;  
  \(\ldots\)  
}
```

The Listing 5 provides an example code implementing selection of goal oriented behaviours, realized as parametrized macros, satisfying Jazzbot’s goals. While the bot simply triggers behaviours for reincarnation, walking around, danger aversion and social behaviour, the execution of behaviours finally leading to getting an item are guarded by belief conditions. This way, we introduce an order on these behaviours. Recall, that in the goal base, the goal to get an item triggers the tasks to search for it, pick it up and deliver it simultaneously. Using a different handling of goals directly in the goal base, we could implement ordering of the goals already there and then trigger the individual behaviours without a belief base guard condition.

### 3.4 Control cycle

Putting together the previously designed mst’s implementing the agent’s model of perception $\tau_{perc}$, its goals commitment strategies $\tau_{cs}$ and the agent’s behaviour, i.e. the action selection mechanism $\tau_{act}$, we implement a control cycle of the BSM agent program. According to ordering and combination of the mst’s a designer can 1) develop the control model of the agent, as well as 2) control determination of the agent program. In the Jazzbot example we could consider either a case in which in a single computation step the bot non-deterministically either perceives, handles its goals, or acts: $\tau_{perc} \circ \tau_{cs} \circ \tau_{act}$, or sequentially executes all the stages: $\tau_{perc} \circ \tau_{cs} \circ \tau_{act}$. Different orderings of the mst’s yield different overall behaviours of the agent as well.

As we already discussed above, according to the internal structure of the partial mst’s of the BSM agent program, the agent can for example either check all its sensors in a single cycle, or consider only one of them non-deterministically. Similarly for the goal commitment strategies and action selection mechanism. Different structuring of the partial mst’s inside $\tau_{perc}$, $\tau_{cs}$ and $\tau_{act}$ allows a programmer to implement various control models. Below we provide few examples of partial control models expressed using an LTL-like notation [15]:

$$\bigcirc [\models_G \varphi_G \rightarrow \tau_G]$$

**Listing 6** Implementation of Jazzbot’s control cycle.

```c
define(‘ACT’);
{  
  \(/\) The actual Jazzbot agent program +/-  
  PERCEIVE , HANDLE_GOALS , ACT
}
```

Finally, the Listing 6 sums up the running example of this paper. It provides the implementation of the control cycle implemented in Jazzbot using the macros defined in the previous subsections. Note, that the bot executes in every step all the stages of its control cycle sequentially.

### 4 PUTTING IT TOGETHER

Finally, we can put together a set of more general design guidelines for development of embodied agent systems implemented in a rule-based languages similar to BSM. In this paper we focus on agents featuring belief and goal bases. The central element around which our design considerations revolved were agent’s goals.

Goals determine currently active (enabled) behaviours and thus serve as a trigger for agent’s exogenous behaviours, which are the
visible manifestations of its functionality. Additionally, a goal is associated with a commitment strategy steering its adoption and satisfaction, or dropping (both determined by conditions on agent’s beliefs). This leads to a notion of goal oriented behaviours.

A set of goal oriented behaviours can be characterized by a tuple \( (\phi, \kappa_\phi, \kappa_\Sigma, \tau) \), where in the case of Jazzbot agent, \( \phi \in L_{ASP} \) is a goal, \( \kappa_\phi, \kappa_\Sigma \subseteq \{ \tau \in L_{ASP} \} \) are sets of its adopt and drop conditions w.r.t. the belief base \( B \) respectively, and \( \tau \subseteq \{ \tau | \tau \text{ is an mst} \} \) is a set of behaviours triggered by \( \phi \). Informally, a commitment strategy behaviour \( \tau_{\text{cs}} \), then should contain conditional mst’s of the form \( \Rightarrow \phi \longrightarrow \Box g \phi \) with \( \Box g \phi \in \{ \Box g B, \Box g \Sigma \} \) and \( \phi \in \kappa_\phi \) being either an adopt, or a drop condition. \( \tau \) then contains conditional mst’s of \( \tau_{\text{act}} \) similar to \( \Rightarrow \phi \longrightarrow \tau \), where \( \tau \in \tau \). A very similar view can be formulated for beliefs featuring belief adoption and drop conditions (w.r.t. agent’s perceptions) and being loosely associated with adopt/drop conditions of agent’s goals. In a consequence, such considerations would lead to a formal characterization of causal information flows between the agent’s knowledge bases, a topic beyond the scope of the presented work.

By generalizing the presented approach to development of Jazzbot we arrive to the following methodological steps/guidelines for designing BSM agents:

1. Identify the set of agent’s goals and design their interactions w.r.t. the employed KR technology.
2. Design a set of behaviours \( \tau_{\text{act}} \) triggered by these goals (supposed to achieve them),
3. Identify the adoption and satisfaction (drop) conditions for these goals and design concrete commitment strategies for them in \( \tau_{\text{cs}} \),
4. Identify the relevant part of agent’s beliefs w.r.t. the conditions associated with the goals,
5. Design the agent’s belief base including appropriate belief relationships w.r.t. the employed KR technology,
6. Design the model of perception \( \tau_{\text{perc}} \), by identifying the percepts of the agent and link them to corresponding beliefs,
7. Finally construct the global BSM agent program by appropriately structuring and combining the mental state transformers \( \tau_{\text{perc}}, \tau_{\text{cs}} \) and \( \tau_{\text{act}} \) into a control cycle.

The presented guidelines, centered around the notion of a goal, loosely fit the general view of methodologies for agent-based systems like e.g. Tropos, or MaSE [4]. In these, one of the main results of the analytical stage of a single agent system are agent’s goals, or tasks, associated with agent’s roles. Such methodologies are usually not coupled to a particular agent architecture, the details of the agent design are therefore left to a particular platform. The guidelines proposed here thus informally fill this gap, at least w.r.t. the BSM framework. However, we are convinced that some of the considerations discussed above apply also to other agent oriented rule-based language, especially when considering heterogeneous KR technologies in a single agent.

5 DISCUSSION & CONCLUSION

To our knowledge, not too much was reported on implementation techniques and source code structuring of larger, non-trivial agent systems in agent oriented programming languages like Jason, or 3APL [6]. Some sketchy notes on overall system design can be found in Jason and 2APL team reports from Multi-Agent Programming Contest 2007 and 2008 [2, 12, 13], however no more general methodological considerations are discussed there and authors focus solely on design of their particular agent system. From the published source code of these projects\(^6\), it can be seen that the authors extensively use escape calls into Java code and the Java implementation comprised a significant part of their system implementations: representation of the agent’s environment, shortest point to point path planning, inter-agent coordination, etc. The framework of Behavioural State Machines makes calls to external code a first class concept of an agent programming language and facilitates only interactions between the agent’s knowledge bases.

This unconstrained approach allows for custom implementation of various strategies for handling agent’s mental attitudes and control models. While the mainstream approach in agent programming languages is to choose a set of constraints an agent system must obey, our approach is different. We propose a generic and flexible agent programming framework, even capable of emulation of other agent programming languages (see [11]), and subsequently develop a set of methodological and design guidelines, so to say rules of good practice, for development of cognitive agents. We accompany the discussion by examples of code patterns supporting implementation according to these design guidelines.

The presented discussion provides a snapshot of our current experience and knowledge in programming cognitive agents with the framework of Behavioural State Machines. We discuss ideas and issues resulting from an ongoing work towards proposing a set of design patterns for cognitive agent programming. The main contribution of the presented work is an attempt to lift these considerations into a set of methodological steps, partly also applicable to implementation of agents in other rule-based agent oriented programming frameworks (AgentSpeak(L) family of languages). However, the BSM framework was specifically designed in a liberal way and thus it allows use of heterogeneous knowledge representation technologies together with implementation of arbitrary interactions among them. Therefore, because of additional constraints these languages (mostly BDI oriented) impose on agent programs, some of the presented interactions between agent’s knowledge bases and implementation techniques (e.g. reasoning about agent’s goals in AnsaProlog\(^7\)) might be difficult to implement in them.

In the future research, we aim to study formal specification methods (like e.g. source code annotations) for cognitive agents, by trying to generalize examples of control models like those presented in Subsection 3.4. Subsequently we aim at characterizing more complex code structures and templates, by means of dynamic, or temporal logic adapted for the BSM framework. Our line of research follows a bottom-up approach: instead of proposing a way to design a system by analyzing it, we rather try to experiment with live implementations and collect experiences, which could later serve as a basis for a generalization.

REFERENCES


\(^6\) Communicated over Agent Contest mailing lists: publicly available at http://cig.in.tu-clausthal.de/agentcontest/.


Representation and Integration: Combining Robot Control, High-Level Planning, and Action Learning

Ronald Petrick, Dirk Kraft, Kira Mourão, Christopher Geib, Nicolas Pugeault, and Mark Steedman

Abstract. We describe an approach to integrated robot control, high-level planning, and action effect learning that attempts to overcome the representational difficulties that exist between these diverse areas. Our approach combines ideas from robot vision, knowledge-level planning, and connectionist machine learning, and focuses on the representational needs of these components. We also make use of a simple representational unit called an instantiated state transition fragment (ISTF) and a related structure called an object-action complex (OAC). The goal of this work is a general approach for inducing high-level action specifications, suitable for planning, from a robot’s interactions with the world. We present a detailed overview of our approach and show how it supports the learning of certain aspects of a high-level representation from low-level world state information.

1 INTRODUCTION AND MOTIVATION

The problem of integrating low-level robot systems with high-level symbolic planners introduces significant representational difficulties that must first be overcome. Since the requirements for robot-level control and vision tend to be different from that of traditional planning, neither representation is usually sufficient to accommodate the needs of an integrated system. Overcoming these representational differences is a necessary challenge, however, since both levels seem to be required to produce human-like behaviour.

In general, robot systems tend to use representations based on vectors of continuous values, which denote 3D spatial coordinates, joint angles, force vectors, or other world-level features that require real-valued parameters. Such representations allow system builders to succinctly specify robot behaviour since most of the computations required for low-level robot control are effectively captured as continuous transforms of continuous vectors over time. On the other hand, high-level planning systems typically use representations based on discrete, symbolic models of objects, properties, and actions, described in logical languages (e.g., [5, 23, 16, 27, 31]). Instead of modelling low-level continuous processes, these representations capture the dynamics of the world or the agent’s knowledge at a more abstract level, for instance by characterizing the state changes that result from deliberate, planned action.

In this paper we describe an approach for integrating a robot/vision system with a high-level planner, that attempts to overcome the representational challenges described above. In particular, our approach gives rise to a system that is capable of automatically inducing certain aspects of a high-level representation suitable for planning, from the robot’s interactions with the real world using basic “reflex” actions. This paper describes work currently in progress. As such, we do not address the entire problem of learning action representations, but instead focus on two important parts: object learning and action effect learning. Our approach uses a simple representational unit called an instantiated state transition fragment (ISTF) and a related structure called an object-action complex (OAC) [7], both of which arise naturally from the robot’s interaction with the world—and world objects in particular. These notions also help us address certain control problems, for instance the relationship between high-level sensing actions and their execution by the robot, and representational issues that arise at different levels of our system. Although we only consider a portion of a larger learning problem, we are also interested in implementing these ideas within a framework that includes the lowest-level control mechanisms right up to the high-level reasoning components. Finally, we believe our approach is general and that these ideas can be successfully transferred to other robot platforms and planners, with capabilities other than those we describe here.

To illustrate our approach, we will consider a simple robot manipulation scenario throughout this paper. This domain consists of a robot with a gripper, a table with a number of objects on it, and a “shelf” (a special region of the table). The robot has a camera to view the objects in the world but does not initially have knowledge of those objects. Instead, world knowledge must be provided by the vision system, the robot’s sensors, and the basic action reflexes built into the robot controller. The robot is given the task of clearing the objects from the table by placing them on the shelf. The shelf has limited space so the objects must be stacked in order for the robot to successfully complete the task. For simplicity, each object is assumed to be roughly cylindrical in shape and has a “radius” which provides an estimate of its size. An object $A$ can be stacked into an object $B$ provided the radius of $A$ is less than that of $B$, and $B$ is an “open” object. Unlike a standard blocks-world scenario, the robot will not have complete information about the state of the world. In particular, we will often consider scenarios where the robot does not know whether an object is open or not and must perform a test to determine an object’s “openness”. The robot will also have a choice of four different grasping types for manipulating objects in the world. Not all grasp types can be used on every object, and certain grasp types are further restricted by the position of an object relative to other objects in the world. The set of available grasp types is shown in Figure 1. Finally, actions in this domain can fail during execution and the robot’s sensors may return noisy data.

The rest of the paper presents an overview of our approach from a representational point of view, and discusses the main components.

1 School of Informatics, University of Edinburgh, Edinburgh EH8 9LW, Scotland, United Kingdom, contact e-mail: rpetrick@inf.ed.ac.uk.
2 The Maersk Mc-Kinney Møller Institute, University of Southern Denmark, DK-5230 Odense M, Denmark.
of our system. In Section 2 we describe the basic representations used in the paper. In Section 3 we discuss how object information is discovered from the robot/vision system’s initial experiences in the world. In Section 4 we describe the high-level planner and plan execution monitor. In Section 5 we introduce a mechanism for learning the effects of actions from state descriptions. In Section 6 we discuss the current state of implementation in our system and some early empirical results. Finally, in Section 7 we discuss the advantages of our approach from a representation point of view, and describe some areas of future work.

2 BASIC REPRESENTATIONS

At the robot/vision level, the system has a set Σ of sensors, Σ = \{σ₁, σ₂, ..., σₙ\}, where each sensor σᵢ returns an observation \textit{obs}(σᵢ) about some aspect of the world. The execution of a robot-level motor program or robot action may cause changes to the world which can be observed through subsequent sensing. Each motor program is typically executed with respect to particular objects in the world. We assume that the robot/vision system does not initially know about any objects and, thus, can’t execute many motor programs. Instead, the robot has a set of basic reflex actions that aren’t dependent on particular objects and can be used for exploring the world initially.

The planning level representation is based on a set of fluents, \{f₁, f₂, ..., fₘ\}: first-order predicates and functions that denote particular qualities of the world, robot, and objects. Fluents represent high-level (possibly abstract) counterparts of some of the properties the robot is capable of sensing. In particular, the value of a fluent is a function of the observations returned by the sensor set, i.e., \textit{fᵢ} = \textit{Γ}(Σ). Typically, each fluent depends on a subset of the sensor observations and not every sensor need map to a fluent (some sensors are only used at the lower control level). Fluents can also be parameterized by high-level versions of the objects known at the robot level.

A state is a snapshot of the values of all instantiated fluents at some point during the execution of the system. States represent a point of intersection between the low-level and high-level representations, since states are induced from a set of sensor observations and the corresponding sensor/fluent mappings (i.e., the functions \textit{Γ}). High-level actions represent abstract versions of some of the robot’s motor programs. Since all actions must ultimately be executed by the robot, each action is decomposable to a fixed set of motor programs. Thus, the robot’s interaction with the world can be viewed as a simple state transition system: the robot’s sensor observations give rise to a state description; executing an action brings about changes in the world that can be observed through subsequent sensing. More importantly, every interaction of this form provides the robot with an opportunity to observe a small portion of the world’s state space, which we refer to as an instantiated state transition fragment (ISTF) [7].

Formally, an ISTF is a tuple \(⟨sᵢ, \textit{MP}_i, \textit{Obj}_i, sᵢ₊₁⟩\), where \(sᵢ\) is the state that is sensed before applying the motor program instance \(\textit{MP}_i\), \(\textit{Obj}_i\) is the object that the motor program is defined relative to, and \(sᵢ₊₁\) is the state sensed after executing the motor program. Thus, an ISTF is a situated pair of an object and an action that captures a small fragment of the world’s state transition function. The states \(sᵢ\) and \(sᵢ₊₁\) contain snippets of the robot’s information about these states, some of which may be irrelevant to the action being applied.

We will also consider a second representational structure that results from generalising over instances of ISTFs. Such structures are referred to as object-action complexes (OACs) [7], and are similar to ISTFs, but contain only the relevant instantiated state information needed to predict the applicability of an action and its effects, with all irrelevant information pruned away. An OAC is defined by a tuple of the form \(⟨S, \textit{MP}, \textit{Obj}, sᵢ⟩\), where \(S\) and \(sᵢ\) are two states, \(\textit{MP}\) is a set of motor programs, and \(\textit{Obj}\) is a class of objects. In this case, \(S\) only describes those properties of the world state that are required to execute any of the motor programs in \(\textit{MP}\); when acting on an object of class \(\textit{Obj}\), \(sᵢ\) describes a world state which captures the properties changed by the motor program.

Typically, we consider ISTFs and OACs formed from partial state descriptions. Such descriptions arise since the robot is not always able to sense the status of all objects and properties of the world (e.g., occluded or undiscovered objects). We also note that the robot’s sensors may be noisy and, thus, there is no guarantee that state reports only contain correct information. Furthermore, certain sensor operations have associated resource costs (e.g., time, energy, etc.). For instance, the robot can perform a test to determine whether an object is open by “poking” the object to check its concavity. Such operations are only initiated on demand at the discretion of the high-level planning system.

3 VISION-BASED OBJECT DISCOVERY

The robot system includes a vision component based on an early cognitive vision framework [15] which provides a scene representation composed of local 3D edge descriptors that outline the visible contours [26]. Because the system initially lacks knowledge of the objects that make up the scene, the visual representation is unsegmented: descriptors that belong to one object are not explicitly distinct from the ones that belong to other objects, or the background.

To aid in the discovery of new objects, the robot is equipped with a basic reflex action [1] that is elicited by specific visual feature combinations in the unsegmented world representation. The outcome of these reflexes allows the system to gather knowledge about the scene, which is used to segment the visual world into objects and identify basic affordances. We consider a reflex where the robot tries to grasp a planar surface in the scene. Each time the robot executes such a reflex, haptic information allows the system to evaluate the outcome: either the grasp was successful and the gripper is holding something, or it failed and the gripper simply closed. A failed attempt forces the system to reconsider its original assumptions, whereas a successful attempt confirms the feasibility of the reflex. Once a successful grasp is performed, the robot gains physical control over this part of the scene. If we assume that the full kinematics of the robot’s arm are known (which is true for industrial robots), then it is possible to segment the grasped object from the rest of the visual world as it is the only part that moves synchronously with the robot’s arm.

With physical control, the system visually inspects an object from

Figure 1. Available grasping types in the robot manipulation scenario

---

3 We do not focus here on the problem of learning high-level action schema (i.e., the set of action names and their parameters) or the action/motor program mappings. Instead, we assume that the action schema are provided with the corresponding mappings to robot-level motor programs.
a variety of viewpoints and builds a 3D representation [13]. Features on the object are tracked over multiple frames, between which the object moves with a known motion. If features are constant over a series of frames they become included in the object’s representation, otherwise they are assumed not to belong to the object. (A more detailed description of the accumulation process can be found in [13].) The final description is labelled and recorded as an identifier for a new object class, along with the successful reflex (now a motor program). Using this new knowledge, the system then reconsiders its interpretation of the scene: using a representation-specific pose estimation algorithm [3] all other instances of the same object class are identified and labelled. By repeating this process, the system constructs a representation of the world objects, as instances of symbolic classes that carry basic affordances, i.e., particular reflex actions that have been successfully applied to grasp objects of this class.

The technical implementation of the pose estimation algorithm has only recently become available. Prior to this, a circle detection algorithm was developed to recognise cylindrical objects, to which the domain was restricted for this work. Four grasp templates were used to define the primitive reflex actions in an object-centric way (where concrete grasps were generated based on the object pose). Although this approach negates the need for the general pose estimation algorithm, the conclusions drawn from experiments in this limited scenario are still easily transferable to the general case.

Figure 2 illustrates the “birth of an object.” In (a), the dots on the image show the predicted structures. Both spurious primitives, parts of the background that are not confirmed by the image, and the confirmed predictions are shown. In (b), the shape model learned from the object in (a) is shown. In (c) and (d), two additional objects are shown along with their learned shape models. The "gap" in the shape models corresponds to where the robot’s gripper held the objects.

The object-centric nature of the robot’s world exploration process has immediate consequences for the high-level representation. First, newly discovered objects are reported to the planning level and added to its representation. At the planning level, objects are simply labels while the real-world object information is stored at the robot level. Such a representation means that we can avoid certain types of real-valued information at the high level (e.g., 3D location coordinates, object orientation vectors, etc.) and instead refer to objects by their labels (e.g., obj1 may denote a particular red cup on the table). With the addition of new objects, the planning system can immediately start using such objects in its reasoning and plan construction. Since we assume that object names do not change over time, high-level object labels act as indices into the low-level object representation. Thus, plans with object references will be understandable to the robot/vision system. Finally, the successful identification of new objects will cause the robot/vision system to start sending regular state updates to the planning level about these objects and their properties. In particular, the ISTFs that result from subsequent interactions with the world will contain state information about these objects, provided they can be sensed by the robot. The planning level can then use this information for plan construction and plan execution monitoring. Additional details about the link between the robot/vision and planning systems are given in Section 6.

4 PLAN GENERATION AND MONITORING

The high-level planner is responsible for constructing plans that direct the behaviour of the robot in order to achieve a set of goals. For instance, in our example domain a plan might be constructed to clear all “open” objects from the table. Plans are built using PKS (“Planning with Knowledge and Sensing”) [24, 25], a knowledge-level conditional planner that can operate with incomplete information and high-level sensing actions. Like other symbolic planners, PKS requires a goal, a description of the initial state, and a list of the available actions before it can construct plans. Unlike traditional approaches, PKS operates at the knowledge level by modelling the agent’s knowledge state, rather than the world state. By doing so, PKS can reason efficiently about certain types of knowledge, and make effective use of non-propositional features, like functions, which often arise in real-world scenarios.

PKS is based on a generalization of STRIPS [5]. In STRIPS, a single database represents the world state. Actions update the database in a way that corresponds to their effects on the world. In PKS, the planner’s knowledge state is represented by five databases, each of which stores a particular type of knowledge. Actions are described by the changes they make to the database set and, thus, to the planner’s knowledge state. PKS also supports ADL-style conditional action effects [23].

Using PKS’s representation language, we can formally model the example robot scenario by describing the objects, properties, and actions that make up the planning level domain. As we described above, objects at the planning level are simply labels that denote actual objects in the world identified by the robot/vision system.

High-level domain properties are defined by sets of logical fluents, i.e., predicates and functions that denote particular qualities of the world, robot, and objects. For instance, to model the example object manipulation scenario we include fluents such as:

- open(x): object x is open,
- gripperempty: the robot’s gripper is empty,
- ingripper(x): object x is in the gripper,
- ontable(x): object x is on the table,
- isin(x, y): object x is stacked in object y,
- reachableX(x): object x is reachable using grasp type X, and
- radius(x, y): the radius of object x is y,

among others. While most high-level properties tend to abstract the information returned by a set of sensors at the robot level, some properties correspond more closely to individual sensors (e.g., gripperempty closely models a low-level sensor that detects whether the robot’s gripper can be closed without contact, while ontable requires data from a set of visual sensors concerning object positions).

High-level actions represent counterparts to some of the motor programs available at the robot level. For instance, in the example scenario the planner has access to actions like:

- graspA-stack(x): grasp object x from a stack using grasp type “A”,
- graspA-table(x): grasp x from the table using grasp A,
- putInto-object(x, y): put object x into an object y on the table,
- putAway(x): put x away on the shelf, and
- findout-open(x): determine whether x is open or not,

among others. Some actions like “grasp A” are divided into two actions to account for different object configurations, however, the motor programs that implement these actions do not necessarily make such distinctions. Furthermore, the object-centric nature of the planning actions means that they do not require 3D coordinates, joint angles, or similar real values but, instead, include parameters that can be instantiated with specific objects. Actions also exist for other grasping options (B, C, and D) available at the robot level. Actions like findout-open are high-level sensing actions that direct the robot to gather information about the world state that is not normally provided to the planner as part of its regular state updates.
Preconditions

Here, action. Table 1 shows two PKS actions from the example domain.

An action’s preconditions specify the domain properties that must hold for an action to be applied, while an action’s effects encode the

planner know

φ

doing sensing actions that return binary information. An expression like

K

−
K

add
K

(1)

In this plan, obj1 is grasped from the table and put it into obj2, before

the stacked objects are grasped and removed to the shelf.

The planner can also build more complex plans by including sensing actions. For instance, if the planner is given the goal of removing the “open” objects from the table, but does not know whether obj1 is open or not, then it can construct the conditional plan:

This plan senses the truth value of open(obj1) and reasons about the

possible outcomes of this action by including branches in the plan: if open(obj1) is true (the K+ branch) then obj1 is grasped and put away; if open(obj1) is false (the K− branch) then no further action is taken.

To execute plans, the planning level interacts with the robot/vision

system. Actions are fed to the robot one at a time, where they are

converted into motor programs and executed in the world. A stream of ISTFs is also generated, arising from the motor programs being

executed. Upon action completion the robot/vision level informs the

planner as to any world state changes (the final state of the last ISTF).

An essential component in this architecture is the plan execution

monitor, which assesses action failure and unexpected state informa-

tion to control replanning and resensing activities. In particular, the

difference between predicted and actual state information is used to
decide between (i) continuing the execution of an existing plan, (ii)

Table 1. Examples of PKS actions in the object manipulation domain

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>graspA-table(x)</td>
<td>K(clear(x))</td>
<td>add(K_f, ingripped(x))</td>
</tr>
<tr>
<td></td>
<td>K(gripperempty)</td>
<td>add(K_f, ~gripperempty)</td>
</tr>
<tr>
<td></td>
<td>K(ontable(x))</td>
<td>add(K_f, ~ontable(x))</td>
</tr>
<tr>
<td></td>
<td>K(radius(x) ≥ minA)</td>
<td>add(K_f, radius(x) ≤ maxA)</td>
</tr>
<tr>
<td>findout-open(x)</td>
<td>~K_c(open(x))</td>
<td>add(K_c, open(x))</td>
</tr>
</tbody>
</table>

Actions in PKS are described by their preconditions and effects. An action’s preconditions specify the domain properties that must hold for an action to be applied, while an action’s effects encode the changes made to the domain properties as a result of executing the action. Table 1 shows two PKS actions from the example domain. Here, K_f refers to a database that models the planner’s knowledge of simple facts, while K_c is a specialized database that stores the results of sensing actions that return binary information. An expression like K(φ) denotes a knowledge-level query that intuitively asks “does the planner know φ to be true?”

Given a goal, initial state description, and action list, the planner can build plans that are executable on the robot platform. We currently provide an interface that allows a human user to specify a high-level goal directly to the planning system. Initially, the planner does not know anything about the state of the world. After the robot/vision system performs its early exploration process and begins to produce ISTFs, an initial state description is generated and supplied to the planner automatically with information about newly discovered objects and their sensed properties, described in terms of the high-level fluents. Since PKS can model an agent’s incomplete knowledge, the predicate and function instances in the initial state are treated as known state information, with all other state information considered to be unknown. We currently assume that the action schema are supplied to the planner as input, as are the mappings from high-level actions to low-level robot motor programs. (In Section 5 we consider how high-level action effects can be learned directly from state information.)

For instance, if we consider the situation in the example domain where two unstacked and open objects obj1 and obj2 are on a table, the planner can construct a simple plan using the above domain encoding to achieve the goal of clearing the table:

\[
\begin{align*}
&\text{[graspD-table(obj1),} \\
&\text{putInto-object(obj1, obj2),} \\
&\text{graspB-table(obj2),} \\
&\text{putAway(obj2)]}. \\
\end{align*}
\]

This plan senses the truth value of open(obj1) and reasons about the possible outcomes of this action by including branches in the plan: if open(obj1) is true (the K+ branch) then obj1 is grasped and put away; if open(obj1) is false (the K− branch) then no further action is taken.

To execute plans, the planning level interacts with the robot/vision system. Actions are fed to the robot one at a time, where they are converted into motor programs and executed in the world. A stream of ISTFs is also generated, arising from the motor programs being executed. Upon action completion the robot/vision level informs the planner as to any world state changes (the final state of the last ISTF).
asking the vision system to resense a portion of a scene at a higher
resolution (in the hope of producing a more detailed state report),
and (iii) replanning from the unexpected state using the current state
report as a new initial planning state. The plan execution monitor
also has the important task of managing the execution of plans with
conditional branches, resulting from the inclusion of sensing actions.

When a high-level sensing action is executed at the robot level, the
results of the sensing are made available to the robot/vision system
in a subsequent ISTF, and passed to the planner as part of a state
update. In our example domain, sensing actions like findout-open
allow the robot to use its lower-level object information to make more
informed decisions as to how such actions should best be executed
(e.g., for findout-open the robot could “poke” an object to determine
its openness). The plan execution monitor uses the returned informa-
tion to decide which branch of a plan it should follow, and feeds the
correct sequence of actions to the lower levels. If such information is
unavailable, resensing or replanning is triggered as above.

5 LEARNING ACTION REPRESENTATIONS

The planner is capable of constructing plans that direct the robot’s ac-
tions, in contrast to the reflex-based exploration of the world that the
robot must initially perform. This shift from undirected to directed
behaviour relies on an action specification that encodes the dynam-
ics of the world in which the robot operates. While we have described
how the robot/vision system is capable of generating ISTFs, the state
information encoded in such fragments contains information that is
both relevant and irrelevant to an action specification. The domain
information required for planning actions, however, is more like the
information found in a set of OACs, i.e., a generalization of the in-
formation in a set of ISTFs. Thus, presented with enough examples
of such state transitions, a learning procedure should be able to filter
out the irrelevant information and identify the necessary state infor-
amation required for OACs and planning operators.

Using machine learning techniques to learn action specifications
is not a new idea, and prior approaches have addressed this problem
using a variety of techniques. For instance, inductive learning [32]
and directed experimentation [8] have been applied to data repre-
sented in first-order logic, without noise or non-determinism. Other
approaches have used schema learning to learn probabilistic action
rules operating on discrete-valued sensor data [9]. Also, k-means
clustering of equivalence classes, followed by extraction of sensor
data features, has been used to train support vector machines (SVMs)
to predict deterministic action effects in a given context [4], [18] pro-
poses a method of modelling actions by learning control laws that
change individual perceptual features of the robot’s world. Recently,
attention has shifted to methods which exploit relational structure in
order to improve speed and generalisation performance. [22] gener-
ates and refines rules using heuristic search, and shows that relational
deictic rules are learnt more effectively than propositional or purely
relational rules. [30] uses a logical inference algorithm to efficiently
learn rules in relational environments.

Our approach is based on a connectionist machine learning model,
namely \textit{kernel perceptron learning} [2, 6]. This approach is particu-
larly useful for our task since it can be shown to provide good per-
formance, both in terms of training time and the quality of the models
that are learnt, making it an attractive choice for practical systems.

Learning the complete dynamics of a planning domain requires
the ability to learn both action preconditions and effects. Currently,
our learning mechanism only addresses the problem of learning ac-
tion effects, and the action schema and preconditions are supplied as
input. Since an action’s effects determine the changes made to a state
during execution, the problem reduces to learning particular mapp-
gings between states. Furthermore, our current mechanism can only
learn standard STRIPS and ADL action effects, and is restricted to
relational state properties (i.e., no sensing actions or functions).

The input to the learning mechanism uses a vector representation
that encodes a description of the action being performed and the state
at which the action is applied. For each available action the vector in-
cludes an element that is set to 1 if the action is to be performed, or
0 otherwise. For states, we consider object-independent and object-
dependent properties separately. In the case of object-independent
properties (e.g., gripperempty), the vector includes an element for
each property, representing its truth value at the state being consid-
ered (1 = true, 0 = false). For object-dependent properties we con-
sider each property on a per object basis, and represent only those
properties of the objects directly involved in the action being applied,
and the objects related in some way to those objects. A form of deict-
ic representation is used (similar to [22]), where objects are specified
in terms of their roles in the action, or their roles in a property. Instead
of maintaining a “slot” in the input vector for each possible role, roles
are allowed to overlap. Thus, each object is represented by a set of
inputs, one for each object-specific predicate (e.g., ingripper), and
each relation with another object (e.g., isni). To bind relations to the
correct objects, extra predicates are used isni-obj1, isni-obj2, etc.).
This representation significantly reduces the number of inputs since
its size is dependent on the actions and relations between objects,
rather than the absolute number of objects in the world.

Overall, the input vector has the form: (actions, object-
independent properties, object slot 1 predicates, object slot 2 predi-
cates, . . . , object slot n predicates). Figure 3 shows one such input
vector for an action-state pair. In this case, the action performed is
graspA-stack. The “grasped object” properties are represented in the
object obj1 slot, while the “object below the grasped object” prop-
erties are represented in the object obj2 slot. Here, gripperempty,
clear(obj1), isni(obj1, obj2) and ontable(obj2) are true in the state,
since the corresponding bits are set to 1; all other bits are set to 0.

The output of the learning mechanism is a prediction of the prop-
erties that will change when the action is performed. The output is
also encoded as a binary vector, with each bit representing one prop-
erty of the state: the output value is 1 if the property changes and
0 if it does not. As with the input vector, object-independent properties

<table>
<thead>
<tr>
<th>Input vector</th>
<th>Corresponding action/property</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>graspA-table(obj1)</td>
</tr>
<tr>
<td>1</td>
<td>graspA-stack(obj1)</td>
</tr>
<tr>
<td>0</td>
<td>graspB-table(obj1)</td>
</tr>
<tr>
<td>0</td>
<td>graspC-table(obj1)</td>
</tr>
<tr>
<td>0</td>
<td>putInto-object(obj1, obj2)</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>gripperempty</td>
</tr>
<tr>
<td>0</td>
<td>ontable</td>
</tr>
<tr>
<td>1</td>
<td>clear</td>
</tr>
<tr>
<td>0</td>
<td>isni-obj1</td>
</tr>
<tr>
<td>0</td>
<td>isni-obj2</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. A binary input vector to the learning mechanism
are represented by single elements, and object-specific properties are represented on a per-object basis in slots. Overall, the output vector has the form: \( \text{object-independent properties, object slot 1 predicates, object slot 2 predicates, \ldots, object slot n predicates} \).

Using the above representation, the learning mechanism is tasked with finding the association between action-precondition pairs and their effects, i.e., rules of the form \((A, \text{Pre}_{A}) \rightarrow \text{Eff}_i\). Currently, we have focused on learning the effects of standard STRIPS and ADL planning actions. Thus, all action effects involve either conjunctions of predicates (in the case of STRIPS) or conjunctions of predicates conditioned on other conjunctions of predicates (in the case of ADL).

As a result, it is sufficient to learn a rule for each effect predicate separately and we can treat the learning problem as a set of binary classification problems, one for each (conditional) effect.

A classifier that is both simple and fast is the perceptron [28]. The perceptron maintains a weight vector \( \mathbf{w} \) which is adjusted at each training step. The \( i \)-th training vector \( \mathbf{x}_i \in \{0, 1\}^n \) is classified by the perceptron using the decision function \( f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x}) \). If \( f(\mathbf{x}) \) is not the correct class then \( \mathbf{w} \) is set to \( \mathbf{w} + \eta \mathbf{x} \); if \( f(\mathbf{x}) \) is correct then \( \mathbf{w} \) is left unchanged. Provided the data is linearly separable, the perceptron algorithm is guaranteed to converge on a solution in a finite number of steps [21, 17]. Otherwise, the algorithm oscillates, changing \( \mathbf{w} \) at each misclassified input vector.

Since the problem of learning action effects is not linearly separable in general, we adapt the perceptron algorithm by applying the kernel trick [6]. By doing so, we implicitly map the input feature space into a higher-dimensional space where the data is linearly separable. Since the mapping is implicit, we avoid a massive expansion in the number of features, which may make the problem computationally inaccessible. The kernel trick is applied by rewriting the decision function in terms of the dot product of the input vectors:

\[
f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x}) = \text{sgn}(\sum_{j=1}^{n} a_j \phi_j(\mathbf{x}_j), \mathbf{x}_i),
\]

where \( a_j \) is the number of times the \( j \)-th example has been misclassified by the perceptron. By replacing the dot product with a kernel function \( k(\mathbf{x}, \mathbf{x}_i) \) which calculates \( \phi(\mathbf{x}) \cdot \phi(\mathbf{x}_i) \) for some mapping \( \phi \), the perceptron algorithm can be run in higher dimensional space without requiring the mapping to be explicitly calculated. An ideal kernel is one which allows the perceptron algorithm to run over the feature space of all conjunctions of features in the original input space, allowing an accurate representation of the exact conjunction of features (action and preconditions) corresponding to a particular effect. In our case, the kernel \( k(x, y) = 2^{\text{same}(x,y)} \) is used, where \( \text{same}(x, y) \) is the number of bits with the same value in both \( x \) and \( y \) [29, 10]. (See [19] for a more detailed discussion of this approach.)

### 6 Integration and Empirical Results

In this section we consider two separate interactions between the components described above, forming part of the larger system we are in the process of implementing (see Figure 4). In Section 6.1, we consider the link between the planning level and the robot/vision level, and the execution of high-level plans on the robot platform. In Section 6.2 we focus on the learning mechanism and the actions that arise from the object manipulation scenario. Certain aspects of our system, such as the plan execution monitor and the inclusion of the learning mechanism within the larger system, are currently under development and have not yet been fully implemented. The robot/vision system forming the basis of our implementation consists of an industrial 6 degrees of freedom robot with a two finger grasper, a high resolution stereo camera system, and a haptic torque sensor mounted between the robot and grasper, providing the measurement of forces at the wrist.

#### 6.1 Linking high-level plan generation with robot/vision-level plan execution

From an integration point of view, the robot/vision system is currently linked directly to the planning level and we are experimenting with plan generation and execution. Since the planner is not able to handle raw sensor data as a state description, the low-level ISTFs generated by the robot/vision system must be abstracted into a language that is understandable by the planner. As a result, sensor data is “wrapped” and reported to the planner in the form of “symbolic” ISTFs with state representations that include predicates and functions. Since our present focus is on object and action learning, we have simply hard-coded the mappings between certain sensor combinations and the corresponding high-level properties.

For instance, some of the predicates used in the example manipulation domain are computed as follows:

- **inigrasper, gripperempty**: Initially the gripper is empty and the predicate *gripperempty* is formed. As soon as the robot grasps an object, and confirms that the grasp is successful by means of the gripper not closing up to mechanical limits, the system knows that it has the object in its hand and can form a predicate *inigrasper*\((\text{objX})\), using its visual information about discovered objects to identify the label \(\text{objX}\) corresponding to the object in the gripper. A negated predicate *–gripperempty* is also generated, as are negated *inigrasper* instances for objects not in the gripper. Releasing the object returns the gripper to an empty state again.

- **reachableX**: Based on the position of a circle forming the top of a cylindrical object in the scene, as returned by the circle detection algorithm, we can compute possible grasp positions (for the different grasp types) for each object. Using standard robotics path planning methods we can then compute if there is a collision-free path between the start position and the pose the gripper needs to reach the object for a particular grasp.

- **isin, clear, instack**: These three predicates are computed based on geometric reasoning. Since the object height is not known we can only use the \(x, y\)-plane information. Furthermore the fact that objects with a bigger radius are lower in the stack is assumed. Objects whose centres (in the \(x, y\)-plane) are closer than 40mm are selected as stack candidates. The sorted stack candidates can then be checked for real inclusion using the circle centres and radii.

---

![Figure 4](image-url)
• open: We do not assume that all objects in the world are “open.” Unlike the previous properties which can be determined directly from ordinary sensor data, the robot must first perform an explicit test in order to determine an object’s openness. In this case, the robot gripper is used to “poke” inside the potential opening of an object. If the robot encounters a collision where forces acting upon the gripper are above a certain threshold, then the object is assumed to be closed. Otherwise, we assume the object is open. (We also envision a second, purely visual test for openness using dense stereo, but this approach has not yet been implemented.)

After an initial exploration of its environment, the robot/vision system provides the planner with a report of the current set of objects it believes to be in the world, along with a (possibly partial) state report of the sensed properties of those objects. Using this information as its initial knowledge state, and the high-level action specification described in Section 4, the planner attempts to construct a plan to achieve the goal of clearing a set of objects from the table.

Once a plan has been generated, it is passed to the plan execution monitor which sends actions to the robot/vision level one step at a time. At the robot level, a high-level action is decomposed into a set of motor programs which are then executed by the robot in the world. Currently, the mapping of actions to motor programs is pre-programmed and supplied as part of the input to the system. During the execution of low-level motor programs, a stream of ISTFs is generated and recorded by the robot/vision system. After an action has been executed its success or failure is reported back to the plan execution monitor, along with a new report on the state of the world (the final state of the last ISTF). In our current implementation, the plan execution monitor simply terminates the execution of a plan if it encounters an unexpected state property, or a reported failure of an action. Otherwise, it sends the next action to the robot for execution. (No replanning or focused resensing is performed.) For instance, Figure 5 shows the robot executing the four-step plan described in (1) of Section 4 for clearing the table. (The “shelf” in this case is a special area at the side of the table.)

When a conditional plan with sensing actions is executed, the plan execution monitor sends findout-open actions to the robot/vision level like any other action. At the robot level, such an action is executed as the specific “poke” test described above to determine an object’s openness. The results of this test are returned to the plan execution monitor as part of the updated state report. The monitor then uses this information to determine which branch of the conditional plan it should follow. From the point of view of the robot, it will only receive a sequential stream of actions and will be unaware of the conditional nature of the plan being executed. Figure 6 shows the robot testing the openness of two objects after receiving a sensing action from the planning level. In (a), the test fails since the object is not open; in (b) the test succeeds and the object is assumed to be open.

6.2 Learning STRIPS and ADL action effects in the object manipulation domain

Separate from the above robot/vision-planner integration, we established a preliminary link between the action effect learning mechanism and the planner. In particular, we applied our learning procedure to learn the effects of STRIPS and ADL planning actions, using data simulated from the example object manipulation domain.

The learning mechanism was evaluated using data similar to the ISTFs the robot/vision system is capable of producing. Both STRIPS and ADL versions of the high-level actions were considered. (For example, the two actions graspA-stack and graspA-table described in Section 4 were merged into a single ADL action, along with other changes.) Sensing actions and references to functional fluents were ignored. Two data sets were constructed to train and test the learning mechanism. Individual input vector instances were generated by randomly selecting an action, and setting the inputs for the preconditions required for the action to 1. The action input was set to 1, and all other action inputs to 0. The remaining input bits were used to create the two separate data sets. For the training data, half of the inputs in each instance were randomly set to 0 or 1, with the other half all set to 0 (vice versa for the testing data). Outputs were set to 1 if a state property changed as a result of the action and 0 if not. Thus, the data used to train the learning mechanism incorporated the (strong) assumptions that (i) all the necessary precondition information for an associated action was included as part of an input vector, and (ii) no spurious state changes was represented as part of an output vector. Noise was introduced in the irrelevant bits of the input vector, however, only relevant changes were included in the corresponding output vector.

The learning mechanism was evaluated over multiple test runs using 3000 training and 500 testing examples. To determine an error bound on our results, 10 runs with different randomly generated training and testing sets were used. (All testing was done on a 2.4 GHz quad-core system with 6 Gb of RAM. All times were measured for Matlab 7.2.0.294.) The results of our testing are shown in Fig-
Figure 7. Results from experiments in the object manipulation domain (from [19]). In (a), the error rate for the learnt STRIPS actions is shown, while (b) shows the error rate for the ADL actions. In both cases, the average error dropped to less than 3% after 700 training examples. The standard perceptron error rate, included for comparison, shows significantly worse performance: over 5% error after 3000 training examples. In (c), the training time for both STRIPS and ADL actions is shown (for 1 bit of the effect vector), while (d) shows the prediction time (for 1 bit of 1 prediction). In practical terms, the learning mechanism was quite efficient, requiring 0.035 seconds to train the system on 3000 examples and 1.84 × 10⁻³ seconds to test the system per output. For our particular example domain, there was little difference between the training and prediction times of STRIPS actions, compared with those for ADL actions. (In general we expect performance on ADL domains will always take longer than STRIPS domains, particularly when the conditional effect training examples are very dissimilar to the other training examples available.) Overall, the learning mechanism was able to effectively abstract away the irrelevant information from the ISTFs to produce a high-quality model of the action effects suitable for planning (at least for our current example domain).

7 DISCUSSION

From a representational point of view we have argued that ISTFs and OACs, grounded from actions performed at the robot level, can be viewed as the representational unit that underlies higher-level representations of objects, properties, and actions (“representation through integration”). As the low-level robot/vision system explores the world, successful actions produce ISTFs; on the basis of multiple experiences of particular ISTFs, OACs and high-level action models can be learned. Although some aspects of our approach are currently hard-coded (e.g., the action/motor program mappings), our learning mechanisms are nevertheless able to abstract away from “irrelevant” state information in the ISTFs to learn certain high-level OAC relationships from the robot’s interactions with the world.

The resulting representations also enable interesting interactions between the components of the system (“integration through representation”). For instance, the planner can ignore some details about the execution of actions at the robot level (e.g., sensing actions like `findout-open`) and can avoid making certain commitments that are better left to the robot level (e.g., planning-level grasping actions are unaware of low-level properties like object location, gripper orientation relative to an object, etc.). Thus, we do not try to control all
aspects of robot behaviour at the planning level, but apply the planner’s strengths to problems it can more readily solve. (For instance, PKS does not perform path planning but is more proficient at planning information-gathering operations.) As future work we are extending these ideas, for instance to allow the robot/vision system to choose between a set of possible tests to perform when executing a *tindout-open* sensing action, while leaving the planning-level action specification unchanged.

We have focused on two particular learning problems in this work: object learning and action effect learning. As a result, we have avoided addressing other learning problems (e.g., learning the low-level sensor combinations that lead to particular high-level properties, or the mapping of high-level actions to low-level motor programs), which we leave to future work. Our focus on an implementation "from the world level to the knowledge level," however, provides us with a suitable testing framework for investigating such learning challenges as well as new planning contexts. Moreover, we are also interested in using this platform to explore other high-level learning tasks such as language acquisition.

We must also improve the scalability of our approach and overcome certain assumptions that are not realistic in real-world robotic systems. For instance, the learning mechanism has mainly been tested using state descriptions that are more “complete” than the ISTFs the robot/vision system is likely to produce. One way we can adapt our approach is by using a noise-tolerant variant of the perceptron algorithm, such as adding a margin term [11]. We also believe these techniques can be applied to irrelevant output data (i.e., irrelevant state changes in the action effects), since such changes behave like noise. Additional work is needed to extend our approach to more complex action representations, notably sensing actions and functions. We also believe our approach can be extended to learn action preconditions, provided it is possible to only represent a small number of objects in the state at a time. An attentional mechanism of some sort may be of help in this task [14]. Finally, although we have tested our learning mechanism on simulated data from the same domain used for the robot/vision-planner experiments, we are also aiming to test our learning mechanism with online data generated directly from the robot/vision system. Additional work is also needed to complete the remaining components of our system, most notably the plan execution monitor.

Our approach for integrating a robot/vision system with a high-level planner and action learning mechanism combines ideas from robot vision, symbolic knowledge representation and planning, and connectionist machine learning. The current state of our work highlights some significant interactions between the specific components of our system, however, we believe our approach is much more general and can be applied to other robot platforms and planners. (For instance, we have recently begun work to test some of our components and specifications on a humanoid robot platform.) The components we describe in this paper form part of a larger project called PACO-PLUS4 investigating perception, action, and cognition—combining robot platforms with high-level representation and reasoning based on formal models of knowledge and action [12].

ACKNOWLEDGEMENTS

The work in this paper was partially funded by the European Commission through the PACO-PLUS project (FP6-2004-IST-4-27657).

REFERENCES


---

4 See www.paco-plus.org for more information about this project.


A Seamless Navigation Aid for the Blind
Georgios Sakellariou\textsuperscript{1} and Aristotelis Zosakis\textsuperscript{2}

Abstract. This paper presents a platform for a seamless indoor/outdoor navigation system aimed at aiding blind people. The main goal is for navigation to be achieved with as little a-priori knowledge of the space as possible. This approach is driven by the fact that the performance of common navigation aids depends greatly on the precision of positioning techniques and a-priori mapping. A cognitive robotics inspired method is proposed to tackle this issue. A cognitive map model of the space is proposed, which enables dynamic map building as the person navigates inside the space. Lastly, an integrated indoor/outdoor positioning mechanism guarantees seamless operation in both environments.

1 INTRODUCTION

Blindness and ways of dealing with the increasing problem of visual loss are very important public health issues, with 45 million blind people in the world today. Every day, one hundred people in the UK start to lose their sight. What is more, the number of blind people is expected to double over the next 25 years, partially due to population ageing and diabetes. In addition, the 10 million visually impaired people in the US showcase the severity of this social problem. Visual impairment inhibits everyday lives and affects employment status, justifying the great effort that is being put into estimating the degree and prevalence rate of vision loss in the population. Given that the number of young and potentially more mobile blind or partially blind people is increasing (Figure 1), aids for the visually impaired are needed more than ever. Meanwhile, the high cost of breeding and training guide dogs (the Guide Dogs charity needs 50 million pounds a year to run) creates the need for a cheaper solution.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig1.png}
\caption{Number of people registered as blind (left) or partially sighted (right) in England 1997-2006 by age band, UK National Statistics 2006}
\end{figure}

\section{AN INTELLIGENT AID}

Even though various attempts to attack the problem of indoor navigation aids have been made, a useful device able to function in diverse environments with no external interference remains to be seen. Many aids for blind people focus on outdoor navigation tools, partly due to the wide availability of GPS today. Some of these systems, like EasyContact by Il Village, have gone operational or have been sponsored by local authorities in order to improve the quality of life of blind people. Others, like StreetTalk, Sendero GPS and Trekker, aim to provide blind-friendly interfaces to GPS navigation devices.

Indoor navigation aids are far more sparse, due to the inherent challenges of navigating inside an unknown cluttered scene. Most indoor navigation systems [16], [5], [19] rely on some knowledge of the building prior to navigation, or some fixed beacons transmitting data to aid positioning of the person in the building. Both of these solutions could prove to be a handicap to the widespread usage of such a system, since they would both require some investment on the part of the building owner. Moreover, they primarily focus on 2-D exploration [12]. Under the light of these considerations, the capability of a system to aid a blind person in any indoor scene could be a paramount factor towards its success.

\subsection{A Holistic Approach}

The high level goals of the system aim to tackle the problem of blind navigation aiding as a whole, without falling into the pitfalls of incomplete systems of limited utility. Firstly, we aim to compensate for visual impairment by attempting to reproduce visual cognition. That is enhance the experience of the visually impaired individual by augmenting his sense of the environment. Second, we propose a system that enables dynamic scene perception and mapping with as little a-priori knowledge as possible. In this dynamic map, a wayfinding function also has to be incorporated. Last, we make every effort to provide a friendly and meaningful user interface. This is of prime importance, since the sensory inputs a blind person can provide are limited. In order to accomplish the aforementioned requirements, we decided on a set of features that would be indispensable to the design of the system. Therefore, the major methodological objectives have been identified as follows:

- Develop a navigation aid capable of helping a blind person reach a target located anywhere seamlessly - be it indoors or outdoors
- People, objects and obstacles should not impair the performance of the system
- Use information gathered from previously explored areas to build a reusable map
- Avoid short-range collisions

These objectives are reflected in our approach to the design (Section 3) and architecture (Section 4) of the system.
COUPLING PERCEPTION & ACTION

The methodological framework that is being proposed is based on modern cognitive vision [4], with perception and action being tightly coupled to scene description (Figure 2).

The benefits of such a methodology are twofold when exploring an unknown or partially known environment. First, the user can be informed on courses of action by using this incomplete information that is available at any given point in time. Second, environment perception is closely tied to scene description, thereby dynamically augmenting the available information for future use.

### 3.1 Environment Perception

Conceptual Spaces [3] provide a geometric representation of the perceived space. The framework lies at the crossroads between neural and symbolic computation, and is capable of accommodating a multitude of sensory inputs. Conceptual Spaces have already been used in robot vision systems [1], [17]. In essence, conceptual spaces are n-dimensional metric spaces whose axes represent qualities. Concepts are then defined as particular regions in these spaces. The advantage of conceptual spaces when perceiving and characterising a scene [6] lies in the fact that object categories, or concepts, do not have to maintain absolute relationships with each other unless they appear in the same conceptual space. This is of primary importance when exploring unknown scenes. In that respect, conceptual spaces solve the ontology problem by making it local. For example, if a user is looking for a red door, and he/she has seen one in another building, the system can start looking for similar concepts without having to first define the concept red or door. Adequate sensors need to be specified in order to achieve the desired level of visual perception:

- **Colour stereo camera** - Colour will be of paramount importance to a system aimed at aiding the blind, since colour is an attribute that makes its absence felt the most by the visually impaired.
- **Microwrap Impulse Radar** - Microwrap Impulse Radars (MIR) provide adequate range finding and motion detection functions. They are low-power, low-cost, highly portable and can detect movements in the close vicinity of the sensor.
- **Proximity sensors** - Proximity sensors will act as a failsafe device in case other sensors fail to detect obstacles.

### 3.2 Linguistic Representation

As the system tries to essentially become the eye of the user, a common ground for communication and representation of the visual input is to be determined. Language is the default representation model for the world we observe and much more. Since blindness does not allow for visual grounding between what ones says and what one means, the blind apply other senses and identify associations in order to connect words to physical objects or environments. Recognition of objects can be achieved through the perception of their qualities (e.g. chair: has back & legs). Such qualities give evidence of the object’s identity and are relative to the ones represented geometrically by the conceptual spaces framework. Some of those qualities are perceived solely through vision, e.g. colour, while others are not. An alternative approach [20] would be to associate concepts with their affordance properties, whereby the object’s utility is the decisive factor in these associations.

We consider that our system already has a minimal body of knowledge regarding geometric regions (concepts) connected with words. This can be thought of as a number of associations between words and images. But this level of knowledge is not adequate. The human-machine interaction is based on constant references to words or names for objects and areas. A name itself cannot unveil any quality or property of the signified, it is more or less arbitrary. For example, if the user requests to be led to the couch, and couch is an unknown word to the machine, it cannot proceed with the navigation process, even in the case it has visual contact. This is merely another example of the symbol grounding problem, a frequently discussed one in AI. Furthermore, it has no hint whatsoever of what to search for. In order to make our system capable of dealing with new user queries, we must provide it with a structure that helps it obtain semantic information for each word. A semantic hierarchy of words can become the necessary resource, as in [21], where an OWL ontology for indoor environments is constructed and links to instances of the observed objects are added to it. For our approach, however, the semantic knowledge is important only for the communication and it does not interfere in the analysis and comparison of the perceived data. WordNet [2] is a freely distributed hierarchy of word senses for the English language that characterises different kinds of relations between them, such as is-a-kind-of, is-a-part-of relations for nouns and is-a-way-to for verbs. Using only the noun hierarchy is sufficient for our application, since mostly nouns are used to refer to possible destinations. Compound terms like swimming pool or living room are also included. If for two noun concepts X, Y exists a relation X is-a-kind-of Y, we call Y a hyponym of X and X a hypernym of Y, e.g. saloon is a hyponym of room. The terms holonym and meronym are used respectively for is-a-part-of relations.

The benefit from incorporating WordNet is based on the assumption that related or neighbouring concepts tend to have similar qualities. Therefore, the system can draw some clues as for what is requested, in case an unknown word is uttered, just by browsing some similar concepts described by known words. Even a single known word can be proven helpful. Let’s go back to the previous example. We suppose the word couch is unknown. In WordNet, as a noun, it has 3 senses:

1. sofa, couch, lounge – an upholstered seat for more than one person
2. couch – a flat coat of paint or varnish used by artists as a primer
3. couch – a narrow bed on which a patient lies during psychiatric or psychoanalytic treatment

Each sense is linked to a set of synonyms or synset (e.g. {sofa, couch, lounge} for the first sense) and is accompanied by a short definition and/or example known as gloss. In the navigation aid’s physical context, the first sense or ‘couch (1)’ is the one most often encountered. However, the following process can be repeated for each sense. A small set of related word senses can be extracted from the hierarchy and is depicted in Figure 3, where broader senses are set higher and are linked to their hyponyms or children in the tree.
Concepts like seat\(^3\) or chair contain the general notion and entail the practical use of a couch. Let’s suppose that the concept chair is known, and represented as a region in the \(n\)-dimensional space. The system is now much more aware of the requested target. It must locate an object similar to a chair but not a chair. Such directives can be modeled geometrically. If visual contact of a couch is already made, a vector of its qualities is available. All that is missing is a name for it. Relation of concepts in the hierarchy may possibly imply relation of properties and hence geometrical proximity. An unnamed concept that neighbours the region of chair can be one major candidate offer to the user when he requests for couch. He, in turn, can assert the system’s offer. In worse cases, visual data for the target concept are unavailable and relations cannot be justified. The system must, then, consider the closest matching objects it has already perceived or extract a further level of related senses from the WordNet hierarchy in search of other known ones.

Another source of information is the use of more statistics-based metrics, like the WordNet similarity measures. Working reversely from above, using these measures we can assign a similarity score between each known word sense and the new word. The concept vectors that approach the region of the most similar word, or words, can help identify the target. When the extraction of related senses from the hierarchy cannot yield any known words, similarity scores can offer the solution by evaluating what we already know. WordNet similarity measures are often used in Natural Language Processing problems, like word sense disambiguation [13], [18]. There is a wide selection of suggested measures, from strictly hierarchy-based, like Leacock and Chodorow’s [10], to others using the information content of concepts based on occurrence frequencies in corpora, like Lin’s [11] or Jiang and Conrath’s [7]. In [14] a new measure was introduced that represents each sense as a context vector of the words of its gloss that is called a gloss vector. It computes the similarity of a sense pair as the cosine of the angle of the two corresponding gloss vectors.

\(^3\) in Figure 3 seat appears twice but with slightly different senses, the gloss for seat (3) is: furniture that is designed for sitting on and for seat (4): any support where you can sit (part of a chair).

### 4 SYSTEM ARCHITECTURE

An approach incorporating ideas from mobile robot navigation is a great contribution to indoor navigation aids for the blind. Robotics research has been preoccupied with navigation in unknown scenes for decades. An analogy between a blind person and an autonomous robot can be utilised in order to put these ideas into use. An overview of how these techniques could be integrated in a navigation aid is shown in Figure 4.

The architecture can be split into two major components, the perception/action component and the scene description component (Section 3). Inputs to the perception/action component will be divided between four modules, each performing a different task. One to perceive the environment, one to listen to input from the user, and two modules for indoor and outdoor positioning respectively. It outputs data back to the user and/or the scene description component. The scene description component is responsible for dynamically building a map of the scene, taking into account information coming from the perception/action component, as well as from a map database of already explored scenes, or a-priori knowledge of current scenes.

What makes this aid intelligent is the interaction between the different components of the system, the user and the environment. Such interactions can take the form of expectation when exploring a newly encountered scene.

### 4.1 Indoor/Outdoor Positioning

In order to determine the current position of the user, a range of positioning and navigation technologies have to be employed:

- **GPS** - The predominant source for outdoor positioning information.
- **WLAN** - WLAN can be used for indoor positioning.
- **Dead reckoning** - The tried and tested dead reckoning method is indispensable to a system that aims to be as failsafe as possible. Both GPS and WLAN have variable performance depending on the environment they operate in.
4.2 Input/Output Interface

An adequate input/output interface is of primary importance when the intended user is a blind person. The gap arises when the user can connect words to objects even without seeing them, while the system cannot connect a name to an object that has not yet been learned. This, however, is a characteristic we can benefit from, if the system is adaptive to user corrections and supervision, through dialogue. To maximise ease of use, a voice recognition mechanism could prove to be the most appropriate mode of interaction. Off-the-shelf solutions exist, like IBM’s ViaVoice, that can be used with the assistance of a portable PC. Queries regarding desired destinations or the current position can be passed on using such methods involving speech recognition. Common functions can be hardwired to buttons on a custom keyboard. For example, yes/no keys with Braille labels can facilitate human responses to the system’s queries. The limitations of natural language processing can be bypassed if a more strict definition of a controlled language and human-machine interaction scenarios are applied. In this manner, ambiguities and complex but imperfect methodologies are avoided. Let’s separate the commands or queries to and from the system into three distinct categories:

- **Destination finding queries**: these constitute the basic input of the navigation system from the user’s point of view. A destination finding query can be examined in two levels of analysis. The inner target object or area and the grammatically formatted request that refers to it. The length of spoken requests and their lack of structure can often entail ambiguities, pauses, repetitions and of course errors. Hence, in these cases general speech recognition applications can face serious problems and even fail, not to mention the detailed structure of the grammars needed to analyse such naturally spoken sentences. For these reasons, we support controlled dialogue-based destination finding queries. VoiceXML is the state-of-the-art interactive voice dialogue handling standard. A transition from traditional Interactive Voice Response (IVR) platforms to ones enabled with the VoiceXML technology is gaining ground in a variety of businesses providing e-commerce, banking, entertainment or directory services, such as AT&T, Lucent and Motorola. Web content, even entire hypertext documents, can be exploited at a low cost and high user friendliness with limited requirements in hardware (e.g. a mobile phone) from the user’s side. Such a web browsing application via audio is HearSay [15]. It partitions and labels HTML pages based on their structure and semantic content and creates VoiceXML dialogues for fast and flexible hands-free navigation. VoiceXML can combine speech synthesis and recognition, Dual-Tone-Multiple-Frequency (DTMF) key input and prerecorded audio. Queries, user menus or web-like forms can be modelled as speech interaction procedures. With small sections of code, like the following, simple interaction processes can be adequately described.

```xml
<form id="destination_query">
  <block>Welcome.</block>
  <field name="destination">
    <prompt>Where would you like to go?</prompt>
    <grammar src="destination.grxml"/>

    <prompt>I will guide you to <value expr="destination.target"/>
  </grammar>
  ...
  </field>
</form>
```

VoiceXML documents can link to speech recognition grammars that describe user requests, specifically the words and patterns to be expected for by the speech recogniser. The appropriate W3C standard for this task is the Speech Recognition Grammar Specification (SRGS). An SRGS grammar is presented either in augmented Backus-Naur Form (BNF) or in XML (e.g. a grammar.grxml file), and can be as expressive as a Context-Free Grammar (CFG). The user usually combines imperatives like ‘find’ and ‘reach’ or polite phrases like ‘I would like to go to’ with the name of the target object or area to form the desired request. The VoiceXML interpreter conducts the dialogue, i.e. executes prompts, and with the help of the grammar analyses the user’s responses so that the actual target words are detected and discriminated from the request context. The grammar is in constant accordance with the acquired knowledge of already traced places. At the implementation level, as soon as a known target is recognised, the system enters wayfinding mode. If the target word is unknown, it attempts to relate it to known concepts in the manner analysed in 3.2.

- **Instruction commands**: once the destination finding query has been submitted and the data regarding the target are processed, both by vision and by database resources, the system is ready to guide the person. With the current position on the map in respect, the system generates simple directions that can be transmitted by a speech generation module. These directions can belong to a closed set of positioning expressions, like: ‘turn left/right’, ‘proceed straight’, ‘stop’.

- **Confirmation requests**: as soon as the system has concluded a guiding session, it can address a request to the user in order to verify its accuracy, such as: ‘Living room approached, please confirm’. Another case in which a confirmation is needed is when previously unknown targets are detected and offered as guesses. Simple yes/no answers by the user are adequate at this level. Further information can be obtained in the case of negative answers. The user can then indicate the correct name for the approached location – linguistic information onwards connected to the visual resources. This is one prominent way to deal with the symbol grounding problem, the machine’s inability to connect the signified with the signer. Even with sight problems affecting the perception of their environment, people can still achieve object identification by exploiting other senses like tactition. Confirmation procedures can also be modeled with VoiceXML and SRGS and newly acquired data for reached targets can be used to update the speech recognition grammar for future destination finding queries.

A number of the most recent or most common destination finding queries can be dynamically stored to the system’s memory. All the necessary steps for their completion can also be maintained, e.g. setting of temporary check points on the map. Finally, a special instant positioning command-button can be available. Using this at any location, the user can be informed about their current position with respect to the constructed map or previous requests. Nearby known objects or landmarks present on the map can also be used to provide further help, as their association with words is now established.

4.3 Dynamic Map Building & Map Database

Most humans know a large variety of points and places, but always seem to use a set of main paths to the required destination. These
main paths could be seen as a skeleton of possible movements in a 2-D space. In a similar fashion, a skeleton of main routes in an indoor environment could be seen as the basis of navigating inside it. The crucial point of having a skeleton available is that the user can easily reach the proximity of the desired destination without navigating blindly in search of the target. Cognitive maps \([9]\), in conjunction with the Spatial Semantic Hierarchy \([8]\), can facilitate dynamic spatial modelling and wayfinding through a set of training-navigation sessions.

The map database serves the purpose of collecting mapping data from the dynamic map building module and makes them available for quick future use via the Data Management & Central Database module (Figure 4). In this fashion, not only are previously explored areas quicker to navigate in the future, but in practice it creates another useful capability. If a blind person is assisted in navigating around a room by a another person, then he/she can do it on their own from that point on. In real life, this is a useful and very meaningful function, as it is equivalent to a person showing you around a room.

5 AUGMENTED PERCEPTION MECHANISMS

The presented framework is capable of dealing with unknown scenes. However, in typical cluttered spaces, limitations originating from the sensory inputs that are realisable in a portable and wearable system are not uncommon. Even though in principle the system can autonomously navigate in any space, some input could be required in order to improve efficiency. This external input can be put into two main categories. First, a local map transmission mechanism can be established. In particularly complex environments, downloading a local cognitive map could increase dramatically the performance of the system, avoiding meaningless exploration until the desired level of knowledge is achieved. Second, the use of local positioning information will enable the system to minimise the effect of positioning errors. Even though GPS positioning is reliable in open spaces, in city landscapes interference from ambient reflections causes inaccuracies that could impair system performance. Likewise, WLAN has varied performance depending on signal strength and distance from the source. Hence, local positioning aiding could prove useful when the scenery contains many obstructions. This can be achieved by local stations transmitting absolute and accurate coordinates that have been pre-determined by other methods.

6 CONCLUSION

In this paper, we presented a framework for a seamless aid for the blind or partially sighted. The framework features several major benefits. First, it enables a blind individual to navigate in unknown environments with the help of dynamic map building. As such, it can use local positioning aids only as an enhancement and does not solely rely on their existence. Second, the architecture caters for symbol grounding that is particular to the user and his/her perception on the environment. Therefore, it does not rely on ontologies that could be confusing to a blind person unfamiliar with the area being navigated. Last, it treats the user interface as an integral part of an effective aid, and does not fall into the pitfalls of ad-hoc solutions for controlled environments.

REFERENCES

Notes on a qualitative theory of shadows

Paulo Santos\(^1\) and Hannah M. Dee\(^2\) and Valquiria Fenelon\(^3\)

Abstract. Recently, cognitive psychologists and others have turned their attention to the formerly neglected study of shadows, and the information they purvey. These studies show that the human perceptual system values information from shadows very highly, particularly in the perception of depth, even to the detriment of other cues. However with a few notable exceptions, computer vision systems have treated shadows not as signal but as noise. This paper makes a step towards redressing this imbalance by considering the formal representation of shadows. We take one particular aspect of reasoning about shadows, developing the idea that shadows carry information about a fragment of the viewpoint of the light source. We start from the observation that the region on which the shadow is cast is occluded by the caster with respect to the light source and build a qualitative theory about shadows using a region-based spatial formalism about occlusion. Using this spatial formalism and a machine vision system we are able to draw simple conclusions about object and ego location for a mobile robot.

1 Introduction

The purpose of this work is to develop a qualitative spatial reasoning framework about cast shadows, or shadows for short. Qualitative spatial reasoning (QSR) aims at the logical formalisation of space from elementary entities such as regions, line segments, directions amongst others [10]. The purpose of this field is to provide clear representations and efficient automated reasoning methods for handling commonsense knowledge about space.

From the formalisation of shadows we aim to provide a computer vision system with inference methods capable of concluding facts about the position, orientation and motion of objects in the world from the visual observation of objects and their shadows. As we show in Section 2, the cognitive (informational) content of shadows is great, and we humans use this in our day to day perception of depth and motion. A shadow is caused when an object (a “caster”) comes between a light source and a surface (a “screen”). Self shading is what occurs when the caster and screen are the same object, and the informational content of such shadows has been investigated at length within the computer vision community (as shape from shading). However, cast shadows (in which the caster and screen are different objects) have usually received less attention in scene understanding. We shall concern ourselves in this paper with cast shadows, restricting the investigation to the more common case in which the caster and screen are largely opaque. The present work takes one particular aspect on reasoning about cast shadows, developing the idea that shadows provide the viewpoint of the light source. In other words, from the viewpoint of the light source the shadow is completely occluded by the caster. From this observation we construct a qualitative theory about shadows upon the spatial theory about occlusion known as the region occlusion calculus (ROC)[29].

Section 2 discusses prior work in shadow reasoning, from within Philosophy, Psychology and Computer Vision to motivate the following sections. The region occlusion calculus (ROC) is discussed in Section 3. Section 4 presents an extension of ROC to deal with shadows (which we call Perceptual Qualitative Relations about Shadows – PQRS), and Section 5 provides examples of inferences within the extended theory. Some conclusions are finally drawn on Section 6.

2 Shadows in the cognitive sciences

The importance of cast shadows in our depth perception was intensely exploited in Renaissance paintings [7, 12]. In particular, it was through investigation of how the 3D world could be depicted in 2D paintings that projective geometry came to be developed in the 15th century. However the cognitive processes behind the perception of the external world using shadow as cues have only recently begun to be investigated [20, 9].

Casati in [5, 6] points out that cast shadows carry information about the presence and location of the light source and the caster when they are inside or outside the observer’s field of view. Shadows also carry information on the intensity of the source, the shape of the caster and the texture of the screen, and it is possible to hypothesise the distance between the caster and the screen given whether or not

\(^1\) psantos@fei.edu.br. Elect. Eng. Dep., FEI, Sao Paulo, Brazil
\(^2\) hannah@comp.leeds.ac.uk. Dep. of Computing, University of Leeds, UK
\(^3\) vfenelon@gmail.com. Elect. Eng. Dep., FEI, Sao Paulo, Brazil

Figure 1. Sometimes shadows carry information about objects outside of view, via the “viewpoint” of the light source. Photo 1(a) shows the artwork Shadow Stone, by Andy Goldsworthy.
the caster and the shadow are in contact with each other. Another important fact about the information content of shadows is that they can be seen as providing the observer with a second viewpoint: that of the light source, as the shadow depicts the projection of the caster’s terminator line. However, as Casati goes on to point out, shadows are far from reliable sources of information. Noise should be taken into account, since shadows of single objects may be perceived as split (due to uneven screens) or misleadingly perceived as connected to their caster (due to occlusion).

In the field of computer vision much shadow detection work is centred around the idea of shadow as noise. When subtracting background from video to find objects of interest, shadows are a major source of false positives, hence shadow detection becomes important for noise reduction (see, e.g., [23]). When we consider systems which use shadows there are only a handful: [4] use known 3D locations and their cast shadows to perform camera calibration and light location (using known casters and screen to tell about light source); [8] uses the moving shadows cast by known vertical objects (flagpoles, the side of buildings) to determine the 3D shape of objects on the ground (using the shadow to tell about the shape of the screen). Perhaps most relevant to the current paper is the work of Balan et al. [1], who use shadows as a source of information for detailed human pose recognition: they show that using a single shadow from a fixed light source can provide a similar disambiguation effect in using additional cameras.

One essential component of many computer vision shadow detection systems is absent when we consider the problem in the context of a mobile robot, and that is a reliable model of background. Within computer vision, shadow modelling is usually performed in the context of a static camera, and therefore takes advantage of the construction of some statistical background model (usually a Mixture of Gaussians) constructed over some tens if not hundreds of frames (e.g. [26, 23]), and sometimes additional hardware e.g. [17] who use two cameras. Finlayson et al. in [14] give impressive results in shadow suppression from single images, but require a high quality calibrated camera, something which is often absent in a robotics setup. Salvador et al. in [31] present a promising technique which also takes into consideration edge detection results in classifying shadows in single images.

The psychological work reported in [18, 22, 25] discusses experimental results which suggest that the human perceptual system prefers cues provided by shadows over other information (such as changes in the object’s size due to motion in depth) in order to infer 3D motion of objects. In one experiment discussed in [18], a number of human subjects were presented with a computer simulation in which the shadow of a static square (cast on a chequered screen) moves away from its caster. A picture of this simulation is shown in Figure 2. Most subjects reported perceiving the square moving towards and away from the background according to the shadow motion, even though the size of the square remained unchanged throughout the experiment (this was clear from the unchanging chequered background). It is worth pointing out that, geometrically, there is a number of possible competing hypotheses for the shadow motion that would be more coherent than object motion as an explanation (e.g. the motion of the light source). Moreover, the subjects even reported having the illusion of the objects changing in size depending on the shadow’s motion. These findings (summarised in [22]) suggest that the human perceptual system is biased to use shadow information on the interpretation of 3D motion and that shadow information can even over-ride notions of conservation of object size. This justifies the development of a shadow processing stage in any cognitively plausible vision system.

![Figure 2. A sequence of snapshots from a graphics simulation showing the shadow of a static square moving away from its caster (adapted from [22] with kind permission of the author).](image)

Apart from providing a strong cue about motion in depth, cast shadows also provide information that could be used in the interpretation of surface shape of the screen, however experimental findings suggest that this information is not used by the human perceptual system [22].

Further psychological evidence shows the importance of shadows to object recognition [3], to the determination of optical contact [25] and on how the perception of shadows affects visual search [30] and reach-and-grasp tasks [2].

In this paper, perception (and reasoning about) cast shadows is understood as the problem of inferring spatial relations from the observation of objects and their shadows. The use of cast shadows in such processes, however, presupposes the solution of the shadow correspondence problem [20, 7], which involves the segmentation of shadows in scenes and the connection of shadows to their relative casters [22]. Shadows, like holes, are dependent objects – without a caster, they do not occur. Matching shadows to their casters is a hard problem for various reasons: there may be various competing possibilities to match shadows and objects in a complex scene (i.e. the shadow correspondence problem is underconstrained); the screen may not be planar which may turn a point-to-point matching into a complex non-linear registration procedure; and shadows of close objects may merge. Psychological experiments suggest that the human visual system utilises an initial coarse matching between shadows and their casters that would allow for a re-evaluation given evidences from high-level reasoning procedures [20]. Casati in [7] presents some evidence, from the observation of Renaissance paintings, that the perceptual system solves the shadow correspondence problem even when the shadows depicted represent a more complicated situation than naturally observed. Given these complications, we incororporated a partial solution to this problem using as heuristics the idea that a shadow connected to an object is the shadow cast by this object. A complete solution to the shadow correspondence problem is outside the scope of this paper. Instead, we concentrate on formalising the information content of shadows, using a region-based ontology from a qualitative spatial reasoning perspective. The next section presents the underlying theories with which shadows are formalised in this work.

3 Background

This section presents the qualitative spatial reasoning approaches that are used in the development of this research. A comprehensive overview of this field can be found in [10].

One of the best known QSR approaches is the Region Connection Calculus (RCC) [27]. RCC is a many-sorted first-order axiomatisation of spatial relations based on a reflexive, symmetric and non-transitive dyadic primitive relation of connectivity (C/2) between
two regions. Informally, assuming two regions \( x \) and \( y \), the relation \( C(x, y) \), read as \( “x \) is connected with \( y” \), is true if and only if the closures of \( x \) and \( y \) have a point in common.

Assuming the \( C/\) relation, and that \( x, y \) and \( z \) are variables for spatial regions, some mereotopological relations can be defined. Some of them are: \( DC(x, y) \), which is read as \( “x \) is disconnected from \( y” \); \( EQ(x, y) \): \( “x \) is equal to \( y” \); \( O(x, y) \): \( “x \) overlaps \( y” \); \( P(x, y) \): \( “x \) is part of \( y” \); \( PO(x, y) \): \( “x \) partially overlaps \( y” \); \( PP(x, y) \): \( “x \) is a proper part of \( y” \); \( EC(x, y) \): \( “x \) is externally connected with \( y” \); \( TPP(x, y) \): \( “x \) is a tangential proper part of \( y” \); \( NTTPP(x, y) \): \( “x \) is a non-tangential proper part of \( y” \); \( TPP/2 \) and \( NTTPP/2 \) are the inverse relations of \( TPP/2 \) and \( NTTPP/2 \) respectively. These relations are formally defined as follows [27].

\[
\begin{align*}
[D1] & \quad DC(x, y) \equiv \neg C(x, y) \\
[D2] & \quad P(x, y) \equiv \forall z [C(z, x) \rightarrow C(z, y)] \\
[D3] & \quad EQ(x, y) \equiv P(x, y) \land P(y, x) \\
[D4] & \quad O(x, y) \equiv \exists z [P(z, x) \land P(z, y)] \\
[D5] & \quad PO(x, y) \equiv O(x, y) \land \neg P(x, y) \land \neg P(y, x) \\
[D6] & \quad EC(x, y) \equiv C(x, y) \land \neg O(x, y) \\
[D7] & \quad PP(x, y) \equiv P(x, y) \land \neg P(y, x) \\
[D8] & \quad TPP(x, y) \equiv PP(x, y) \land \exists z [EC(z, x) \land EC(z, y)] \\
[D9] & \quad NTTPP(x, y) \equiv PP(x, y) \land \neg \exists z [EC(z, x) \land EC(z, y)]
\end{align*}
\]

RCC has been applied in qualitative simulations [11], on expressing shape [16], in qualitative theories of motion [35, 24], as a basis for modeling object-observer phenomena such as occlusion [29, 28, 19], and other domains as discussed in [10].

RCC represents qualitative mereotopological relations between spatial regions independently of any observer’s viewpoint. In contrast [15] proposes a lines-of-sight calculus in order to represent relative positions between pairs of non-overlapping convex bodies as seen from a particular observer. The main interest in this formalism is the representation and manipulation of information about visual occlusion between objects. Inspired by these ideas, Region Occlusion Calculus (ROC) [29] was proposed to represent the various possibilities of interposition relations between two arbitrary shaped objects as an extension of the Region Connection Calculus. The relations constituting ROC are represented in Figure 3. These relations are defined over RCC relations along with the primitive relation \( TotallyOccludes(x, y, v) \), which stands for \( “x \) totally occludes \( y \) with respect to the viewpoint \( v” \).

In order to make explicit both the distinction between a body and the region of space it occupies, and the distinction between a physical body to its projection with respect to a viewpoint, ROC assumes, respectively, the functions \( r \) (region) and \( i \) (image). The region function can be understood as a mapping from a physical body and its occupancy region:

\[ r : \text{body} \rightarrow \text{spatial\_region}. \]

Similarly, the image function is a mapping from a physical body to its relative 2D projection with respect to a viewpoint:

\[ i : \text{body} \rightarrow \text{spatial\_region}. \]

In the remainder of this section we present those ROC axioms (introduced in [29]) that are used in our theory of shadows.

Formula (1), below, is the ROC axiom that states that \( “x \) totally occludes \( y, x \) totally occludes any part of \( y” \):

\[
\forall x y z v \ [\text{TotallyOccludes}(x, y, v) \wedge P(r(z), r(y))] \rightarrow \text{TotallyOccludes}(x, z, v). \quad (1)
\]

The fact \( “x \) totally occludes \( y, no part of \( y \) totally occludes part of \( x” \) is formalised in formula (2).

\[
\forall x y z v \ [\text{TotallyOccludes}(x, y, v) \rightarrow (2)
\]

\[
\forall z u [[P(r(z), r(x)) \wedge P(r(u), r(v))]] \rightarrow \neg \text{TotallyOccludes}(u, z, v)]
\]

In order to simplify notation, the following definitions are included in the theory:

\[
\begin{align*}
[D9] & \quad \text{Occludes}(x, y, v) \equiv \exists z u P(r(z), r(x)) \land P(r(u), r(y)) \\
&D10 \quad \text{PartiallyOccludes}(x, y, v) \equiv \text{Occludes}(x, y, v) \land \neg \text{TotallyOccludes}(z, y, v) \\
&D11 \quad \text{MutuallyOccludes} \equiv \text{Occludes}(x, y, v) \lor \text{Occludes}(y, x, v)
\end{align*}
\]

With these abbreviations the ROC relations (Figure 3) can be defined by the following axiom schemas [29]:

\[
\begin{align*}
\Phi \Psi(x, y, v) & \equiv \Phi(x, y, v) \land \Psi(i(x, v), i(y, v)) \quad (3) \\
\Xi \Psi^{-1}(x, y, v) & \equiv \Xi(y, x, v) \land \Psi(i(x, v), i(y, v)) \quad (4)
\end{align*}
\]

where if:

\[
\begin{align*}
\Phi & \equiv \text{NonOccludes}, \text{then } \Psi \in \{DC, EC\} \\
\Phi & \equiv \text{TotallyOccludes, then } \Psi \in \{EQ, TPP, NTPPP\} \\
\Phi & \equiv \text{PartiallyOccludes, then } \Psi \in \{PO, TPP, NTPPP\} \\
\Phi & \equiv \text{MutuallyOccludes, then } \Psi \in \{PO, EQ, TPP, NTPPP\} \\
\Xi & \equiv \text{TotallyOccludes, then } \Psi \in \{EQ, TPP, NTPPP\} \\
\Xi & \equiv \text{PartiallyOccludes, then } \Psi \in \{PO, TPP, NTPPP\} \\
\Xi & \equiv \text{MutuallyOccludes, then } \Psi \in \{TPP, NTPPP\}
\end{align*}
\]

It is worth noting that, without taking into consideration the MutuallyOccludes relations, if the objects shown in Figure 3 were interpreted as regions on a plane, the ROC predicate representing each relation would collapse to its corresponding RCC relation. I.e., for instance, the relation \( \Phi \Psi(x, y, v) \) (definitions as above) would collapse to \( \Psi(x, y). \) This observation is important to understand the meaning of Axiom (11) below.
4 Perceptual qualitative relations about shadows (PQRS)

For the purposes of this work we assume a static light source, denoted by the constant symbol $L$, situated above the observer$^4$. We also assume that the scenes are observed from an egocentric point of view that is represented by $v$ (in contrast to a global, “birds-eye” perspective). In order to simplify the notation we also assume that shadows are cast on a single screen $Scr$ which does not need to be flat or continuous, since (as we shall see) shadow detection in this work does not take into account the shapes of image regions, but only the intensity of the pixels composing them. The basic part of the theory has a sort for physical bodies (including the casters, the screen and the shadows): $a_1, \ldots, a_n$; sorts for time points: $t_1, \ldots, t_n$; and spatial regions: $r_1, \ldots, r_m$. For convenience we represent shadows by the symbols $s_1, \ldots, s_n$. It is assumed throughout this paper that the variables are universally quantified, unless explicitly stated.

The set of perceptual qualitative relations about shadows (PQRS) includes the region occlusion calculus and a subset of the region occlusion calculus (ROC) comprised of the relations $\text{NonOccludesDC} (o, s)$, $\text{NonOccludesEC} (o, s)$, $\text{PartiallyOccludesPO} (o, s)$, $\text{PartiallyOccludesTPP} (o, s)$, $\text{TotallyOccludesTPPI} (o, s)$, $\text{TotallyOccludesEQ} (o, s)$ and $\text{TotallyOccludesNTTPPI} (o, s)$ for a caster $o$ and its shadow $s$. The remaining ROC relations (shown in Figure 3) have no model with respect to casters and their cast shadows.

We introduce the predicate $\text{Shadow} (s, o, Scr, L)$ that denotes that $s$ is a shadow of object $o$ on the screen $Scr$ from the light source $L$. It is also convenient to define the following sort predicate:

$$I_{s,o,\text{Shadow}} (s) \equiv \exists o \in \text{scr}, l \text{ Shadow} (s, o, \text{scr}, l)$$

We can now state as an axiom that the shadow of an object $o$ is the region in a screen that is totally occluded by the caster from the light source viewpoint. Formally, we have:

$$\text{Shadow} (s, o, Scr, L) \rightarrow \text{PO} (r(s), r(Scr)) \wedge$$

$$\text{TotallyOccludes} (o, s, L) \wedge \neg \exists o \in \text{TotallyOccludes} (o', o, L).$$

The third conjunct of the righthand side of Formula (5) guarantees the existence of the shadow of $o$.

Therefore, it follows from (5) and the ROC axiom (2) that no shadow occludes its own caster$^5$, as denoted by Theorem (T1) below.

$$\text{Shadow} (s, o, Scr, L) \rightarrow \neg \text{TotallyOccludes} (s, o, L).$$

It is also a consequence of Axiom (5) and the ROC axioms the fact that no shadow casts a shadow itself (cf. Theorem (T2)):

$$\text{Shadow} (s, o, Scr, L) \rightarrow \neg \text{Shadow} (s', s, Scr', L).$$

Proof. Let’s assume, reasoning by contraposition, that $\text{Shadow} (s, o, Scr, L)$ and $\text{Shadow} (s', s, Scr, L)$ are true, for an object $o$ and shadows $s'$ and $s$. Thus, we have both $\text{TotallyOccludes} (o, s)$ from Axiom (5) and $\neg \text{TotallyOccludes} (o, s)$ from (T1).

We can also prove that if two shadows of distinct objects partially overlap, then the objects will be in a relation of occlusion with respect to the light source, as expressed in Theorem (T3).

$$(T3) \text{ Shadow} (s, o, Scr, L) \wedge \text{Shadow} (s', o', Scr, L) \wedge \neg o = o' \wedge \text{PO} (s, s') \rightarrow (\text{Occludes} (o, o', L) \vee \text{Occludes} (o', o, L))$$

Proof: From (5) we have that $\text{Shadow} (s, o, Scr, L)$ implies $\text{TotallyOccludes} (o, s)$ which, together with (1), forces

$$\forall o \in \text{PO} (ps, s) \wedge \text{PO} (po, o) \wedge \text{TotallyOccludes} (po, ps)$$

Likewise, $\text{Shadow} (s', o', Scr, L)$ implies $\text{TotallyOccludes} (o', s')$ and $\text{PO} (s, o)$ implies $\exists z [\text{PO} (z, s) \wedge \text{PO} (z, s')].$

From these facts we can conclude that

$$\exists o \in \text{PO} \text{TotallyOccludes} (po, z) \wedge \exists o' \in \text{PO} \text{TotallyOccludes} (po', z),$$

and the consequent of the theorem follows considering Definition (3).

We can obtain similar results to those above considering partial shadows (instead of the whole shadows represented in Shadow(2)), i.e. shadows of parts of a caster object $o$ that are not occluded by any other object with respect to the light source $L$. The definition of partial shadow is stated below.

$$\text{PartialShadow} (s, o, Scr, L) \equiv$$

$$\exists o, po \in \text{PO} \text{PartiallyOccludes} (o', o, L) \wedge$$

$$\neg \text{Occludes} (o', po, L) \wedge \text{Shadow} (s, po, Scr, L).$$

4.1 Relative location

The results above relate the perspective view of shadows and their casters from the light source viewpoint. It is possible, however, to reason about shadows from arbitrary viewpoints: relating shadows with occlusion suggests the distinction of five regions defined from the lines of sight between the light source, the caster and its shadow, as represented in Figure 4(b). Therefore, any viewpoint $v$ located on Region 1 will observe the shadow $s$ and the object $o$ as $\text{NonOccludesDC} (o, s, v)$; similarly, if $v$ observes $o$ and $s$ from Region 3 she should see that $\text{PartiallyOccludesPO} (o, s, v)$ and from Region 5 that $\text{TotallyOccludesNTTPPI} (o, s, v)$. Region 4 is the surface defined by the lines of sight from $l$ tangential to $o$, from where $v$ would observe $\text{TotallyOccludesTPPI} (o, s, v)$. Region 2, from where $v$ sees $\text{NonOccludesEC} (o, s, v)$, is a bisected conic surface defined by the lines connecting opposite sides of the object and its shadow, starting at infinity and stopping at the object.

It is worth noting that, if only the top part of the shadow is considered, the five regions described above can also be defined in the case where the shadow is connected to its caster (cf. Figure 4(a)), whether or not the shadow is completely cast on the ground.

By including axioms for left/right information in the theory (cf. [29] omitted here for brevity), besides of locating an observer in regions 1, 2, 3 and 4 (shown in Figure 4(b)), we would be able to say that this observer is on to the left of object $o$ and shadow $s$, and analogously to the right-hand side of $o$ and $s$.

Let the terms $\text{Region} (i)$ (where $i \in 1, 2, 3, 4, 5$) represent the regions in Figure 4(b) and let the relation $\text{located}(r, o, s)$ represent an observer $r$ located at a region $r$ with respect to an object $o$ and its shadow $s$. We define formally the relative location with respect to shadow as follows.
The next section discusses some examples and preliminary experiments of how the formalism presented in this paper could be used to infer facts about scenes.

5 Examples and preliminary experiments

Figures 5(a) and 5(b) show the resulting images from the segmentation of the photos in Figures 1(a) and 1(b) respectively. The segmentation and the correspondence between shadow and caster were handcrafted. Observed objects in Figure 5(a) and 5(b) are coloured in light grey, whereas their corresponding shadows are shown in dark grey. Shadows without known casters are shown in black.

Thus, it is possible to hypothesise (using a simple idea of abduction [32]) on ROC relations from observing object shadows. E.g., if two shadows are observed as EC it is possible to infer that their occupancy regions of respective casters are NonOccludesEC from the lightsource viewpoint. This is a trivial inference, but a similar idea may be used to hypothesise on the existence of an object that is hidden behind another, as exemplified in Figure 5(b).

Shadows (as well as occlusion) are also important cues for depth perception. The region occlusion calculus incorporates a primitive relation for nearness \(N(x, y, z)\), read as “\(x\) is nearer to \(y\) than \(x\) is to \(z\)”, along with a set of axioms originally from [34] in order to relate occlusion with comparative distance. The nearness relation is related with occlusion in ROC by the following axiom:

\[
\forall x \, y \, v \left[ \text{PartiallyOccludes}(x, y, v) \rightarrow N(v, x, y) \right]
\]  

representing that “if a body \(x\) partially occludes a body \(y\) with respect to some viewpoint \(v\) then \(x\) is nearer to \(v\) than \(y\) is to \(v\)”. It is easy to see that Axioms (5) and (12) imply the commonsense fact \(N(L, o, s)\) (for a light source \(L\), an object \(o\) and its shadow \(s\)) and consequently that \(N(L, o, Scr)\). It is a consequence of this fact that if a shadow \(s\) (from caster \(o\)) is observed cast on an object \(o'\) (\(o \neq o'\)) then we know that \(o\) is nearer to the light source than \(o'\), even though \(o'\) is never directly perceived by the observer. Formally:

\[
\text{Shadow}(s, o, o', L) \rightarrow N(L, o, o').
\]  

We start from the assumption that all the shadows shown in Figure 5 are genuine shadows (and not, for example, paintings representing shadows). Considering 5(a), labelling the components of the photos in Figures 1(a) and 1(b) respectively. The segmentation and the correspondence between shadow and caster were handcrafted. Observed objects in Figure 5(a) and 5(b) are coloured in light grey, whereas their corresponding shadows are shown in dark grey. Shadows without known casters are shown in black.

\[
\text{Shadow}(s, o, Scr, L) \land \text{Shadow}(s', o', Scr, L) \land \Phi \Psi(r(o), r(o'), L) \rightarrow \Psi(s, s')
\]
which we are in region 1 according to Figure 4(b); and the shadow connected to the pot. From this second shadow, Axiom 4 can be used to determine that there is an object, distinct from the pot, whose occupancy region is disconnected from the pot.

**Preliminary experiments**

We are currently implementing the ideas for relative location presented in Section 4.1 on our ActivMedia PeopleBot using a monocular colour camera. Shadow detection is accomplished by mapping the images captured by the camera into a HSV colour space and thresholding on V, whereby high values (light objects) are filtered out and low values (dark objects) are casters. Shadows are located within a value range in between light and dark objects whose thresholds are found by experimentation. Noise and some spurious shadows are filtered out by morphological operations. Shadow correspondence is solved using a simple heuristic: *a shadow that is connected to an object is the shadow cast by this object*. Clearly, this is only an initial approximation to the problem, as in normal conditions various shadows (from distinct objects) may connect to various objects.

In our preliminary experiments the robot was immersed in a prepared office-like environment containing one target object and where the light-source was a single sharp artificial light located above the scene at one side of the room (cf. Figure 7). The robot was set to navigate through the room, stopping after a certain time interval to analyse its position with respect to the object-shadow locations introduced in Section 4.1. In the robot set up, however, the regions represented in Figure 4(b) had to be slightly modified in order to account for the uncertainty in locating the robot on the one-dimensional regions 2 and 4. This modification is shown in Figure 6, where Regions 2 and 4 are now defined as the shaded spaces surrounding the respective original regions.

![Figure 6. Distinct regions wrt shadows used in the preliminary experiments.](image)

Although the environment was prepared in order to avoid overlapping shadows, there were shadows cast by non-target objects at the scene background. On top of that, initially, we set the robot to move on a shiny floor, from where reflexes of objects around the room (as well as the robot) were mirrored. These “spurious elements” caused the system to mis-correspond shadows and objects, locating the robot at wrong regions as a consequence. Figure 8 shows an example of the sort of segmentation we could get of images from the robot; from this view, our initial implementation located the robot correctly on region 1 (cf. Figure 6).

![Figure 7. Robot’s office-like environment at FEI, São Paulo.](image)

![Figure 8. An image from our office-like environment.](image)

In our preliminary experiments, the robot collected 118 snapshots around the target object (the black bucket in Figures 7 and 8), which was kept at the centre of the camera view. The system located the robot correctly into one of the five object-shadow regions (Figure 6) in 96 out of the 128 snapshots. Out of the 32 mis-locations, 60% were related to the borderlines separating two regions, whereas the remainder were due to noise from the scene background (mainly dark regions on the wall that were segmented as shadows and linked to the target object). These results are summarised on Table 1 that shows in the second column the number of right answers given by the system when the robot was located on the regions stated in the first column. The third column shows the total number of snapshots taken from each particular location.
Table 1. Summary of the experiments.

| Region 1 | 39 | 36 |
| Region 2 | 27 | 35 |
| Region 3 | 15 | 21 |
| Region 4 | 6  | 7  |
| Region 5 | 9  | 9  |

The obvious conclusion about the mis-locations in our experiments is to blame the usual suspect: the computer vision part should be enhanced in order to allow high-level inferences. Given the challenge of implementing robust shadow detection in a system with a moving camera we prefer instead to use a cognitive model of shadow perception, assuming a theory constraining the visual interpretations of scene components such as shadow, object and depth (and also reflections). We sketch in the current paper what such a theory would look like, and what data it should be able to explain, although the theory is not yet complete.

6 Discussion and open issues

This paper has identified the perception of cast shadows as an open problem within knowledge representation and computer vision, and has presented the theory Perceptual Qualitative Relations about Shadows (PQRS) which allows simple inferences about qualitative relations between caster, shadow, light source and screen. This formalism has been used to prove several theorems on commonsense facts about space, and has been shown to be effective in making inferences on images drawn from real world images.

Cast shadows and their relations to our depth perceptions were extensively studied during the Renaissance [13], however, only recently the perception of shadows has been considered as a subject for scientific enquiry [20]. This paper is a first step towards providing a rigorous account of the information content of shadows using formal knowledge representation techniques. We chose to build a qualitative theory using logic as lingua franca. It is worth pointing out that the choice of a qualitative theory written on logic does not substitute numerical methods, but complement them making explicit the knowledge content from a domain, adding a more abstract layer to a system, whereby it is possible to make inferences about the knowledge encoded and also to prove theorems about the theory proposed. Moreover, a logic formalisation of shadow leads to interesting venues for extending the work presented, such as the formalisation of occlusion and shadow predicates from a discretisation of the data provided by the vision system (following the guidelines proposed in [32]) or the inclusion of the defined predicates as fluents in a situation calculus framework (as proposed in [33]). This further development is left for future investigations.

The formalisation proposed in this paper is based on the observation that shadows provide a fragment of the light source viewpoint, with respect to which the region where the shadow is cast is occluded by the caster. Therefore, we extend a region-based formalisation of occlusion in order to account for reasoning about the light source viewpoint from the observation of cast shadows. With this initial formalisation we defined a method for relative location with respect to shadows. The relative location method was tested on a mobile robot navigating on an office-like scenario from where the robot was capable of deciding correctly its location in four (out of the five) relative locations defined in Section 4.1 (Figure 4(b)), as discussed in Section 5. Although preliminary, this implementation allows the experimentation of the ideas presented in this paper, grounding them on an application domain. This has the obvious advantage of making explicit some pitfalls we have uncovered during development of the proposed framework. A complete evaluation of this work, however, is still to be done. Future work shall also consider the motion of shadows as an indication of the caster motion, taking as inspiration the psychological evidence discussed in [18].

We expect that the ideas presented in this work can be further integrated in intelligent vision systems in order to endow them with the basic machinery for reasoning about space using the entire spectrum of perceptual cues contained in the visual observation of the world.

7 Acknowledgements

Paulo Santos acknowledges support from FAPESP, São Paulo, Brazil; Hannah Dee acknowledges support from EPSRC project LAVID, EP/D061334/1, UK; and Valquiria Fenelon is a graduate student sponsored by CAPES, Brazil.

This project was initiated on exchange visits from Hannah Dee and Paulo Santos sponsored by the British Council.

Many thanks to the two anonymous reviewers for the thoughtful comments.

REFERENCES


Evolutionary Genetic Programming for Autonomous Robot Motion Control

Fabrizio Abrate and Basilio Bona and Federico Tibaldi

Abstract. Autonomous motion control design is still one of the most challenging issues in mobile robotics, and currently different techniques are under theoretical investigation and experimental validation. To name only a few, we recall here the model-based control, the reactive paradigm, their integration (hybrid control), the neural network based approaches, genetic algorithms and fuzzy-logic control. Another way to achieve autonomy is studied in the field of Evolutionary Robotics (ER), and uses population-based artificial evolution, which consists of repeating fitness-based testing and selection that mimics natural evolution. The automatic methods for creating intelligent autonomous controllers, coming out from the evolution process, allow rovers to move safely in changing environments. In this paper first we demonstrate how a software tool for genetic programming can be used to generate evolvable code, implementing simple robot behaviors. Then a specific architecture combining the basic behaviors in an higher level behavior is developed. Experimental results in simulation are provided to validate the single behaviors and the proposed architecture.

1 INTRODUCTION

To be effectively used in future service applications, mobile robots must possess a substantial autonomy. As an example, let us consider simple assistive robots, i.e., mobile platforms assisting elderly people at their homes. These robots must be able to safely move around in an environment that is largely a-priori known, but that may be slowly varying during the day as a consequence of human activities: small furniture displacement, different positions of objects, motion of people inside the rooms, etc. Moreover the robots must recharge without human intervention.

When robust motion control and safe obstacle avoidance is guaranteed, robotic engineers may develop complex behaviors, as image analysis, speech or gesture recognition, manipulation, and additional functions necessary to secure tasks satisfaction and acceptability from the users.

A number of different approaches have been used for autonomous motion control design. To name only a few, the deliberative (i.e., model based) control, the reactive or reflexive paradigm, and the integration of both in the so-called hybrid control (see [1, 9] and the references herein). Other approaches are based on neural networks [14], genetic algorithms [12, 13] and fuzzy-logic control [4].

One of the emerging area of research within the much larger field of fully autonomous robots is evolutionary robotics (ER). ER uses population-based artificial evolution [5] to evolve autonomous robot controllers (sometimes called robot brains).

The process of controller generation consists of repeated fitness-based testing and selection that roughly mimics natural evolution.

One of the primary goals of evolutionary robotics is to develop automatic methods for creating intelligent autonomous robot controllers, exploiting what comes out from the evolution, in such a way that they do not require direct programming by humans. It turns out that reaching this goal would allow robots to move safely and to adapt to dynamic environments.

Genetic programming (GP), introduced by Cramer [3] and further developed in [6], [8] and [10], is a relatively recent type of evolutionary computation that came out as an extension of genetic algorithms. GP considers computer programs and makes them evolving under the genetic adaptation paradigm, so that the fitness evaluation can be performed only after running the program itself. From the point of view of motion control, this approach gives the possibility to build pieces of code that implement series of commands or functions whose input is the sensory data and the output is directly supplied to the motors. At the end of the adaptation cycle, the “best-fitted” candidate computer program is available and can be directly coded into the target robot CPU.

One of the first attempt to implement simple behaviors can be found in [7], where a simple wall-following task is evolved using GP techniques.

In this paper, a twofold objective is pursued. First we demonstrate how a software tool for genetic programming (μGP and μGP3) [2, 11], which was originally designed to prepare assembly programs for specific microprocessors, may also be used to generate evolvable code (Matlab code in our case) implementing simple robotic behavior, namely wall-avoidance, area coverage/exploration and recharge.

Then, having secured the ability to generate simple behaviors identified by a single fitness function, the second objective is to develop an architecture that combines these behaviors, exploiting again the evolutionary power of μGP3.

The paper is organized as follows: Section 2 gives an account of the architecture of the simulator, where each controller is tested, Section 3 reports the results of the three simpler behaviors, while Section 4 provides the architecture of the supervisor. Section 5 presents and discusses the simulation results and Section 6 draws the conclusions.

2 Simulator

The simulator includes an arena, a rover and its controller. The controller code is based on Matlab language, for two reasons: first to allow a simple and fast evaluation/visualization of the results, and second, to test the μGP3 ability to interface with high level languages.
2.1 The Arena and the Rover

The simulated rover motion takes place inside a rectangular arena whose area is \( L \times H \), where \( L = 20 \) and \( H = 10 \); for simulation purposes, the arena is divided into 200 (1 \( \times \) 1) elementary cells. The arena is limited by four walls (see Figure 1); a recharge area is always present in the arena: the rover charges instantly when it enters this area.

The rover is simulated as a massless triangle carrying simulated onboard sensors. At each time step \( k \) the rover state is defined by three quantities: two cartesian coordinates \( x(k) \) and \( y(k) \), measured with respect to the arena center (reference system \( \mathcal{R}_0 \)), and the absolute heading \( \theta(k) \); they together define the pose \( p(k) \) of the rover: \( p(k) = [x(k) \ y(k) \ \theta(k)]^T \) (see Figure 1).

The rover speed vector is \( s(k) = [\dot{x}(k) \ \dot{y}(k)]^T \), and the axial speed is defined as \( s(k) = \|s(k)\| = \sqrt{x^2 + y^2} \); rotation speed is \( \omega(k) = \dot{\theta}(k) \), positive if counterclockwise. Rover axial speed is assumed constant in the interval \( k \) and directed along the unit vector \( [\cos \theta(k) \ \sin \theta(k)]^T \).

![Figure 1. The simulated arena.](image)

2.2 Controller

The controller is structured to perform four successive steps:

1. Self-sensing and environment sensing, using proximity sensors, recharge-area location sensor, battery charge status sensor. Each sensor provides a simulated noise-free signal according to the pose of the rover inside the arena and the time elapsed from the previous recharge.
2. Behavior activation: a rover behavior is a sequence of actions that a rover performs in order to complete one or more tasks specified by the user; every action is generated by the rover control system according to the rover state, sensors readouts, and tasks objectives. Wall-avoidance behavior, exploration behavior, recharge behavior.
3. Behavior fusion: the supervisor carries out an integration of the different behaviors to generate the motion command;
4. Rover motion: the command resulting from the behavior fusion is generated and supplied to the motors.

2.3 Sensors measurements

The rover behavior at step \( k \) depends on its state and on the readings from sensors. For simplicity only three types of sensors were considered and simulated, namely proximity sensors, recharge-area location sensor and battery charge status sensor.

2.3.1 Proximity sensors

Four onboard proximity sensors measure the distance \( d_i, i = 1, \ldots, 4 \) from the four walls; each sensor provides a quantity \( z_{w,i}(k) \) that is inversely proportional to the distance:

\[
z_{w,i}(k) = \begin{cases} 
    d_p - d_i(k) & \text{if } d_i(k) < d_p \\
    0 & \text{if } d_i(k) \geq d_p 
\end{cases}
\]

where \( d_i(k) \) is the distance from the \( i \)-th wall and \( d_p \) is the maximal sensing interval of the sensors. The four readings are stored in a vector \( z_w(k) \).

2.3.2 Recharge-area location sensor

The recharge area is simulated as a circle centered in \((x_c, y_c)\) with radius \( \rho \). The onboard recharge-area location sensor measures the rover distance from the recharge area center \((z_r(k))\), and the angle \( \alpha \) between the rover absolute heading and the straight line connecting the rover center to the recharge area center \((z_c(k))\) (see Figure 1). The recharge area is sensed only if it is in the sensor angle-of-view \( \beta \) and inside the sensing interval \( d_r \).

2.3.3 Battery charge status sensor

The battery charge status sensor provides a value \( 0 \leq z_b(k) \leq 1 \) proportional to the battery charge level. The value is 0 if the battery has no residual charge and 1 if the battery is fully charged.

The recharge-area location and the battery status are stored in the battery vector \( z_b(k) \).

2.4 Sub-controller generation and fitness

At each generation \( g = 1, \ldots, G_{max} \), \( \mu G_p^3 \) creates a maximum number of individuals \( I_{max}(g) \) according to its internal rules. The maximum number \( G_{max} \) of evolutionary generations is established by the user.

Each individual \( i \), with \( i = 1, \ldots, I_{max}(g) \) is a piece of Matlab code, built according to a set of rules established by the evolutionary behavior imposed by \( \mu G_p^3 \) and characterized by some pre-defined fitness function; each piece of code so generated represents a potential sub-controller candidate \( CS_{i,g} \). We prefer to use the term “sub-controller” instead of “behavior” to point out the real code that implements the rover motion.

The fitness of each individual candidate is measured performing a simulation; to avoid overfitting and biases, each \( CS_{i,g} \) candidate behavior is tested using several initial poses; in our case, \( m_p = 12 \) different initial poses were used.

The overall individual fitness function is evaluated considering the worst fitness function of the \( m_p \) simulations. At the end of all the \( I_{max}(g) \times m_p \) simulations of the current generation, a fitness vector \( J_p \in \mathbb{R}^{I_{max}(g)} \) is generated and supplied to \( \mu G_p^3 \) that starts a new evolutionary cycle.

The fitness function is tuned according to the desired overall behavior of the mobile rover, as specified in Section 3.
Figure 2 illustrates the recursive procedure implemented by the evolutionary programming $\mu Gp^3$.

In practice a generic $CS_{i,g}$ is nothing else that a sequence of Matlab instructions performing assigned operations on input data. In our case, the operations are limited to sums and products of previously stored results, plus the multiplication of the result by a weight $w$, $|w| \leq 1$, plus the storage of the resulting operations in shared memory cells.

The $CS_{i,g}$ code accepts as input data the content of a vector of real numbers that includes the rover measurement vector $y(k)$, containing the previous step rover pose $p(k)$, the proximity sensors measurements $z_w(k)$ and the battery vector $z_b(k)$.

$$\begin{pmatrix} y(k) \\ p(k) \\ z_w(k) \\ z_b(k) \end{pmatrix} \in \mathbb{R}^{10} \quad (2)$$

The $CS_{i,g}$ code terminates generating as last step the rover command $u(k+1)$, containing the desired axial speed and heading

$$u(k+1) = \begin{pmatrix} s_{d}(k+1) \\ \theta_d(k+1) \end{pmatrix} \quad (3)$$

Each evolved individual $CS_{i,g}$ is plugged into a simulation program, where its fitness is measured and used by $\mu Gp^3$ to produce a new generation of candidates $CS_{i,g+1}$.

The simulation scheme is shown in Figure 3, where $RP$ is the block that computes the rover pose using odometry measurements.

![Figure 3](image-url)

**Figure 3.** The $CS_{i,g}$ sub-controller simulation scheme.

The evolutionary mechanism of $\mu Gp^3$ decides on the following issues

- which operator type to use (sum or product of cell values);
- which are the inputs to the operator, i.e., definition of the stack indexes where the inputs are stored;
- which weight to apply to each operator output;
- where to store the results, i.e., definition of the output stack cell indexes;
- which stack elements to use to create the desired command signal $u$.

### 2.5 Rover command generation

The simulation takes into account the physical limits that constrain the rover command values. The axial speed $s_{d}(k)$ can vary between $s_{\text{min}}$ and $s_{\text{max}}$.

Given a sampling time $T$ and a desired heading $\theta_d(k)$, the angular speed $\omega(k)$ is computed in such a way to guarantee the minimum heading error $\theta_d(k) - \theta(k)$ compatible with the some angular speed constraints.

### 3 Simple behaviors

The global rover behavior is obtained combining three simpler behaviors, namely wall avoidance, exploration and recharge.

#### 3.0.1 Wall-avoidance

Wall-avoidance behavior rewards free rover motion inside the arena, securing no (or very few) contacts with the four walls. Wall-avoidance behavior is implemented by a wall-avoidance sub-controller.

Wall-avoidance fitness function is given by the number of wall impacts recorded during the simulation. If necessary, wall-avoidance behavior can be transformed, with minor adjustments, into other similar behaviors, as “obstacle-avoidance” or “wall-following”.

#### 3.0.2 Exploration

Exploration is a behavior that rewards those rovers exploring a large number of arena cells.
Two fitness functions are used; the primary one counts the number of explored cells, while the secondary counts the impact number when the explored cells number is the same for two individuals. To avoid a biased behavior, the cells already explored are not counted again.

3.0.3 Recharge

The recharge fitness function counts the number of motion steps needed to reach the recharge area, and the resulting command minimizes this fitness.

4 Supervisor

In order to take into consideration a more general controller, able to navigate the rover in an unknown arena, avoiding wall impacts and maximizing the number of explored cells, while visiting the recharge area in due intervals, a supervisor was evolved using $\mu Gp^3$, $\mu Gp^3$ blends the three previous behaviors assigning a variable weight to each of them according to the rover state and sensors reading. This is what we call a behavior fusion and is similar to the evolved subsumption mechanism scheme presented in [7], the main differences being the evolutionary-based generation of sub-controllers, the language used, and the overall characteristics of the $\mu Gp^3$ software.

To choose which sub-controller is activated, according to the rover state, the supervisor compares the sensors measurements with some thresholds defined by $\mu Gp^3$. In our experiment, the rover state can meet two potentially critical conditions, namely CrashDanger (CD) and BatteryLevelDanger (BLD). The rover is in CD if it is near to a wall and $\theta(k)$ points toward it, and it is in BLD if the battery charge level is low. These two situations are handled using two thresholds $K_w$ and $K_e$.

The speed and heading commands generated by the supervisor take into account different combinations of the critical conditions CD and BLD.

The supervisor activates a specific function $f(\cdot, \cdot)$ to manage the most critical conditions, i.e., when the rover status verifies at the same time CD and BLD. Two critical thresholds ($K_u$ and $K_p$) sort out the most dangerous condition between CD and BLD. If the rover status exceeds these thresholds, the function computes the average value between the two commands.

4.1 Rover state update

During the simulation, the rover state is updated according to

$$x(k+1) = x(k) + s(k)\cos(\theta(k))$$

$$y(k+1) = y(k) + s(k)\sin(\theta(k))$$

In practice, the rover first moves at the required speed, maintaining the previous step heading $\theta(k)$, then turns to reach the required heading; this is one of the possible strategies, another being to turn first and then move. Since no path planning is considered, these strategies are simple but effective.

5 Experiments

A certain number of experiments have been carried out in order to test each sub-controller generated by the evolutionary process.

### Wall Avoidance

$\mu Gp^3$ has to minimize the fitness function, which counts the number of impacts made by the rover. The emerging strategy consists in moving with the lowest axial speed and with the maximum angular speed in this way the rover avoids impacts in every situations, but it is not able to explore the arena.

### Exploration

Every elementary cell of the arena is marked when the rover remains inside it for at least one simulation step, and $\mu Gp^3$ has to maximize number of the marked cells. In this case the rover can explore a big portion of the the arena, but it impacts the walls many times.

### Recharge

$\mu Gp^3$ minimizes the fitness functions that counts the steps made between two recharges. The rover moves as straight as possible in order to use the lowest possible number of steps, and this behavior causes few wall crashes.

### Supervisor

The Supervisor behavior evolution is different with respect to the previous ones since it takes into consideration the global rover behavior. The evolution has to adapt the thresholds to allow the rover to explore while avoiding impacts, and recharge the battery. In this case, $\mu Gp^3$ does not evolve a piece of code, but only some real weights the four real weights $K_w$, $K_p$, $K_u$, $K_e$ defined in Section 4. Exploration sub-controller is only activated when the rover is not is a potentially critical condition CD or BLD.

5.1 Experimental Results

Some relevant statistics related to the four behaviors are summarized in Table 1.

Table 1. Statistics related to the four behaviors: B1 is the Exploration behavior, B2 the Wall Avoidance, B3 the Recharge, B4 the Supervisor

<table>
<thead>
<tr>
<th>Statistics</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
</tr>
</thead>
<tbody>
<tr>
<td>generation #</td>
<td>150</td>
<td>237</td>
<td>2430</td>
<td>488</td>
</tr>
<tr>
<td>explored cells #</td>
<td>195</td>
<td>4</td>
<td>106</td>
<td>74</td>
</tr>
<tr>
<td>explored cells # (max)</td>
<td>165</td>
<td>1</td>
<td>25</td>
<td>35</td>
</tr>
<tr>
<td>explored cells # (mean)</td>
<td>164</td>
<td>2</td>
<td>56</td>
<td>54</td>
</tr>
<tr>
<td>battery expirations # (max)</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>battery expirations # (mean)</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>battery expirations # (max)</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td># of crashes, (max)</td>
<td>88</td>
<td>0</td>
<td>47</td>
<td>0</td>
</tr>
<tr>
<td># of crashes, (min)</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td># of crashes, (mean)</td>
<td>65</td>
<td>0</td>
<td>14</td>
<td>0</td>
</tr>
</tbody>
</table>

The maximum, minimum and mean values for the number of explored cells, battery expirations and crashes are obtained by taking respectively the maximum, minimum and the average value over $m_p = 12$ experiments. Recalling that the arena is divided into 200 cells, notice that the maximum number of explored cells by the Exploration behavior is 195, and the average is 184, which is the 92% of the entire arena.

The maximum number of battery expirations in the Recharge behavior is 1, with average 0. In the Exploration behavior the crashes are quite frequent, since trying to maximize the coverage of a certain area also leads to increase the number of impacts. Instead, the number of crashes in the Wall Avoidance behavior is 0 on average.

These results confirm that the three single behaviors perform according to our expectations. The Supervisor behavior allows the simulated rover to explore on average 54 cells, which is only the 27% of
the arena, similar to what has been obtained by the Recharge behavior. This is probably due to the fact that the Recharge behavior, since it minimizes the path to reach the recharge area, also minimizes the number of explored cells, which is in contrast with the effects of the Exploration behavior. However, the battery is always timely recharged and the rover never impacts.

To sum up we do not claim that the performances of the Supervisor behavior are optimal in any sense, we simply observe from the statistics that the overall behavior is satisfactory and relevant from the application point of view, allowing a rover to autonomously adapt to potentially different environments while performing a task which is the result of different subtasks.

5.2 Discussion

The control system generated by $\mu$Gp$^3$ allows the rover to adopt the suitable behavior: it can avoid impacts against the walls, recharge its battery and explore the arena.

Wall-avoidance and Recharge behaviors are crucial for the rover life, since it cannot continue to explore, if it crashes or its battery expires. Therefore, the respective sub-controllers have an higher priority than the Exploration one, resulting in the rover stopping to explore as soon as its battery charge level is too low or it is too close to a wall. For this reason, the rover cannot go too far from the recharge area, since otherwise it cannot return to it before the battery charge expires. Moreover, the rover does not go near the arena perimeter, since it could be too close to a wall and it could impact against it.

Figure 4 (b) shows the rover motion between two successive recharges. It starts from the recharge area, then it leaves it to start exploring; as soon as the battery level decreases, it moves again towards the recharge area. Figure 4 (a) shows a complete simulation when the Supervisor behavior is active.

6 Conclusions

This paper presents a motion control solution for a rover, obtained genetically evolving a Matlab based computer code. Three different basic behaviors were evolved, namely Wall Avoidance, Recharge and Exploration. These were successively combined by another genetically evolved supervisor program to produce a robust general motion control algorithm. The experimental results in simulation show that the genetically evolved single behaviors compliant with their fitness specification, and the Supervisor behavior shows the sought characteristics as a fusion of the three single-evolved behaviors.

The technique can be used to obtain behaviors similar to the Supervisor and be employed in practical applications, such as service space robotics, where intelligent, task-oriented and reactive behaviors are of primary interest.

In the near future, some additional work will be developed. First we intend to employ a more realistic robotic simulator to take into account dynamical effects in the rover motion and more realistic sensor models. Then the final idea would be to implement on real rovers individuals evolved in simulation, and to verify if with few modifications the code can produce acceptable fitness values in real-world experiments. At the same time a real multi-objective genetic programming tool, derived from the present $\mu$Gp$^3$ software, will be tested and the results compared with the present ones.

7 Acknowledgements

Giovanni Squillero, Ernesto Sanchez, Danilo Ravotto, Alberto Tonda, Alessandro Salomone made this work possible by numerous contributions and constant support during the use of $\mu$Gp$^3$.

This work was partly supported under the “MACP4Log” grant from Regione Piemonte.

REFERENCES

Robot Topological Map Generation from Formal Route Instructions

Mohammed Elmogy¹ and Christopher Habel¹ and Jianwei Zhang¹

Abstract. Mobile robots and Humanoids need to use spatial information about their surrounding environment in order to effectively plan and execute navigation tasks. This spatial information can be presented in various ways to increase the interaction between the human and the robot. One of the more effective ways is by describing the route verbally which bridges the gap between the forms of spatial knowledge of such robots and the forms of language used by humans.

In this paper, we build a topological map for robot route description. This map represents the route’s motion actions and spatial relationships graphically to plan robot’s navigation task. The map is generated by using Formal Route Instructions (FRIs) to simplify the route description process and also to avoid ambiguity. FRIs are designed to be simple, easy to use, and suitable for naïve users.

1 INTRODUCTION

A robot is an intelligent, multi-purpose machine, which can carry out a variety of online tasks. An essential aspect, which distinguishes robotics from other areas of AI, is the interaction of robots with humans and with their surrounding environment. In a robot system, various autonomous components such as sensing, recognition, planning, control, and their coordination must cooperate in recognizing the environment, solving problems, planning a behavior, and executing it. In order to make this interaction more intelligent, a robot needs functions such as: the ability to understand the environment by visual recognition, the ability to perform dexterous manipulation using force, tactile, and visual feedback, the ability to plan task procedures, the ability to communicate with humans, the ability to learn how to perform tasks, the ability to recover from errors, and so on [1]. All of these functions are required for robot intelligence to be realized adequately.

Due of the potential for interaction with humans, research in humanoid robotics has made significant progress in recent years. The key reason for preferring humanoids is their shape, which seems to be optimal for being taught by humans and learning from humans. Humanoid robotics labs worldwide are working on creating robots that are similar to humans in shape and behavior. These similarities have been proven to facilitate the communication task between the human and the robot. Recent studies also show additional advantages of humanoids which can be summarized into three points. The first advantage is that human interaction with robots is easier if robots are humanoid because of its shape. The second is that acceptance of robots by humans is easier for those with humanoid shape. The last advantage is that the efficiency of teaching and programming a robot is highest with humanoids [2-4].

A more natural interaction between the humans and the humanoid can be achieved if there are ways to bridge the gap between the forms of spatial knowledge of such robots and the forms of language used by humans, enabling them to communicate by using this shared knowledge. A natural language interface supports more ‘natural’ styles of interaction between robots and users. Most typical scenarios include a human user instructing a robot to perform some actions in these scenarios, such as moving to a location or manipulating an object. Route descriptions, which are used to guide the robot in executing navigation tasks in the surrounding environment, are considered as one of the most important natural language interfaces between human and robots in realizing effective human-robot interaction.

The remaining part of this paper is structured as follows. In the next section, robot navigation is elucidated. How to generate good route instructions and what are the different broad categories of route-based navigation in robotics are presented in details. In section 3, the main building blocks of our system architecture is explained. The route processing module is discussed in details with concentrating on how the Formal Route Instructions are used to describe the user’s route to the robot. Also, the topological map representation of the route description is introduced.

2 ROBOT NAVIGATION

Navigation has always been an interdisciplinary topic of research, because mobile agents of different types are inevitably faced with similar navigational problems. Therefore, human navigation can readily be compared to the navigation of other biological organisms or of artificial mobile agents like mobile robots. Thus navigation can be described as a coordinated and goal-directed movement through the environment by biological organisms or intelligent machines. It involves both the planning and execution of movements.

Following Montello [5] we consider navigation to consist of two components: locomotion and wayfinding. Locomotion is the movement of the agents’s body around an environment,
coordinated specifically to the local or close surroundings. There are various modes of locomotion which are important because they determine much about the way we acquire and process information as we move. Thus, with respect to humans ‘locomotion’ concerns the level of motor processes and automatic (or unconscious) cognitive and sensor processes. On the other hand, wayfinding refers to the goal-directed and planned movement of a body around an environment in an efficient way. It requires a goal locality, a destination the agent wants to reach. A wayfinding action such as following verbal directions clearly requires the activation of long-term knowledge representations (the cognitive map) into working memory in order to access one’s knowledge of place layouts [5, 6].

Thus, ‘wayfinding’ concerns higher cognitive and communicative processes. The great majority of acts of navigation involve both locomotion, and wayfinding components. Evidence for the distinction’s validity is provided by the simple fact that you can have one without the other. They are generally components of an integrated system of navigation that can be separated only conceptually, but sometimes they can be separated literally.

With respect to robots, their navigation system is based on three basic components. The first is planning, which computes a trajectory or path between two points (starting and end points). Note that path planning for robots has to include both levels, wayfinding and locomotion. The second component—often also called navigation—provides the robot with those pieces of information needed to move and to follow the computed path/trajecotry. The last component is environment representation, which enables the robot to know its location and its heading. For indoor robot navigation, systems are classified into three groups: map-based navigation using predefined geometric and/or topological models, map-building-based navigation constructing geometric and/or topological models on its own, and mapless navigation using only object recognition and actions associated to these objects [7].

In vision based robot navigation systems, vision sensors (cameras) are used to provide a huge amount of data that should be processed in real time. The elements extracted from the data are compared to reference models which are stored previously as knowledge database. In these systems, they concentrate on the field of shape understanding using the data captured from the vision sensors. Environment interpretation stresses the use of natural landmarks to ease the navigation and the pose estimation of a mobile robot.

### 2.1 Robot Route Instructions

Route instructions specify spatial information about the environment of the route and temporal information about the actions (movements, turns) to be performed. Human route instructions are usually conveyed either verbally in spoken discourse or written texts, or by graphical means, i.e. by illustrating the route on a map, or drawing sketch-maps. A third possibility is to combine these two kinds of external representations leading to multimodal route instructions. Whereas verbal route instructions focus on the actions to be performed and take the spatial environment as the frame for these actions, maps and other pictorial representations foreground the spatial environment without possessing adequate means for representing the actions [8]. Nevertheless, all type of route instructions have to provide correlated actions, paths, tracks, positions and landmarks to describe the navigation path to the navigating agent. All of these route instruction components can be classified and categorized into main groups to facilitate the analysis of the navigation task [9].

Good route instructions should contain adequate information about the following two aspects. The first aspect concerns navigation actions, in particular locomotion actions and perception actions, which are performed by the robot to reach its destination. The second is the spatial environment, in which the intended locomotion of the robot will take place. The instructor’s primary task is to choose a good combination of communicational means to transfer the relevant information concerning both aspects to the robot [8].

MacMahon [10] proposes four basic actions to be used in following route instructions: turning in place, moving from one place to another, verifying a view description against an observation and terminating a current action. The primary characteristic of a path is the change of location. Turns can be viewed as changes in orientation. These considerations led to four basic types of Navigational Information: moves, turns, positions and orientations. Altogether, moves and turns can be subsumed under the general notion of actions, and positions and orientations can be viewed as verifications.

The field of route-based robot navigation is regularly be classified into four categories. The first category is Guidance, which is mainly concerned with directly leading an agent by external cues – either by following a particular gradient or moving to match the current sensory image with a stored image of the target or of the surroundings. In all of these cases, the robot tries to locally maximize a predefined criterion without knowledge of spatial relations in the environment or about its own position. The second is the Guidance Place Recognition – Triggered Response. For place recognition based strategies, complex spatial behaviors are triggered at distinct points in space. Once the correct place is recognized, the associated action (e.g., movement in a particular direction or guided behavior) will lead to complex trajectories. The main problem of this strategy obviously consists in the correct identification of a place. The third category is Topological Navigation, which describes navigation based on topological networks, is thus a more flexible extension of place-triggered navigation. The basic elements of this type of networks are places and some connections between these places. Finally, the last category is the Metrical Navigation. Unlike the last two approaches, which divide space into a small number of distinct places and the space in-between, metrical navigation does not require such a distinction in principle. The metric most frequently used is Euclidean, thus distances and angles are well defined and can be used to drive spatial navigation. Pre-existing maps, which specify the metrical relations between objects in the environment of the agent, are often supplied directly or are autonomously constructed by triangulation and integration of sensory information [11, 12].

### 2.2 Topological and Metric Maps for Robots

Building a representation of the environment is an essential task for a mobile robot that aims at moving autonomously in the surrounding space. The representation of spatial knowledge can be considered at two different levels of abstraction. On the one hand, metric (geometric) maps represent the environment according to the absolute geometric position of landmarks. On the other hand, a topological map is a more abstract representation that describes relationships among features of the environment, without any absolute reference system [13].

Approaches in the metric paradigm generate fine-grained, metric descriptions of a robot’s environment. In these representations, the robot’s environment is defined by a single global coordinate system, in which all mapping and navigation
takes place. Typically, the metric map is structured by a grid with each cell of the grid representing some amount of space in the real world. These grids become quite sophisticated at representing the spatial structure of the world.

On the other hand, approaches in the topological paradigm generate coarse, graph-like descriptions of environments, where nodes correspond to significant, easy-to-distinguish places or landmarks, and arcs correspond to actions or action sequences that connect neighboring places. Topological maps are qualitative descriptions of the robot's workspace, in which the environment is represented as places and connections between places. Topological maps can also be more compact in their representation of space, in that they represent only interesting places and not the entire environment. Topological maps have been proved as very successful for mobile robots [10].

In principle, topological maps could be scale to the size of large-scale indoor environments better than metric maps could, because a coarse-grained, graph-structured representation is much more compact than a dense array, and more directly suited to problem solving algorithms. However, purely topological maps have difficulty in distinguishing adequately between different places, and have not been applied to large environments in practice. Recent progress in metric mapping has made it possible to build useful and accurate metric maps of reasonable large scale environments, but memory and time complexity pose serious problems [14].

3  INSTRUCTED NAVIGATION FOR HUMANOID ROBOTS (INHR)

In the present section we exemplify some features of the INHR system (Instructed Navigation for Humanoid Robots), which is designed with the purpose of efficient and effective navigation of Fujitsu HOAP-2 Humanoid robots (Humanoid for Open Architecture Platform), with navigation tasks in a miniature city. The miniature city is built on a 5m x 3.2m area which is suitable to the HOAP-2 Humanoid dimensions (50cm x 25cm x 16cm) [15]. Figure 1 shows the layout and the physical realization of our miniature city. Currently, route instructions are communicated to the humanoid by using formalized expressions, called Formal Route Instructions (FRIs) via a Graphical User Interface (GUI).

The system is composed of three main modules as shown in Figure 2. The first module is the route processing module which receives route descriptions from the user and transfers them into topological maps (TM). TMs are connected in the spatial planning stage (module 3) with the output from the vision stage to calculate the Humanoid actions and generate the motion commands. The second module is the vision processing module. It starts with capturing video streams from the Humanoid’s cameras, and processes these streams to detect and recognize the landmarks in the miniature city. It is also involved in calculating the distance between robot and the recognized landmark. The final module is the action processing. This module constructs a structured representation of the Humanoid motion commands, to be realized by the Humanoid’s actuators.

This paper focuses on the route processing module stages and how to process the route given by the user to generate a topological representation of the spatial environment. The route processing module consists of three main stages. The first is the stage of processing FRIs (Formal Route Instructions), which are expressions in a formalized style used by the human instructor to build route descriptions preventing misunderstanding and ambiguity. In this stage, the route’s reference is an intrinsic reference which describes all objects and directions with respect to the Humanoid’s body. The intrinsic reference is used to avoid the conversion process between different reference systems (relative, intrinsic, and absolute). The second stage is the CRIL (Conceptual Route Instruction Language) [9] representation which is used to analyze the route description to motion actions, spatial relations, and landmarks. This stage interacts with the landmark’s database to retrieve the main features of each landmark described in the route. The final stage is the topological map presentation which is used to translate the CRIL actions to a spatial graph. This graph is used to connect each landmark in the route description with its neighbor landmarks in a network by using route spatial relationships. This graph is submitted to the action processing stage to process the route and generate the Humanoid actions. In the next subsequent sections, the three stages of the route processing module will be elucidated in details.

![Figure 1](image-url)  
**Figure 1.** Humanoid miniature city. **Top:** Layout of the miniature city with an exemplary route (fat line), whose verbal description is discussed below. **Bottom:** Physical realization of the miniature city.

When robots navigate in indoor environments, it requires an adequate representation of the working space. This representation should be abstract enough to facilitate higher-level reasoning tasks like strategic planning or situation assessment, and still be detailed enough to allow the robot to perform lower-level tasks like path planning/navigation or self-localization. A common belief in the robotics field is that robots need to represent and reason about information at different levels of abstraction at the same time.

If the environment is proposed to be represented by local metric maps connected into a topological network. This technique allows the use of maps that are not metrically consistent on the global scale, although they are metrically consistent locally [13].
3.1 The Level of Formal Route Instructions

FRIs provide elementary instruction statements in a formalized language, which eventually leads to a sequence of INHR-internal commands to be executed by the Action Processing Module. FRI is intended as a semi-formal language to be used by non-expert users via a structured GUI; currently we prepare an empirical test of the usability of the FRI-interface. Each FRI represents an instruction, which relates motion actions with some landmark(s) by use of suitable spatial relationships. The inventory of FRIs contains—currently—three classes of commands; this inventory reflects the inventory of CRIL, which will be described in section 3.2 (see [9], [16]). The first class includes position commands that refer to the current position of the robot. These commands are primarily used to identify the start and end position of the robot by using a spatial preposition and a landmark. They can also be used during the robot route description to describe relevant confirmations of the robot’s current position with respect to a landmark. These commands are represented in FRI by using three different commands: $START(), $STOP(), and $BE() as shown in Table 1. The prominent role of landmarks for route instructions is reflected by the syntactical condition that position FRIs have landmarks as obligatory arguments (see column ‘syntax’ of Table 1; optional arguments are coded via square brackets). The second class is that of locomotion commands. These commands give the robot the order to move to a particular region or to go in a particular direction with respect to certain landmarks. In FRI, this type of commands can be presented by different operators depending on the situation such as: $GO(), $CROSS(), $PASS(), and $FOLLOW(). The last category is that of change of orientation commands. These commands are used to change the direction of the robot by turning or rotating to a specific direction [16]. $TURN(), and $ROTATE() commands are used in FRIs to represent the orientation changes of the robot during the execution of its navigation task.

Additionally, FRIs can be used to build complex instructions that are structured as ‘blocks’. Each block begins with a $GO() statement and ends before the next $GO() statement—except for the starting and ending statement. All statements in the block are processed in serial and executed as a single sub-route in the Humanoid navigation task.

Table 1 gives an overview of the commands mentioned above exemplified with corresponding verbs, prepositions, and adverbs, which are told to naïve users for introducing them in FRI-usage. For example, $GO() command can be represented by the following syntax:

$GO([Count], [Direction]| [Pre1], Landmark1, [Pre2], [Landmark2])

The first parameter in the syntax (Count) presents the number of turns whereas Direction specifies the direction of the turn. Pre represents the formal counterpart to a preposition or an adverb which will be used in the spatial statement. Finally, Landmark specifies the landmark name. The FRI sequence presented in Table 2 is constructed to lead the robot from the railway station to the town hall in the miniature city (cf. the depiction of this route in Figure 1. In this route, the robot is instructed to begin at a starting point with the railway station to its left. First, the robot has to move to the crossroads, then it is instructed to cross the street, to pass a building on its left, and to move to the next crossroads. Afterwards the robot should turn left, walk down the street until the next crossroads, where it has to turn right. Then he shall walk straight on, pass the Burger King Restaurant to the right, and the C&A department store to the left, and has to go on until reaching the next crossroads. The next instructions include the robot’s crossing of the street, going straight on, passing a church to the left, passing Karstadt department store to the right, and then turning right at the next crossroads. Finally, the robot has to keep walking down the street until it is standing to the right of the town hall, which is determined as its destination.
3.2 CRIL Representation

Tschander, Schmidtko, Habel, Eschenbach and Kulik [9] proposed the idea of a Geometric Agent that simulates instructed navigation in a virtual planar environment. In their approach Conceptual Route Instruction Language (CRIL), that is kindred to Jackendoff’s conception of semantics (see [17], [18]) are used to represent the meaning of natural language instructions. CRIL-expressions are constructed from a basic inventory of descriptive operators. On the one hand, CRIL-expressions specify the semantics of natural language expressions using methods of formal semantics; in particular CRIL functions as the output of an experimental interface from written instructions in German. On the other hand, CRIL is an internal language, that (1) is the representational medium of spatial reasoning [9], (2) it relates internal models to perceptual objects, and thus (3) specifies actions the Geometric Agent can carry out.

In the INHR system, we use CRIL to transfer FRI instructions to actions and spatial relationships. In CRIL, there exist three types of conceptual entities extracted from route descriptions that are currently in the foreground of the INHR approach. The first type concerns motion actions, which can be considered as CRIL counterparts of verbs of motion. The second type includes spatial concepts, which provide the specification of relations between spatial entities and regions, e.g. those that are used for relating paths and landmarks with each other. The last type concerns landmarks and paths and there distinguished features. These components are considered to be derived from the CRIL implementation. In the following subsections, they will be considered briefly.

### 3.2.1 The Inventory of Actions

Natural language descriptions of motion frequently involve two kinds of expressions that connect to spatial structure [9]: a verb of motion (such as ‘go’, ‘turn’, ‘enter’, …) and a directional adverb or a directional prepositional phrase (such as ‘into the zoo’, ‘through the park’, ‘back’, ‘straight on’) [19].

Verbs of “wayfinding relevant actions” are used to indicate the robot behavior during its navigation task. As seen in Table 3, they can be classified into four classes. The first is that of verbs of position, which we use in INHR to represent the current position of the robot. These verbs can be used to identify the start and end position of the robot by using a spatial preposition and a landmark. They can be used also during the robot route description to make confirmation to its orientation. These verbs are represented in CRIL by using the basic concept !BE_AT(x, p), where x is the navigating agent—in our case the robot—and p is the position. The second group is that of verbs of locomotion. These verbs instruct the navigating agent to move in a particular direction or to a particular region often specified with respect to a certain landmark. In CRIL, verbs of this type are based on the concept !GO(x, w), where w is the path to be move on. The third group concerns notifying, this means instructing the navigating agent that it has to perceive an object to insure that it is on the correct way. For this type of instructions in CRIL the basic concept !VIEW(x, r) is used. Whereas notifying is common in natural language instructions, explicitly as well as implicitly, the current inventory of FRI does not include this class; since kindred ways of description can be realized by the block structure (see Table 2). The last group concerns verbs of change of orientation. These verbs are used to instruct navigating agents to turn (move on a curved path) or to rotate, i.e. to change the orientation without locomotion. Both subtypes are based on the basic concept !CH_ORIENT(x, d), where d is a direction of the turn.

### Table 1. The FRIs Commands.

<table>
<thead>
<tr>
<th>Command Type</th>
<th>Command Name</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>$START()</td>
<td>!START([Pre1], [Direction], Landmark1, Pre2)</td>
</tr>
<tr>
<td></td>
<td>$STOP()</td>
<td>!STOP([Pre1], [Direction], Landmark1, Pre2)</td>
</tr>
<tr>
<td></td>
<td>$BE()</td>
<td>!BE([Pre1], [Direction], Landmark1, Pre2)</td>
</tr>
<tr>
<td>Locomotion</td>
<td>$GO()</td>
<td>!GO([Pre1], [Direction], Landmark1, Pre2)</td>
</tr>
<tr>
<td></td>
<td>$PASS()</td>
<td>!PASS([Pre1], [Direction], Landmark1, Pre2)</td>
</tr>
<tr>
<td></td>
<td>$FOLLOW()</td>
<td>!FOLLOW([Pre1], [Direction], Landmark1, Pre2)</td>
</tr>
<tr>
<td></td>
<td>$CROSS()</td>
<td>!CROSS([Pre1], [Direction], Landmark1, Pre2)</td>
</tr>
<tr>
<td></td>
<td>$STOP()</td>
<td>!STOP([Pre1], [Direction], Landmark1, Pre2)</td>
</tr>
<tr>
<td></td>
<td>$START()</td>
<td>!START([Pre1], [Direction], Landmark1, Pre2)</td>
</tr>
</tbody>
</table>

### Table 2. A FRI route description from railway station to town hall.

<table>
<thead>
<tr>
<th>Block Number</th>
<th>FRI Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>START(RailwayStation, left)</td>
</tr>
<tr>
<td>2</td>
<td>$GO (to, CrossRoads)</td>
</tr>
<tr>
<td>3</td>
<td>$GO(forward, into, Street)</td>
</tr>
<tr>
<td></td>
<td>$PASS(Building, left)</td>
</tr>
<tr>
<td></td>
<td>$BE(at, CrossRoads)</td>
</tr>
<tr>
<td>4</td>
<td>$GO (left, into, Street, to, CrossRoads)</td>
</tr>
<tr>
<td>5</td>
<td>$GO(forward, into, Street)</td>
</tr>
<tr>
<td></td>
<td>$PASS(BurgerKing, right)</td>
</tr>
<tr>
<td></td>
<td>$PASS(C-and-A, left)</td>
</tr>
<tr>
<td></td>
<td>$BE(at, CrossRoads)</td>
</tr>
<tr>
<td>6</td>
<td>$GO(forward, into, Street)</td>
</tr>
<tr>
<td></td>
<td>$PASS(Church, left)</td>
</tr>
<tr>
<td></td>
<td>$PASS(KarStadt, right)</td>
</tr>
<tr>
<td></td>
<td>$BE(at, CrossRoads)</td>
</tr>
<tr>
<td></td>
<td>$TURN(right)</td>
</tr>
<tr>
<td>7</td>
<td>$GO(forward, into, Street)</td>
</tr>
<tr>
<td></td>
<td>$STOP(left, TownHall)</td>
</tr>
</tbody>
</table>

### Table 3. CRIL Presentations for natural language verbs particularly relevant in route instructions.

<table>
<thead>
<tr>
<th>Verb Type</th>
<th>FRI Commands</th>
<th>CRIL Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>$START()</td>
<td>!BE_AT(x, p)</td>
</tr>
<tr>
<td></td>
<td>$BE()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$STOP()</td>
<td></td>
</tr>
<tr>
<td>Locomotion</td>
<td>$GO()</td>
<td>!GO(x, w)</td>
</tr>
<tr>
<td></td>
<td>$CROSS()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$FOLLOW()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$PASS()</td>
<td></td>
</tr>
<tr>
<td>Notifying</td>
<td>!VIEW(x, r)</td>
<td></td>
</tr>
<tr>
<td>Change of orientation</td>
<td>$ROTATE()</td>
<td>!CH_ORIENT(x, d)</td>
</tr>
</tbody>
</table>
### Table 4. CRIL implementation of FRIs route

<table>
<thead>
<tr>
<th>FRI Statement</th>
<th>Actions</th>
<th>Spatial Relations</th>
<th>Landmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>$START(RailwayStation, left)</td>
<td>!BE_AT(x,p1)</td>
<td>LOC(p1, LEFT(LM1, rsys1))</td>
<td>LM1(RailwayStation, Shape)</td>
</tr>
<tr>
<td>$GO (to, CrossRoads)</td>
<td>!GO(x,w1)</td>
<td>LOC(w1, TO(LM2, rsys2))</td>
<td>LM2(CrossRoads, Color)</td>
</tr>
<tr>
<td>$GO(forward, into, Street)</td>
<td>!GO(x,d1)</td>
<td>LOC(d1, FORWARD(rsys3))</td>
<td></td>
</tr>
<tr>
<td>SPASS(Building, left)</td>
<td>!VIEW(x,p2)</td>
<td>LOC(p2, LEFT(LM3, rsys4))</td>
<td>LM3(Building, Texture)</td>
</tr>
<tr>
<td>$BE(at, CrossRoads)</td>
<td>!BE_AT(x,w2)</td>
<td>LOC(w2, AT(LM4, rsys5))</td>
<td>LM4(CrossRoads, Color)</td>
</tr>
<tr>
<td>$GO (left, into, Street, to, CrossRoads)</td>
<td>!GO(x,w3)</td>
<td>TO(w3, LEFT(LM5, rsys6))</td>
<td>LM5(CrossRoads, Color)</td>
</tr>
<tr>
<td>$TURN(right)</td>
<td>!CH_ORIENT(x, d2)</td>
<td>TO(d2, RIGHT(rsys7))</td>
<td></td>
</tr>
<tr>
<td>$GO(forward, into, Street)</td>
<td>!GO(x, d3)</td>
<td>LOC(d3, FORWARD(rsys8))</td>
<td></td>
</tr>
<tr>
<td>SPASS(BurgerKing, right)</td>
<td>!VIEW(x, p3)</td>
<td>LOC(p3, RIGHT(LM6, rsys9))</td>
<td>LM6(BurgerKing, Symbol)</td>
</tr>
<tr>
<td>SPASS(C-and-A, left)</td>
<td>!VIEW(x, p4)</td>
<td>LOC(p4, LEFT(LM7, rsys10))</td>
<td>LM7(C-and-A, Symbol)</td>
</tr>
<tr>
<td>$BE(at, CrossRoads)</td>
<td>!BE_AT(x,w4)</td>
<td>LOC(w4, AT(LM8, rsys11))</td>
<td>LM8(CrossRoads, Color)</td>
</tr>
<tr>
<td>$GO(forward, into, Street)</td>
<td>!GO(x, d4)</td>
<td>LOC(d4, FORWARD(rsys12))</td>
<td></td>
</tr>
<tr>
<td>SPASS(Church, left)</td>
<td>!VIEW(x, p5)</td>
<td>LOC(p5, LEFT(LM9, rsys13))</td>
<td>LM9(Church, Shape)</td>
</tr>
<tr>
<td>SPASS(KarStadt, right)</td>
<td>!VIEW(x, p6)</td>
<td>LOC(p6, RIGHT(LM10, rsys14))</td>
<td>LM10(KarStadt, Symbol)</td>
</tr>
<tr>
<td>$BE(at, CrossRoads)</td>
<td>!BE_AT(x,p7)</td>
<td>LOC(p7, AT(LM11, rsys15))</td>
<td>LM11(CrossRoads, Color)</td>
</tr>
<tr>
<td>$TURN(right)</td>
<td>!CH_ORIENT(x, d5)</td>
<td>TO(d5, RIGHT(rsys16))</td>
<td></td>
</tr>
<tr>
<td>$GO(forward, into, Street)</td>
<td>!GO(x, d6)</td>
<td>LOC(d6, FORWARD(rsys17))</td>
<td></td>
</tr>
<tr>
<td>$STOP(left, TownHall)</td>
<td>!BE_AT(x, p8)</td>
<td>LOC(p8, LEFT(LM12, rsys18))</td>
<td>LM12(TownHall, Shape)</td>
</tr>
</tbody>
</table>

#### 3.2.2 Spatial Relations

Spatial prepositions and adverbs are considered as the second kind of expressions of the spatial structure which identifies the spatial relationships. They can be expressed in CRIL by using LOC(w|p, Pre(LM)) syntax, where rsys is a spatial reference system (cf. [9], [16]) or by using the PRE(w ,Pre(LM)) syntax.

On the other hand, the spatial or directional prepositions can be divided into four classes [15]. The first class is the goal prepositions group which specifies the end of the path. The second one is the source prepositions group which gives the start of the path, the third class is the course prepositions group which characterizes the intermediate course of the path, and the final class is the shape prepositions group which identifies the shape of the path.

#### 3.2.3 Landmarks features

The third information item that can be extracted from the route description is the landmarks. Landmarks are chosen to indicate the necessity of changing direction or simply confirm correctness of former decisions about the route. The landmarks in our miniature city are identified by using the following four types:

- **Shape:** The landmark can be recognized by the shape, if its shape has different features and characteristics which can be easily noticed from the other surroundings. These features like the style of the building, contrast, and landmark figure. The railway stations, Town Hall, and churches can be considered as examples of this type.

- **Symbol:** It is used if the building has a unique symbol or trademark. These symbols are used to recognize the landmarks in the city during the navigation process. Supermarkets, restaurants, and shops can be recognized by using their symbols.

- **Texture:** Other landmarks can be recognized by their visual properties such as its texture. These landmarks can be recognized if their textures are symmetric and different from the surrounding objects. Houses and buildings can be identified by their textures.

- **Color:** The landmark color is one of the valuable features which can be used to identify the landmark. We recognize the crossroads in our miniature city by using dotted white lines. The street’s boundaries are indentified by using black lines.
Figure 3. Topological Map presentation of the route (Railway Station to Town Hall).

The landmark is represented in the INHR-adaption of CRIL by using the following syntax:

\[ LM_i(\text{Name}, \text{type}) \]

where \( i \) presents the landmark number in the route. The Name represents the landmark name, The type presents the way that used to recognize the landmark. The landmark’s features are store in the database and they are retrieved and processed by the CRIL stage.

The CRIL representation of route description from the railway station to the town hall is shown in Table 4. Each FRI is converted to its equivalent CRIL action, spatial relationship, and landmark.

### 3.3 Topological Map

As discussed previously, the topological map is a graph representation of the environment, where nodes correspond to distinct places that can be recognized by robot. It is an abstract representation of the spatial knowledge that describes relationships among features of the environment, without using any absolute reference system. We use the actions, spatial relations, and the landmark features which are extracted from the CRIL representation to build the topological map. Figure 3 shows the topological map of the route from the railway station to the town hall in our miniature city.

This map consists of seven nodes. The distance between two nodes presents the execution of CRIL actions which result from one processing unit of the FRI statements. The locomotion actions are introduced as solid arrows, the notation actions are presented as dotted arrows pointing to the landmark. The position actions are drawn as node. Finally, the change-orientation actions are represented as a written direction beside the node.

Landmarks are introduced as rounded corner rectangles. These rectangles contain the landmark number, the landmark name, and the recognition type. The spatial relations are written above or below the action arrows.

### 4 CONCLUSION

In this paper, we presented a Formal Route Instruction Language (FRIs), which can be used by naïve users to formulate route instructions for robots. It is intended to guide humanoid robots while it navigates through the miniature city. By using FRI route directions misunderstanding and ambiguities can be avoided. On the other hand, the Conceptual Route Instruction Language (CRIL) was customized to extract the main motion actions of humanoids, the spatial relations, and the landmarks used for instructing humanoids. Finally, we represented the data extracted from the CRIL as a topological map. This map gives a graphical depiction of the proposed humanoid’s actions and spatial relations. It is also connects the route’s landmarks as a network of landmarks to let the humanoid take an overview of the navigation task.

### REFERENCES


Towards an Episodic Memory for Cognitive Robots

Sascha Jockel\textsuperscript{1} and Martin Weser\textsuperscript{1} and Daniel Westhoff\textsuperscript{2} and Jianwei Zhang\textsuperscript{3}

Abstract. Different disciplines such as psychology and neuroscience have been examining episodic memory (also referred to as declarative memory) for more than three decades. Now, engineering and computer science are developing an increasing interest in episodic memory for artificial systems. We propose a novel framework EPIROME to develop and investigate high-level episodic memory mechanisms which can be used to model and compare episodic memories of high-level events for technical systems. We demonstrate how we applied the framework to the domain of service robotics. High-level events emanate from basic skills, elementary operations, sequences of elementary operations, environmental changes and the detection of human interactions. The framework enables our service robot TASER to collect autobiographical memories to improve action planning based on past experiences. The framework provides the robot with a life-long memory since past experiences can be stored and reloaded. In practise, one main advantage of our episodic memory is that it provides one-shot learning capabilities to our robot. This reduces the disadvantage of other learning strategies where learning takes too long when used with a real robot system in natural environments and therefore is not feasible.

1 Introduction

Memory is central to the human condition and has been investigated at many levels. Neuroscientists have studied the molecular and cellular mechanisms of memory in animals and humans, and psychologists have contributed to our understanding about the different kinds of processes involved in memory through research with amnesic patients and normal subjects. Engelkamp [1] propose to distinguish memory systems based on the type of stored information (e.g. episodic-semantic, verbal-nonverbal-imaginal), the type of processes involved (e.g. declarative-procedural, implicit-explicit) and such memory systems based on the length of time that information is retained (e.g. short-term-long-term).

The study of episodic memory began in the early 1970s when the psychologist Endel Tulving made a first distinction between episodic and semantic memory [2]. At that time episodic memory (EM) was defined in terms of materials and tasks. Tulving specified episodic memory as your experiences of certain, spatio-temporal definite episodes (e.g. your last business trip) and our general knowledge (language translations, facts like “what is a pen” et cetera.) as the semantic memory (SM). However, his suggestion that episodic and semantic memory are two functionally different memory systems quickly became controversial. As a result of the criticism, the episodic memory definition was refined and elaborated in terms of its main ideas such as self, subjectively sensed time, and autonomic consciousness. Today, episodic memory is seen as one of the major neurocognitive memory systems [3] that is defined in terms of its special functions (what the system does or produces) and its properties (how it does that). It shares many features with semantic memory, which it grew out of, but it also possesses features that semantic memory does not have [4]. Episodic memory is oriented towards the past in a way in which no other kind of memory system is. It is the only memory system that allows people to consciously re-experience their past. It has a special and unique relationship with time [5]. Neuropsychology took up the idea of episodic memory and tried to find proofs for the concept in biological systems. Tests on amnesic patients (e.g. the famous hippocampal amnesic H.M. [6, 7]) suggested that the episodic memory is mainly related to the medial temporal lobe and hippocampal structures [8].

The brain uses vast amounts of memory to create a model of the world. Everything a person knows and has learned is stored in this model. The brain uses this memory-based model to make continuous predictions of future events [9]. If those predictions are disproved, the brain learns (e.g. by novelty detection [10]), and adjusts its memories according to the new data. The memory seems to be organised in a hierarchy, each level being responsible for learning a small part of the overall model. Kanerva [11] proposed a sparse distributed memory (SDM) model that offers many of the characteristics that a human memory possesses. He also developed a mathematical model for this theory.

Over the last decade an increasing interest in episodic memory mechanisms can be noticed in engineering and computer science. In Section 2 these research ambitions are discussed. However, first we must review the characteristics of episodic memory in humans that evolve from psychology and neuroscience. Because psychology assumes the automatic memory formation in humans to be an obligatory process, it is not listed as a special characteristic below:

1. Autonoetic: Remembering episodic memory is characterised by a state of awareness unlike that in semantic memory, that is noetic. When one recollects an event autochtonically, one re-experiences aspects of a past experience. Re-experiencing of an already learnt episode is not necessary.

2. Autobiographical: A person remembers an episode from his or her own perspective. There is no possibility to change the viewpoint in AI systems. To put oneself in someone else’s place is the highest achievement of human intelligence. Moreover, there are studies proving that autobiographical and episodic memory are separate memory systems [12].

3. Variable Duration: The time period that is spanned by an episode is not fixed.
4. **Temporally Indexed:** The rememberer has a sense of the time at which the remembered episode occurred.
5. **Imperfect:** Our memory is incomplete and can have errors. New sensations are forced to satisfy already experienced concepts.
6. **Primed:** Recall occurs more quickly when it is primed by repetition, recall of related information, or similar states.
7. **Forgetting:** It is still not clear if forgetting is a problem of actual information loss in long-term memory (LTM), or rather a problem of recall of the memory traces. Currently, mechanisms of active forgetting are being discussed [13].
8. **Level of Activation:** Exposure frequency and recency affect the speed and probability of recall. The level of activation mainly describes the primacy & recency effect where the former is based on LTM effects and the latter is based on the contents of the working memory.

This paper is structured as follows: After this brief introduction to episodic memory from the psychological and neuropsychological point of view, we present some related work in Section 2, particularly from the field of engineering and computer science. Section 3 describes the domain of our multimodal service robot TASER. Our novel EPIROME framework is introduced in Section 4. We conclude with an outline of our future work in Section 5 and give a general conclusion on our EPIROME framework and episodic memory in robotics in Section 6.

## 2 Literature Review – An Excerpt

Mechanisms of episodic memory can be used to develop new learning algorithms and experience-based prediction systems. Agents that do not remember their past are bound to repeat both the previous mistakes and the reasoning efforts behind them. Thus, using an episodic memory helps to save time by remembering solutions to previously encountered problems and by anticipating undesirable states. In literature several important approaches to creating episodic memory in artificial systems have been explored. Computational models of episodic memory can be divided into two categories: abstract an biological. Abstract models make claims about the “mental algorithms” that support recall and recognition judgments, without addressing how these algorithms might be implemented in the brain. Biological models make claims about the computation that support recall and recognition judgments, the main difference being that they also make specific claims about how the brain gives rise to these computations. While the former models account for challenging patterns of behavioural recall and recognition data from list learning paradigms, the brain-model mapping of the latter models provides an extra source of constraints on the model’s behaviour. Even if Norman, Detre & Polyn [14] outline a comprehensive overview on computational models for episodic memory, only few robotic systems exist that make use of such models for learning.

### 2.1 Biological Models

#### 2.1.1 Neural models

An episodic memory model using spiking neurons was presented in [15]. The author describes a model that meets requirements for real-world robotics applications. Requirements were: (a) learn quickly and on-line, (b) recall patterns in their original order and with preserved timing information and (c) complete sequences from any position even in the presence of ambiguous transitions upon cueing. The author proposes a two-layer feed-forward neural network architecture based on SM (spike accumulation and δ-modulation) neurons that are capable of categorising the continuous stream of sensorimotor patterns from a robotic system interacting with its environment. A learning-by-doing task was evaluated where the robot was taught to draw a circle by guiding its hand. By using a revised Hebbian temporal learning rule with synaptic history [15], the network took about 50 epochs to stabilise.

Regrettably, the network is very sensitive to noise and the range of recorded episodes is very small. In our point of view this approach is not considered as episodic memory rather than nondeclarative procedural memory according to the definition of LTM by [8].

#### 2.1.2 Novelty mediated autobiographical memory

Barakova & Lourens [10] focus their research on memory-determined behaviour that relies on the neural mechanisms underlying episode formation. They use the term episodic memory as including event information within its temporal relatedness and directionality. They propose a computational model inspired by the hippocampal system of rats that aims at novelty-driven encoding and recall that facilitates inferential reuse of old memories [10].

Three neural structures are used to form a representation that is further used for navigation. Two simultaneous active neural networks, corresponding to the Cornu Ammonis 1 & 3 areas (CA1 and CA3) perform the major computations. The neurons in the CA3 area account for the temporal aspect and the formation of episodes. The representation in the CA1 area is prone to detect novelty. The third structure, *entorhinal cortex* (EC) provides the input patterns to both areas by projecting it onto CA1 and CA3 within a short time interval. Events that have been learnt as an episode will tend to be recalled together and after each other, even if the presentation order is changed [16].

Although the proposed model is one of the most biologically inspired robotics implementations of emergent behaviours based on episodic memory encoding, it relies mainly on spatial navigation tasks.

#### 2.1.3 SMRITI

SMRITI is a computational model of episodic memory that illustrates the role of the hippocampal system in the acquisition, maintenance and retrieval of episodic memory, and proposes a detailed circuit-level explanation of how the hippocampal system realizes this function in concert with cortical representations. The model demonstrates how a cortically expressed transient pattern of rhythmic activity representing an event or a situation can be rapidly transformed into a persistent and robust memory trace as a result of long-term potentiation and long-term depression [17].

## 2 Abstract Models

### 2.2 Abstract Models

#### 2.2.1 MINERVA 2

MINERVA 2 [18] focus on schema abstraction, recognition, and frequency judgments and is best described as an existence proof: MINERVA 2 proves that it is possible to account for many aspects of memory for individual experiences (i.e., episodic memory) and memory for abstract concepts (i.e., generic or semantic memory) within a single system. MINERVA 2 does not prove that there is only a single system; rather, it proves it can be done.
In MINERVA 2, an item is represented as a vector of features in which each component is represented by the numbers 1, 0, or -1. Memory for an episode (e.g., for learning of a list of words) is a set of encoded vectors, with each event (word) being represented in a separate memory vector. The global similarity of a test item to a memory trace is determined by the sum of the product of each feature in a probe vector and the feature in the corresponding position in the memory trace vector divided by the number of features for which either the probe or the memory trace are nonzero. Accordingly MINERVA 2 has to compare a test item to all items in memory.

In this model each item will be stored in an continually growing matrix of memory traces. Together with the implausible presumption that a biological memory increases linearly, prototype and exemplar stimuli theory are not capable to model and explain human sensitivity to changing frequency of occurrence and the influence of the sample size [19]. Adaptability to robotics seemed to be questionable.

2.2.2 An Episodic-memory approach to the problem of pattern capture and recognition

Tecuci et al. [20] simply outline the following characteristics as requirements for episodic memory: The memory organises temporally ordered events, that are dynamic (i.e., they change the status of the world) and are observed incrementally. Capture and recognition of past events are the basic processes of an episodic memory [20]. An episode is defined as a sequence of actions with a common goal. Their main goal is to achieve a retrieval algorithm that can deal with incrementally available data to make predictions dynamically in a fast and accurate manner. They evaluate their approach on a goal schema recognition task in the Linux Plan Corpus. The task was to predict the type of goal an agent (Linux user) has without exact parameters. Linux users were given a goal (e.g. find a file with “exe” extension) and were instructed to achieve it using simple Linux commands.

Even if they proved that memory retrieval is scalable, they achieved only the same level of performance as statistical approaches. Unfortunately, the system is not able to recognise subgoals of long period plans and is sensitive to noise. A benefit of the system is the reduction of search space by only storing relevant episodes.

2.2.3 SOAR-EM

Nuxoll & Laird extend the CBR paradigm by integrating episodic memory with a general cognitive architecture and developing task independent mechanisms for encoding, storing, and retrieving episodes [21]. They extend SOAR, one of the major cognitive architectures based on production rules [22]. SOAR has two types of knowledge, working memory (short-term, declarative) and production rules (long-term, procedural) and has been extended with episodic memory mechanisms into SOAR-EM.

In previous articles they propose a Pacman-like domain to wander around in a limited grid and collect the most food-points in the least amount of time. Their goal was for the agent to use its episodic memory in place of its knowledge about the food-points to aid in selecting the direction in which it should move. An activation-based matching scheme leads to significantly better results than its unbiased match predecessor that was developed earlier. As the agent acquires more memory items, the eater’s performance continues to improve until it performs at a level comparable to the greedy eater (that only heads to the best food in its direct neighbourhood) [23]. The hypotheses of cognitive capabilities resulting from this episodic memory are discussed and confirmed by implementations in their latest article [21].

2.2.4 LIDA

The Learning IDA (LIDA) architecture incorporates six major artificial intelligence software technologies: the copycat architecture, sparse distributed memory, pandemonium theory, the schema mechanism, the behavior net model, and the sub-sumption architecture [24]. LIDA is an extension for the Intelligent Distribution Agent (IDA) — which is a referred to as “conscious” software agent — by perceptual-, episodic-, and procedural-learning capabilities. It was created as model of human cognition that could be used to suggest possible answers to questions about the human mind. The authors designed and developed a practical application that could act like a human detailer, a person who negotiates with sailors about new jobs who are near the end of their current tours of duty.

A percept in the LIDA architecture can be thought of as a set of elements of an ontology that are relevant to the stimulus. They organise this information into a binary vector, where each field of one or more bits represents an element of the ontology [24]. A cue (the binary vector) will be used to query the content-addressable memories, autobiographical memory (ABM) and transient episodic memory (TEM). Both are based closely on Kanerva’s sparse distributed memory (SDM) [25], as already mentioned in the Sec. 1. A similarity matrix is used to map the system to a specific cognitive system. The retrieved episodes are specified within an ontology ex ante. It is limited to the domain of providing new jobs to sailors.

2.2.5 Memory retrieval through emotional salience and statistical information

Episodic memory retrieval driven by an emotional system of a humanoid robot is realised in [26]. A single episode is defined as a period of task execution of the robot during which the goal of the robot does not change. The retrieval of episodes is accomplished through an algorithm that takes the current episode and selects several stored episodes for placement in the episodic memory-working memory set. The probability that a memory is relevant is calculated through a combination of two independent factors: a history component and a contextual component [27]. The retrieved episodes are used to generate future actions through a planning system. To represent and evaluate emotions they used Haikonen’s system reactions theory of emotions (SRTE) as described in [28]. In their cognitive control experiment, the Agent ISAC (Intelligent SoftArm Control) has to follow a moving object with its cameras. When a person yells “Fire!”, ISAC uses attention, emotion and cognitive control to suspend the current tracking task and warns everyone to exit the room [29]. Unfortunately, ISAC recognises only four objects and four people in its semantic memory [26]. For the purposes of this experiment, episodes that were designed to use a variety of semantic memory units were hand-crafted. Tasks that could be solved cover subjects like placing objects in a certain configuration, greeting humans, and identifying objects.

Finally, it should be noted that the review of related work shows that engineering and computer science are in the early stages of episodic memory modelling. The aforementioned approaches should gave the
reader an overview to current implementations of abstract and biological models of episodic memories in technical systems. For the majority the portability to the domain of robotics appears to be quite problematic. The presented approaches to build an episodic memory have the following problems in common:

- Only applicable in highly limited domains,
- inappropriate for realising higher psychological functionality of episodic memory,
- only consider actions, no perceptual and executive information,
- mostly handle short sequences,
- do not use one-shot learning,
- exhibit gap of terminology of episodic memory among different disciplines.

In addition to these problems, neuroscience revealed considerable evidence that attentional resources are necessary for the encoding of episodic memories, while the nature of the relationship between attention and neural correlates of encoding is unclear [30]. Especially a middle layer between encoding of sensoric stimuli from robot sensors to biological paradigms and high level learning, reasoning and prediction techniques are still missing to move the field of biological plausible computing. The findings offered by neuroscience research should be taken seriously and greater concentration should be given to dynamic network architectures that can alter their structure based on experience. Finally, a more comprehensive understanding of the brain and the central nervous system is critical to achieving better and biologically inspired adaptive computing systems.

3 The Multimodal Service Robot Platform

Since our research background belongs to the field of service robotics we developed a framework to investigate the use of episodic memory in real robot systems. The domain of service robotics is to assist human beings by performing jobs that involve long distances, are auxiliary, dangerous or repetitive et cetera.

Westhoff et al. mention a novel concept for distributed programming of our multi-modal service robot TASER shown in Fig. 1 [31]. Furthermore, they describe numerous practical experiments carried out on the robot. TASER has to work in a dynamic, real-world office environment and due to its mobility every execution of a task slightly changes. TASER is operated by a built-in control system, driven by one Pentium IV 2.4 GHz standard computer. Due to TASER’s evolution novel tasks may occur that can be solved in analogy with already learnt tasks. Thus, improved memory systems to remember previously encountered problems and to anticipate undesirable states are essential. Generalised memories of sequences consisting of action-based, perceptual and executive information can be applied to solving novel problems.

One such task as mentioned above can be described as follows. A high-level service robotic task is e.g. to “serve drinks to guests”.

For this task TASER has to pick up some drinks and glasses in the kitchen. Since the robot does not know where the guests are, it has to walk around to find them. If TASER finds somebody it has to evaluate if the person is an employee or a guest/foreigner e.g. by comparing known and unknown faces via a face detector. If a stranger/guest is detected, TASER offers a drink to him/her. Since only a couple of rooms are of special interest for guests on a visit (e.g. laboratory, climbing robot room, hallway, et cetera), an episodic memory of past, similar experiences will help TASER to realise that it has to search for strangers most frequently in these few rooms. This ensures that the quality and speed of TASER’s service is improved.

High-level tasks may have similar subsequences. If the robot should bring something from the elevator to the workshop or vice versa the subsequence (e.g. “call for the elevator”, “enter elevator”, “pick up object” or “walk along hallway”, “enter workshop”, “place next to worktop”, “put down object on worktop”) may be similarly independent of the object. These sequences can be generalised. In the case of “empty all waste bins” the task vary among concrete executions caused by a dynamic environment and depending on which room may be locked or which waste bin may be unreachable or hidden. These examples show that generation of action sequences can be omitted if subsequences remain reasonably stable during different tasks. Memory retrieval can be used as heuristics to continue or stop the execution of a task. If the goal in a memory-based, predicted sequence is constantly not reachable the robot should either be forced to use other approaches or to combine several subsequences of other less related memory traces to reach the goal. This can be seen as task decomposition.

4 The EPIROME Framework Design

The EPIROME framework offers the capability to record high-level episodic memories as mentioned in Section 1, 2 and exemplified in Section 3. EPIROME is an independent framework that is based on the observer design pattern. The observer pattern is a design pattern for observing the state of an object in a program. It is mainly used to implement a distributed event handling system. The essence of the pattern is that one or more objects (called observers or listeners) are registered (or register themselves) to observe an event which may be caused by the observed object (the subject). A domain-independent abstract layer specifies the interface for event broadcast. Observers can be attached to concrete subjects by using a connection method. Observers have to implement a method newEvent() offered by the abstract interface layer to specify how to process information on a connected subject if a new event occurs. Each subject has a list of observers listening to it. Each time a concrete subject perceives a specific event through the manifold sensors of the robot system, it will call newEvent() for all observers in its observer collection.
In general we think of episodic memories as sequences of events. Each event carries time information and can be assigned to one of the three major event classes: perceptual events, command events and executive events (Fig. 2). Based on this domain-independent hierarchy, a more comprehensive classification can be applied to include domain-specific information. Fig. 2 shows a part of the event hierarchy realised for our service robot TASER. The processing within the memory module of our framework will only work for the domain-independent classification of events. Therefore, the framework can be applied to other domains seamlessly. In the following subsections we give a brief introduction on how we associate robotic events to the above-mentioned major event classes.

### 4.1 Perceptual events

This type of events focus on the recognition and interpretation of sensory stimuli. Perception obviously applies to all sensoric modalities. Robot sensor input gets a high-level interpretation e.g. by mapping to semantic knowledge provided by a regionally labelled site map, basic physics rules et cetera. The following examples illustrate how high-level events emerge from low-level sensor data:

- **Spatial perceptual:** In this architecture we assume an inference component is capable of inferring robots position relative to landmarks in a given semantically, regionally labelled site map. Everytime the robot changes from one region to another it will receive a spatial event as e.g. “I am in front of the laboratory”. Based on this map and the localisation the robot has a sense where it is (e.g. laboratory, kitchen, office, et cetera).
- **Stalled:** If the robot e.g. passes down a planned path and encounters a meanwhile closed door or an obstacle blocking its way, it will receive an event signalling that it cannot pass through. In addition, manipulation events can also perceive if they get stuck.
- **Uni-/Multimodal perceptual:** Perceptions can be based on a single sensor or can be combinations of different sensory information. This information is useful for investigations to multimodal cognitive processing.

### 4.2 Command events

Command events are specifications of the ability to produce movements by interaction of a control unit and actuators. In case of TASER the control unit is the motor control system and the actuators are the servomotors. Command events can be seen as basic movement functionalities, typically: Rotate, Translate, Stop, Open finger et cetera.

### 4.3 Executive events

This category contains high-level abilities and meta events that are mostly reflective or procedural. They are necessary for goal-directed behaviour. Thus they are called executive functions.

- **Goal:** An agent strives for a particular goal, this can be a high-level main task (e.g. “bring me a coffee from the kitchen”, “carry this object to person X”, “empty all waste bins”) or a collection of subtasks (e.g. “go to kitchen”, “grasp waste bins”). This may result from direct user interaction.
- **Ends:** The agent confirms if a task/subtask is reached successfully or not. This also can be a result from feedback of a user. However, it also records if a goal is not reached to avoid the same mistakes next time.
- **Planning:** If the agent makes plans to solve tasks and desires (e.g. recharge, path planning, trajectory planning)
- **Manipulate/Grasp:** The ability to manipulate/grasp objects with e.g. a robot arm, a hand.
- **Handle:** A more complex combination of object grasping, manipulation, translation, et cetera as combination of few basic command events.

The type of used tasks and events (e.g. spatio-temporal events like “I am in the laboratory next to a desk”) demonstrates that our system operates on a high cognitive level. The current EPIROME framework for our robot domain possesses several observers and subjects. The user can choose between several basic command events to solve a desired task (Fig. 3). It has a visualisation tool to present experienced and currently running episodes to the user 4. Each event of an episode gets a dedicated colour depending on the event type (Fig. 3). The colour coding makes it easier to make a first comparison of episodes visually. The map in Fig. 3 shows a hallway and the current position and sensing of the robot. The door application to the right of Fig. 3 is only used during simulation. Doors can randomly open or close to force the robot to stall and rearrange its path. This leads to considerable changes within episodes of the same task. The three episodes in the middle of the visualisation tool are of the same task and look similar. However, the first episode differs from the other two after few events and is longer although the robot stalled due to a closed door. Nevertheless TASER solved its task successfully by replanning.

---

4 A recorded sequence may look as follows (timecodes are not listed): GOALEVENT: Bring newly delivered material from the elevator to the workshop. Bucket lifted. Path found. Rotate to north north east. Translating to position (x: 14303 mm; y: 7765 mm; theta: 45.117) Floor in front of lab. Rotate to east. Translating to position (x: 23646 mm; y: 8245 mm; theta: 0.0) Floor in front of office 2. Rotate to front of office 3. Floor in front of office 4. Rotate to north east. Translating to position (x: 26121 mm; y: 14523 mm; theta: 167.354) Workshop. Put down bucket. Bucket released. Goal successfully accomplished. The majority of events can be deconvolved from complex to atomic actions.
5 Current Work

The basic functionality of our framework for high-level episodic memory research in robotics has been completed. We already developed the EPIROME framework to collect high-level events and developed mechanisms to maintain such high-level events. These events are based on different modules that provide particular high-level information, e.g. the path planner provides information like "move straight on for distance $Y$", "turn right", "turn hard left", et cetera; events might be given apriori by user interaction and commands; an open or closed door can be sensed through the laser-range scanners in combination with a site map; high-level arm movements like beckoning, opening a door, grasping an object on a table, shaking hands has been learned via a content-addressable SDM and can be recognised.

Our current work is twofold. First we extend our system with further event generating modules. These components can be categorised to extend the already used perceptual, command and executive events. Thus, it will satisfy the current design. Additional event generating modules can be e.g. face detectors, manipulation units, object recogniser and locator et cetera. Additionally we are going to verify the episodic memory approach to the problem of pattern capture and recognition through a goal schema recognition task in the Linux Plan Corpus proposed in [20] and compare it to our episodic memory module. Our framework can easily be extended and tested by designing a module with the capabilities proposed by [20].

And secondly, we are transferring and extending the SDM mechanisms that are already used for our manipulator unit to a more generalised memory that combines further robot sensors and actuators. If the robot solves tasks in a new and distinct manner, it will store a new memory trace immediately. Since the mathematical model of SDM follows the basic idea that the distances between concepts in human minds correspond to the distances between points of a high-dimensional space, we expect to get clusters of similar actions and action sequence concepts. Since our manipulator has proved to work well with an SDM, the main problem remains of how to convey the different types of features (based on the sensors) into the high-dimensional input vector required by the SDM. This encoding problem remains to be the major problem in associative-memories of which episodic memory and sparse distributed memory are parts.

At a later step, frequently emerging subsequences within episodes can be generalised to a single abstract meta event. Consequently, individual subsequences are not stored redundantly and the generalised abstract event refers to the memory trace of the experience related to this subsequence. This is consistent with the theory that related concepts are stored close together. The level of activation of a trace of a subsequence has to be primed depending on its occurrence.

Since we can easily add additional concrete observers to the EPIROME architecture it will be a cinch to compare different memory mechanisms implemented by different modules, listening to the same type of events.

6 Conclusion

After three decades of psychological and neuropsychological research, episodic memory is finding its way into engineering and computer science. In this paper we gave an overview to current computational models of episodic memories and outline the problems of portability into robotics. Even for neuroscience the relationship between attention and neural correlates of encoding in episodic memory remains unclear. A better understanding of the encoding of sensoric stimuli from robot sensors to biological paradigms and high-level learning, reasoning and prediction techniques is necessary. We propose that findings offered by neuroscience and psychological research should be taken seriously to get a more comprehensive under-

---

5 In addition the SDM stores a model of the world (concerning the manipulator) while the sensory information at a particular moment is represented as a vector of features (joint angles, tool-center-point & orientation) and a sequence of such vectors represent the passage of time. This has prediction capabilities since after short training the SDM returns predictions of some next arm positions that were normally activated from the current position.

6 We are aware that we have to take temporal indexing of each episode into account if we use generalisation.
standing of the brain and the central nervous system, especially for advanced biologically inspired robotics. Close interdisciplinary work will be indispensable in reaching this major goal.

With our EPIROME framework it is possible to model and compare episodic memories of high-level events for technical systems. We apply this framework to our robot system and it provides TA
er with a life-long memory to improve action planning based on past experiences. Table 1 shows the characteristics of episodic memory that EPIROME already complies in comparison to the psychological and neuropsychological characteristics mentioned in Tab. 1. Unfulfilled conditions are marked with □ in Tab. 1 and have to be discussed before implementing, e.g. it is eligible to realise forgetting as long as no physical storage or processing limitation exist.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>EPIROME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autonoetic</td>
<td>□</td>
</tr>
<tr>
<td>Timely</td>
<td></td>
</tr>
<tr>
<td>distinguishable</td>
<td></td>
</tr>
<tr>
<td>Sensations</td>
<td></td>
</tr>
<tr>
<td>Autobiographical</td>
<td>□</td>
</tr>
<tr>
<td>Robots</td>
<td></td>
</tr>
<tr>
<td>own</td>
<td></td>
</tr>
<tr>
<td>sensations</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>□</td>
</tr>
<tr>
<td>Duration</td>
<td></td>
</tr>
<tr>
<td>Episodes</td>
<td></td>
</tr>
<tr>
<td>bound</td>
<td></td>
</tr>
<tr>
<td>by</td>
<td></td>
</tr>
<tr>
<td>start</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>Event</td>
<td></td>
</tr>
<tr>
<td>Temporally</td>
<td>□</td>
</tr>
<tr>
<td>Indexed</td>
<td></td>
</tr>
<tr>
<td>Timestamps</td>
<td></td>
</tr>
<tr>
<td>Imperfect</td>
<td>□</td>
</tr>
<tr>
<td>Error</td>
<td></td>
</tr>
<tr>
<td>simulation</td>
<td></td>
</tr>
<tr>
<td>/ Sensor</td>
<td></td>
</tr>
<tr>
<td>uncertainty</td>
<td></td>
</tr>
<tr>
<td>Primed</td>
<td>□</td>
</tr>
<tr>
<td>Framing</td>
<td></td>
</tr>
<tr>
<td>of frequently</td>
<td></td>
</tr>
<tr>
<td>used</td>
<td></td>
</tr>
<tr>
<td>episodes</td>
<td></td>
</tr>
<tr>
<td>Forgetting</td>
<td>□</td>
</tr>
<tr>
<td>Deletion</td>
<td></td>
</tr>
<tr>
<td>of less used</td>
<td></td>
</tr>
<tr>
<td>sequences</td>
<td></td>
</tr>
<tr>
<td>Physical</td>
<td>□</td>
</tr>
<tr>
<td>limited</td>
<td></td>
</tr>
<tr>
<td>storage</td>
<td></td>
</tr>
<tr>
<td>devices</td>
<td></td>
</tr>
<tr>
<td>Level of</td>
<td>□</td>
</tr>
<tr>
<td>Activation</td>
<td></td>
</tr>
<tr>
<td>Probabilities</td>
<td></td>
</tr>
<tr>
<td>in retrieval</td>
<td></td>
</tr>
<tr>
<td>function</td>
<td></td>
</tr>
</tbody>
</table>

| Table 1. Characteristics of episodic memory of the current EPIROME framework (□) and future work (□). |

One major advantage of our episodic memory is that it provides one-shot learning capabilities to our robot. This is very important while it learns novel tasks. Furthermore, by using an SDM, less used executive manipulation tasks will be forgotten. We are not simply processing motor information but also high-level sensory events which are mostly neglected or of low-level in other artificial systems. If an episode based on current sensings does not reach a goal, already experienced episodes or subsequences of past episodes may provide approaches to achieving success. The system structures high-level tasks into a well-composed hierarchy. The EPIROME framework is a sophisticated model to establish human-like learning in AI robot systems and for investigating multimodal cognitive processes.

Acknowledgment

We wish to thank the German Research Foundation DFG for funding this project within the context of CINACS international researc

training group by grant #1247.

REFERENCES

demic Press.
[10] Emilia I. Barakova and Tino Loudens. Spatial navigation based on nov
tional Models of Episodic Memory. August 2006.
[17] Lokendra Shastri. Episodic memory and cortico-hippocampal interac
senschaft (KogWis), pages 262–267, Sankt Augustin, 1999.
[21] Andrew M. Nixoll and John E. Laird. Extending cognitive architectur
A Cognitive Theory of Autodidactic Agency

Loizos Michael

Abstract. We present a summary of our work in developing a unified formal theory for the processes of learning and reasoning. We consider the case where an environment of interest is only passively observed, and the goal is that of acquiring knowledge about the environment that is provably predictive in a defined sense. This theory we propose as the basis for a cognitive architecture of autodidactic agents, agents that efficiently and completely autonomously induce and reason with knowledge about their environment.

Central in our theory is the sensing process, by which information about the state of an agent’s environment is obtained. We take the approach that sensor readings provide an agent only with partial information about its environment. From such observations the agent seeks through learning to identify the underlying reality of its environment. Knowledge so induced is subsequently employed to recover information about the environment that is not explicitly observed. We show that learning and reasoning are conceptually and computationally feasible, despite the demanding requirements of autodidactism.

1 Introduction

The development of formal theories of cognition is arguably an important step in the understanding of intelligence and its eventual replication by machines. Among the cognitive phenomena studied within Computer Science to date, emphasis has been placed on those of learning and reasoning, the processes by which knowledge about a domain of interest is, respectively, acquired and manipulated. With few exceptions [11, 15], these two processes have been studied independently, without considerations for the development of a unified framework. Even in those cases that such a unified framework has been considered, emphasis has been placed either on its empirical aspects, with little concern for the development of a principled computational theory [11], or on the investigation of learning and reasoning about the state of knowledge of an agent, as opposed to learning and reasoning about the state of the agent’s environment [15].

In this paper we present a novel unified view of learning and reasoning in terms of an information-recovery task [10]. We consider domains where an agent’s sensors provide the agent with some incomplete description of the state of its environment. Based on this incomplete information the agent is expected to decide on its actions. Yet, the appropriateness of any action taken is ultimately determined by the underlying real state of the environment. We argue, then, that the agent should strive to recover as much of the missing information as possible, so as to reconcile the reality of its environment with the appearances of that reality that the agent’s sensors provide it with.

We suggest learning as a means for an agent to discover the structure of its environment, which can be subsequently employed for the information-recovery task. This information-recovery task closely resembles that of scientific discovery, as exemplified by the development of physical theories by humans, which are subsequently employed to make predictions in novel, and previously unseen situations. As in the case of physical theories, an agent’s knowledge is expected to recover missing information in an accurate manner, despite the fact that this knowledge is induced from and applied on the partial information that is available to the agent. In this respect, then, our focus in this paper is on the investigation of whether agents can undertake this scientific discovery task in an autodidactic, or self-taught manner, where no teacher is involved in the learning process.

We believe that the development of such cognitive theories of autodidactic knowledge acquisition and manipulation is essential both philosophically as a means of understanding certain aspects of human intelligence, but also as a guide in the development of intelligent agents, prescribing, in particular, how to alleviate the (often infeasible) task of programming such agents with commonsense knowledge about their environment. Although not reported here, we remark that the theory presented in this paper has been successfully employed to extract such commonsense knowledge at a massive scale and completely autonomously from a natural language text corpus [10].

Our exposition starts with a model of the sensing process, upon which the theory of learning and reasoning builds. We restrict our investigation to the case of agents that only passively observe their environment, without taking actions that affect the type of information that they obtain through their sensors. Our setting is not, therefore, related to that of reinforcement learning. This passive acquisition of information makes it necessary to examine learnability without making any assumptions on the type of missing information. It is shown that in certain domains learning the structure of the environment is possible under arbitrary hiding of information, making progress towards a challenge posed by McCarthy [8].

We then discuss how a set of induced rules should be applied, and propose an appropriate reasoning semantics. We establish that reasoning in multiple layers is provably beneficial. Although multi-layered reasoning has been often examined in the past and shown empirically to be beneficial, our result appears to be the first formal separation of single-layered reasoning from multi-layered reasoning in the context of passive learnability with predictive guarantees.

The question of how learning sets of rules should proceed given that the rules are to be reasoned with, is finally considered. We argue that rules should be learned iteratively, and show that such an iterative learning strategy also suffices for learnability to be ensured.

2 Appearance and Reality Dichotomy

[...] human learning from experience often involves learning about the hidden reality behind some phenomenon. [...] Machine learning research, to my knowledge, has so far involved

---

1 This work was supported by grant NSF-CCF-04-27129.
2 School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138, U.S.A. E-mail: loizos@eecs.harvard.edu
classifying appearances, and has not involved inferring reality behind experience. Classifying experience is inadequate as a model of human learning and also inadequate for robotic applications. [...] we use observations to recognize patterns in the world, not just patterns in the observations. – McCarthy [8]

We adopt McCarthy’s distinction between appearance and reality, and propose a learning framework where appearances are used to induce knowledge about the underlying reality [9]. We consider a fixed set $A = \{x_1, \ldots, x_4\}$ of attributes, and think of each attribute in $A$ as an indicator variable of some aspect of the environment that can be determined through an agent’s sensors. So, for instance, $x_3$ could correspond to whether a cat is at distance less than one meter from the agent, $x_7$ to whether the temperature on Mars right now is above freezing point, and $x_{10}$ to whether Alice and Bob are married to each other. The plausibility of an agent being able to sense such properties as determined by $x_3, x_7, x_{10}$ is entirely domain-dependent.

It is assumed that the environment of an agent is governed by some underlying reality that fully specifies values for each of the attributes in $A$. Such a value assignment is an example of the state of the environment, and is denoted by $exm \in \{0, 1\}^{|A|}$. To encode the possibly complex correlations that exist among the values of attributes in $A$, we assume that examples are drawn from some arbitrary probability distribution $D$. We write $exm \sim D$ whenever $exm$ is drawn from $D$.

An agent does not have access to such examples, nor to the probability distribution $D$, except through sensing. Its imperfect sensors are modelled via an unknown masking process $\text{mask} : \{0, 1\}^{|A|} \to \{0, 1, \ast\}^{|A|}$, which stochastically and arbitrarily maps each given example $exm$ to an observation $obs \in \{0, 1, \ast\}^{|A|}$.

In addition to suppressing attribute values, observations could, in the general case, also flip certain attribute values. To simplify the exposition, however, and as it suffices for our study, we assume noiseless sensors. Hence, each drawn observation $obs \leftarrow \text{mask}(exm)$ may only replace $\{0, 1\}$ values in $exm$ with $\ast$ values, indicating that such attribute values are “don’t know”. Such an observation $obs$ is said to mask the example $exm$, and each attribute $x_i \in A$ whose value $\text{obs}[i]$ in observation $obs$ is $\ast$ is said to be masked in $obs$.

The example 00100111, for instance, can be mapped through a masking process to any of the observations 001*0111, 011*0111, 001*0*11 with non-zero probability, but not to the observation 010010 where the last $\{0, 1\}$ value has been changed.

A sensor reading by an agent is thus obtained by: (i) “nature” drawing an example $exm \sim D$ that determines the underlying reality, and (ii) the agent’s sensors drawing an observation $obs \leftarrow \text{mask}(exm)$ that determines the observed appearance of the underlying reality. Sensor readings, then, constitute the agent’s initial beliefs about its environment. During deliberation, an agent recovers missing information by applying rules on its current beliefs, and updating those beliefs according to the rule predictions. Due to missing information, the premises of the rules might not always evaluate to true or false. In those cases the rules predict “don’t know”, indicating that the current beliefs provide insufficient information for a definite prediction to be made. These considerations are formalized next.

A formula $\varphi$ over $A$ is a function $\varphi : \{0, 1\}^{|A|} \to \{0, 1\}$ that maps examples $exm \in \{0, 1\}^{|A|}$ to $\{0, 1\}$ values. No assumptions are made on these formulas. They are aimed to capture conditions that may hold or not in the underlying reality that is modelled by $exm$. Such formulas are employed as premises of rules.

Consider a class $F$ of formulas over $A$. A reasoning rule $(\varphi) \equiv x_i$ over $F$ is comprised of a formula $\varphi \in F$, and an attribute $x_i \in A$: $\varphi$ is the body, and $x_i$ is the head of the rule. Rules, then, capture an agent’s knowledge about the structure of its environment. No assumptions are made on their correctness a priori. One rule could be $(x_3 \lor x_7) \equiv x_{13}$, where the attributes $x_3, x_7, x_{13}$ are interpreted as earlier. Whether this rule is sensible or useful is domain-dependent.

Rules make predictions on a given observation $obs$ by first evaluating their body, and then predicting the computed value for their head. Thus, a means to evaluate formulas on observations is required.

We define the value $\text{val}(\varphi|obs)$ of $\varphi$ given $obs$ to be the common value of $\varphi$ across all examples masked by $obs$, in case such a common value exists, or $\ast$ otherwise. Thus, $\varphi \equiv x_i$ predicts $\ast$ given $obs$ in exactly those cases that $obs$ does not provide sufficient information for the $\{0, 1\}$ value of $\varphi$ to be uniquely determined.

3 Learning from Partial Observations

The dichotomy between appearance and reality is not only of philosophical interest. An agent’s actions are chosen on the basis of what information is available to the agent; hence, an agent’s deliberation process maps appearances (and possibly prior knowledge available to the agent) to actions. Utility for the agent, on the other hand, is obtained on the basis of what holds in the agent’s environment; reality is, therefore, what determines the appropriateness of an action. It is then evident that agents not only should be aware of this dichotomy, but also that it is rational on their behalf to alleviate the discrepancy between appearance and reality by recovering missing information.

In the spirit of autonomy, an agent should learn the rules by which information will be recovered. Since we assume no teachers, rules need to be induced given access only to the agent’s observations, partial as these may be. Yet, the agent is expected to induce rules that predict the truth according to the underlying examples. This requirement of inducing highly accurate rules is formalized next.

A rule $(\varphi) \equiv x_i$ has an accuracy conflict with an example $exm$ given an observation $obs$ that masks $exm$ if $\text{val}(\varphi|obs) \in \{0, 1\}$ and $\text{val}(\varphi|obs) \neq \text{exm}[i]$. So, for instance, for the rule $(x_3 \lor (x_7 \land \neg \varphi)) \equiv x_4$, the example $exm = 1000110$, and the observation $obs = 10**10$, it holds that $\text{val}(x_3 \lor (x_7 \land \neg \varphi)|obs) = 1$, and $\text{exm}[4] = 0$; there is an accuracy conflict, even though the value of $x_4$ is “unknown” to the agent given the observation. Note that there is no accuracy conflict if a rule predicts $\ast$.

Definition 1 (Accuracy) A rule $(\varphi) \equiv x_i$ is $(1-\varepsilon)$-accurate under a probability distribution $D$ and a masking process $\text{mask}$ if

$$Pr[(\varphi) \equiv x_i \text{ has an accuracy conflict with } \text{exm} \text{ given } \text{obs} | \text{exm} \sim D; \text{obs} \leftarrow \text{mask}(\text{exm})] \leq \varepsilon.$$ 

The accuracy of a rule cannot be determined empirically in the general case, since the rule’s predictions are contrasted against some unknown underlying example. A more benign requirement is to induce rules that are highly consistent with the observations.

A rule $(\varphi) \equiv x_i$ has a consistency conflict with an observation $obs$ if $\text{val}(\varphi|obs), \text{obs}[i] \in \{0, 1\}$ and $\text{val}(\varphi|obs) \neq \text{obs}[i]$. So, for instance, for the rule $(x_3 \lor (x_7 \land \neg \varphi)) \equiv x_4$, and the observation $obs = 10**10, \text{val}(x_3 \lor (x_7 \land \neg \varphi)|obs) = 1$, and $\text{obs}[4] = \ast$; there is no consistency conflict independently of what the value of $x_4$ is in the underlying example that gave rise to $obs$.

Definition 2 (Consistency) A rule $(\varphi) \equiv x_i$ is $(1-\varepsilon)$-consistent under a probability distribution $D$ and a masking process $\text{mask}$ if

$$Pr[(\varphi) \equiv x_i \text{ has a consistency conflict with } \text{obs} | \text{exm} \sim D; \text{obs} \leftarrow \text{mask}(\text{exm})] \leq \varepsilon.$$
The following result establishes that highly consistent rules are also highly accurate to the extent possible. It also shows, in some sense, achieving accuracy through consistency is an optimal strategy.

**Theorem 1 (The Relation of Consistency and Accuracy)** For every processing mask \( \varphi \), and every set of rules with head \( x_t \) and premises in some class \( \mathcal{F} \) of formulas, there exists \( \eta \in [0,1] \) s.t:

1. for every probability distribution \( \mathcal{D} \), and every rule \( (\varphi) \equiv x_t \) in the set, the rule is \((1 - \varepsilon)\)-accurate if it is \((1 - \eta \cdot \varepsilon)\)-consistent;
2. there exists a probability distribution \( \mathcal{D}_0 \), and a rule \((\varphi_0) \equiv x_t \) in the set that is \((1 - \varepsilon)\)-accurate only if it is \((1 - \eta \cdot \varepsilon)\)-consistent.

**Proof (sketch):** Define \( \eta \) to be the least, across all choices of examples \( \text{exm} \) and rules \((\varphi) \equiv x_t \) in the set, probability that \((\varphi) \equiv x_t \) has a consistency conflict with \( \text{obs given that} \ \text{obs} \leftarrow \text{mask(\text{exm})} \) and \((\varphi) \equiv x_t \) has an accuracy conflict with \( \text{exm given obs} \).

1−\( \eta \) corresponds to the concealment degree of \( \text{mask} \) w.r.t. \( x_t \) and \( \mathcal{F} \), where \( \eta \) roughly captures the probability with which an agent’s sensors provide feedback as to when inaccurate predictions are made. The concealment degree differs from notions of missing information traditionally considered in the Statistical Analysis literature [7, 13].

Theorem 1 allows learnability to be studied from the conceptually more direct point of view of consistency. We define learnability by extending the Probably Approximately Correct semantics [14].

**Definition 3 (Consistent Learnability)** An algorithm \( \mathcal{L} \) is a consistent learner for \( \mathcal{C} \) by \( \mathcal{H} \) if for every probability distribution \( \mathcal{D} \) that agrees with some \((c) \equiv x_t \) with \( c \in \mathcal{C} \), every masking process \( \varphi \), and every \((\varphi) \equiv x_t \) in \( \mathcal{C} \), given access only to observations drawn from \( \text{mask(\mathcal{D})} \), it returns in polynomial time a hypothesis \((h) \equiv x_t \) with \( h \in \mathcal{H} \) that, w.h.p., is \((1 - \varepsilon)\)-consistent under \( \mathcal{D} \) and \( \mathcal{F} \).

The concept class \( \mathcal{C} \) captures assumed or known bias on the structure of an agent’s environment, while the hypothesis class \( \mathcal{H} \) captures possible constraints on how an agent represents its learned rules. As in the case of the PAC semantics, Definition 3 asks that learning succeeds for every choice of \( \varepsilon \), albeit running time polynomial in \( 1/\varepsilon \) is allowed to be expended. The central part of the definition is the requirement that learning succeeds under an arbitrary masking process. This lack of any assumption on the masking process captures the autodidactic nature of learning, where an agent attempts to learn from whatever information its sensors provide, without relying on some teacher to externally provide more complete information.

Perhaps somewhat surprisingly, learnability under Definition 3 is not trivialized. We are able to prove, amongst others, the following.

**Theorem 2 (Monotonicity Compensates for Missing Information)** For any classes \( \mathcal{C} \) and \( \mathcal{H} \) that contain only monotone formulas, if it is possible to PAC learn (from complete observations), then it is also possible to learn consistently (from partial observations).

**Proof (sketch):** Reduce the consistent learnability problem to that of learning from complete observations, by replacing all \( \times \) values in observations \( \text{obs} \) with the value \( \text{obs}[^t] \) of the target attribute \( x_t \) in \( \text{obs} \). Employ a PAC learner, and return whatever rule it produces.

From known positive PAC learnability results [3], it follows that consistent learnability is possible when \( \mathcal{C} \) is the class of linear threshold formulas, a rather broad and useful in practice class of rules.

On the negative side, not all classes of formulas that are learnable from complete observations remain so when observations are partial (even if only three attributes are masked) [9]. This suggests a separation from certain related and well-studied learning models [1, 6, 14].

---

**4 Reasoning with Induced Knowledge**

Formal logic systems typically treat the agent’s sensor readings and the agent’s knowledge equivalently as part of a single theory. This theory is taken for granted, and one seeks through reasoning to compute its implications. In Propositional Calculus, for instance, the implications of a theory are those propositional formulas that are satisfied by all truth-assignments that classically satisfy the theory. Similar in spirit, albeit more complex, semantics determine the implications of a theory under Reiter’s Default Logic formalism [12], the Answer Set Programming formalism [2], and other formalisms.

The notion of reasoning that we propose in this work significantly diverges from the above. We view knowledge as being unreliable, since having knowledge that perfectly captures the structure of an agent’s environment is unrealistic in real-world domains. Therefore, we can no longer treat this knowledge as granted and simply seek its implications. Rather the goal is external to the knowledge available, and is that of recovering the true state of the agent’s environment.

Reasoning generally involves the application of multiple rules in some given order. Certain rules are applied on the initial belief state, and give rise to an updated belief state. Subsequent rules are applied on this second belief state, giving rise, in turn, to a third belief state, and so on. Rules making “don’t know” predictions do not affect the beliefs, and let the values of attributes persist to the next belief state.

Consider a class of formulas \( \mathcal{F} \). A reasoning layer (of size \( k \)) over \( \mathcal{F} \) is a set \( \ell = \{\langle \varphi^i \rangle \equiv x_t \}_{i=1}^{\ell} \) of reasoning rules over \( \mathcal{F} \), such that each rule head appears at most once — to avoid ambiguity between conflicting predictions. A reasoning sequence (of depth \( d \)) over \( \mathcal{F} \) is an ordered set \( g = \{\ell_j\}_{j=1}^{d} \) of reasoning layers over \( \mathcal{F} \).

**Definition 4 (Reasoning Conclusions)** The conclusion of a reasoning sequence \( g = \{\ell_j\}_{j=1}^{d} \) on an observation \( \text{obs} \) is the observation \( \text{conc}(g | \text{obs}) \) obtained by: (i) recursively computing the conclusion \( \text{obs}^{\text{cur}} = \text{conc}(\{\ell_j\}_{j=1}^{d-1} | \text{obs}) \); and (ii) replacing \( \text{val}(\varphi | \text{obs}^{\text{cur}}) \) for \( \text{obs}^{\text{cur}} \) for every rule \((\varphi) \equiv x_t \) in \( \ell_d \) s.t. \( \text{val}(\varphi | \text{obs}) \in \{0,1\} \). We define \( \text{conc}(\{\} | \text{obs}) = \text{obs} \).

Reasoning layers are considered in order. Each rule \((\varphi) \equiv x_t \) with a \((0,1)\) prediction causes the value of \( x_t \) in the current \( \text{obs}^{\text{cur}} \) to be updated accordingly. For instance, consider the reasoning sequence \( g = \{\langle x_1 \lor x_2 \rangle \equiv x_5, \langle x_6 \rangle \equiv x_2 \}, \{\langle x_T \lor (x_5 \land \neg \varphi) \rangle \equiv x_4 \} \) of depth 2, with two reasoning layers of sizes 2 and 1, respectively. Reasoning layers correspond to the parallel application of rules to draw conclusions. When \( g \) is to be applied on \( \text{obs} = 1**100*1* \), the first reasoning layer is initially considered, and the bodies of its rules are evaluated on \( \text{obs} \). If \( x_1 \lor x_3 \) evaluate, respectively, to 1 and 0. Note that \( x_1 \lor x_3 \) evaluates on \( \text{obs} \) despite the value of \( x_3 \) in \( \text{obs} \) being “don’t know”, since all examples masked by \( \text{obs} \) agree on the value of \( x_1 \lor x_3 \). Now, given the value obtained for each rule’s body, the observation \( \text{obs} \) is accordingly updated on the attribute corresponding to the rule’s head; thus, \( \text{obs} \) becomes \( \text{obs} = 10**100*1* \). Note that the value of \( x_2 \) has been completed from “don’t know” to 0. Note also that the value of \( x_2 \) has changed from 0 to 1. Hence, rule predictions override the corresponding sensor readings — we discuss later how this can be lifted.

Once \( \text{obs}^{\text{cur}} \) is constructed, the process is repeated by considering the second reasoning layer of \( g \). The bodies of the rules are now evaluated on the new observation \( \text{obs}^{\text{cur}} \), and are able to exploit the information on \( x_2 \) and \( x_7 \) that was provided by the rules in the first reasoning layer. Evaluating the body of \( \langle x_T \lor (x_5 \land \neg \varphi) \rangle \equiv x_4 \) on \( \text{obs}^{\text{cur}} \)
gives *, since the rule’s body evaluates to different \{0, 1\} values on the examples masked by \textit{obs}'', depending on the value of \(x_7\). Due to the “don’t know” prediction, the value of \(x_7\) will not be affected when updating \textit{obs}; the updated observation \textit{obs}’’ is, thus, equal to \textit{obs}'. Since all reasoning layers of \(\varphi\) have been applied, the conclusion of \(\varphi\) on the initial observation \(\textit{obs}\) is \(\text{conc}(\varphi|\textit{obs}) = \text{obs}''\).

Without going into the formal details, we very briefly examine the expressivity of our proposed reasoning semantics. Although we have followed the approach that rules encode definitional knowledge about the structure of an agent’s environment, one may wish to also consider epistemic rules that encode knowledge about an agent’s state of knowledge (see, e.g., [12]), or implicational rules (see, e.g., [5]). One such epistemic and implicational rule, for instance, states that ‘‘if an entity is a bird, and it is not known to be a penguin, then it has the ability to fly’’, and captures the natural requirement that in the absence of contrary evidence, a reasoner is expected to deduce that birds are able to fly. It can be shown that reasoning with such epistemic or implicational rules can be simulated by, and seen as a manifestation of, the more primitive form of ordinary reasoning that we have proposed. Similarly, one can encode rules that make predictions on a target attribute \(x_1\) only when the attribute has a * value, so that predictions do not override the value of \(x_1\) when it is observed.

4.1 Soundness and Completeness

In defining how a reasoning sequence updates an observation to reflect an agent’s updated beliefs, we have made no claims as to the appropriateness of the obtained beliefs. A natural means to do so is to insist that each attribute \(x_i\) that is assigned a \{0, 1\} value in a belief \textit{obs}, agrees with \textit{exm}’'; in particular, if the value of \(x_i\) in \textit{obs} is *, then this does not count as a mistake in the beliefs of the agent, but simply as a lack of belief on \(x_i\). The above requirement is concisely captured by defining an observation \(\textit{obs}\) as having a \textbf{soundness conflict} with an example \textit{exm} if \textit{obs} does not mask \textit{exm}.

Given the initial beliefs \textit{obs} of an agent, its final beliefs are the conclusions \(\text{conc}(\varphi|\textit{obs})\) of a reasoning sequence \(\varphi\). It is with respect to these final beliefs that the agent decides on its actions, and it is with respect to these beliefs that \(\varphi\) is evaluated for soundness.

\textbf{Definition 5 (Soundness)} A reasoning sequence \(\varphi\) over a class \(\mathcal{F}\) of formulas is \((1-\varepsilon)\text{-sound}\) under a probability distribution \(\mathcal{D}\) and a masking process \textit{mask} if

\[
\Pr[\text{conc}(\varphi|\textit{obs}) \text{ has a soundness conflict with exm }] \leq \varepsilon.
\]

Since the goal of an agent’s deliberation mechanism is to recover missing information, we would expect that the deliberation mechanism makes some progress in recovering missing information that is not presently necessary in the initial belief state formed from the sensor readings. We define an observation \(\textit{obs}\) to be \textbf{complete for} a target attribute set \(\mathcal{A}_t\) if none of the attributes in \(\mathcal{A}_t\) is masked in \(\textit{obs}\). As for soundness, completeness is of interest with respect to an agent’s final beliefs, obtained as conclusions of a reasoning sequence.

\textbf{Definition 6 (Completeness)} A reasoning sequence \(\varphi\) over a class \(\mathcal{F}\) of formulas is \((1-\omega)\text{-complete}\) for a target attribute set \(\mathcal{A}_t \subseteq \mathcal{A}\) under a probability distribution \(\mathcal{D}\) and a masking process \textit{mask} if

\[
\Pr[\text{conc}(\varphi|\textit{obs}) \text{ is not complete for } \mathcal{A}_t ] \leq \omega.
\]

Unlike formal logic systems, soundness and completeness have a statistical flavor, and are defined externally to any given set of rules.

4.2 Non-Collapsibility of Reasoning

Given our reasoning semantics, we now ask: Are multiple reasoning layers in a reasoning sequence useful? The question is particularly intriguing when the rules are learned, where the eventuality that the learning process may induce and encode in the learned rules the functionality of reasoning, cannot be a priori dismissed.

Valiant [16] offers pragmatic considerations as to why the answer is affirmative in the context of automated acquisition and handling of unaxiomatized knowledge: the statistics of the data might not support the induction of a single-layered reasoning sequence, and even if they do, the induction task might be computationally hard; and programmed rules might need to be integrated in the reasoning process.

Further empirical evidence in support of an affirmative answer are discussed by Dietterich [4] in the context of aggregating multiple learned rules: a statistical reason relates to the scarcity of training data; a computational reason appeals to the hardness of searching the hypothesis space; and a representational reason accounts for the case that the hypothesis class is not sufficiently expressive.

All discussed evidence in favor of an affirmative answer to the question posed appeal, to some extent, to inherent limitations of the learning process. The reasoning process, however, exists independently of the learning process, and might be employed by agents with no learning mechanism. To provide a formal and more general response to our posed question, we first make the question precise.

\textbf{Reasoning collapses for} a target attribute set \(\mathcal{A}_t \subseteq \mathcal{A}\) under a probability distribution \(\mathcal{D}\) and a masking process \textit{mask} if for every reasoning sequence \(\varphi\) that is \((1-\varepsilon)\text{-sound}\) under \(\mathcal{D}\) and \textit{mask}, and \((1-\omega)\text{-complete}\) for \(\mathcal{A}_t\) under \(\mathcal{D}\) and \textit{mask}, there exists a reasoning sequence \(\varphi'\) of depth 1 that is \((1-\varepsilon')\text{-sound}\) under \(\mathcal{D}\) and \textit{mask}, and \((1-\omega')\text{-complete}\) for \(\mathcal{A}_t\) under \(\mathcal{D}\) and \textit{mask}, s.t. \(\varepsilon' + \omega' \leq \varepsilon + \omega\).

Is it, then, possible to identify a reasoning sequence that \textit{strictly} outperforms, in terms of soundness and completeness, every single-layered reasoning sequence? We claim that the reasoning sequence that chains the following two rules in two layers achieves this:

Rule in layer 1: \(\text{(noon time)} \equiv \text{lunch time}\).

Rule in layer 2: \(\text{(lunch time and hungry)} \equiv \text{eat lunch}\).

To illustrate our claim, consider the following attempt to “merge” the two rules into a single rule that predicts \text{eat lunch}:

\begin{align*}
\text{Rule:} & \quad ((\text{noon time or lunch time}) \text{ and hungry}) \equiv \text{eat lunch}.
\end{align*}

Consider a scenario where Alice is hungry, and she is told by Bob that it is lunch time. Since Alice’s wristwatch is broken, she does not observe whether it is noon time. The latter rule evaluates to “don’t know”, since the unknown value of \textit{noon time} does not allow the premises of the rule to be fully determined. By contrast, the rules in the original pair collectively make a definite prediction; the first rule predicts “don’t know”, leaving the observed information that it is lunch time unchanged, while the second rule then deduces that Alice should eat lunch. A similar phenomenon occurs if Alice observes that it is noon time, but she is not told by Bob that it is lunch time.

This phenomenon manifests itself whenever more than one piece of information unilaterally determines the prediction to be made in some context. In the example above, the context is that Alice is hungry, and the two pieces of information each of which unilaterally determines that Alice should eat lunch are \textit{noon time} and \textit{lunch time}.

Consider an observation \textit{obs}, a formula \(\varphi\) over \(\mathcal{A}\), and an attribute \(x_i \in \mathcal{A}\) that is masked in \textit{obs}. Consider changing the value of \(x_i\) in
obs to either 0 or 1. If both of these changes to \( x_i \) cause the formula \( \varphi \) to change its value, then call \( x_i \) critical for \( \varphi \) w.r.t. \( \text{obs} \).

Lemma 3 (Uniqueness of Critical Attributes) There is at most one critical attribute \( x_i \) for each formula \( \varphi \) w.r.t. an observation \( \text{obs} \).

Lemma 3 implies an inherent and unconditional limitation of individual rules. No rule, programmed or induced, and no matter how expressive, may have more than one source of information unilaterally determining its prediction. On the other hand, multi-layered reasoning can be used to properly merge all the sources of information. This realization underlies the result established below.

A masking process mask is \( k \)-critical inducing under a probability distribution \( D \) for a target attribute \( x_i \in \mathcal{A} \), if there exists an observation \( \text{obs} \) — mask\( (D) \) drawn with non-zero probability, in which \( k \) attributes determine (directly or indirectly) the value of \( x_i \).

Theorem 4 (Domains with Non-Collapsible Reasoning) For every \( k \geq 2 \), and every masking process mask that is \( k \)-critical inducing under a probability distribution \( D \) for a target attribute \( x_i \in \mathcal{A} \), reasoning does not collapse for \( \{x_i\} \) under \( D \) and mask.

Proof (sketch): Choose \( q_1 \) that maximizes the performance among reasoning sequences of depth 1. Extend it to \( q_k \) by adding perfectly accurate rules in later layers for each of the \( k \) attributes that determine the value of \( x_i \). The multi-layered reasoning sequence \( q_k \) can be shown to be perfectly sound and complete on some particular set of observations \( \mathcal{O} \). As a consequence of Lemma 3, every single-layered reasoning sequence that makes \( \{0, 1\} \) predictions on all observations in \( \mathcal{O} \) necessarily makes at least one wrong prediction.

The necessity of interim reasoning for inducing multi-layered reasoning sequences is accompanied by a sufficiency property.

Theorem 5 (Learning via Interim Reasoning) Using interim reasoning, and by an appropriate choice of accuracies for rules at each layer, any learning algorithm \( \mathcal{L} \) can be extended to one that produces multi-layered reasoning sequences, does not sacrifice the soundness guarantees of \( \mathcal{L} \), and improves upon its completeness guarantees.

Proof (sketch): Before learning rules for the \( j \)-th layer via interim reasoning, ensure that the reasoning sequence \( q_{j-1} \) that is comprised \( \{0, 1\} \) predictions on all observations in \( \mathcal{O} \) necessarily makes at least one wrong prediction.

In contrast to the pragmatic considerations discussed earlier, Theorem 4 is independent of statistical, computational, or representational assumptions, and applies to programmed and induced rules alike. The established separation presents an explicit, and previously unidentified reason for reasoning in the discussed context.

5 The Necessity of Iterative Learning

The formal benefits of multi-layered reasoning can be seen as a prescription that agents should chain the pieces of knowledge that they employ to draw conclusions. The order in which the rules are to be chained remains, however, unspecified. It is easy to see that not every ordering of rules is useful. In our earlier example for predicting whether Alice should eat lunch, chaining the two rules in the reverse order would yield a reasoning sequence where the rule that predicts \( \text{eat lunch} \) would not exploit the conclusions of the rule that predicts \( \text{lunch time} \). This seems to suggest that one should first induce rules, and then order them in a manner that allows rules to build on the conclusions of other rules. Such ordering is particularly appropriate in hierarchical domains, where the rules do not depend on each other in a cyclical manner, but follow some hierarchy, whereby the premises of rules only depend on the heads of rules lower in the hierarchy. Despite the conceptual merits of a strategy that decouples the induction of rules from the reasoning phase, we argue next that such an approach cannot be employed. Assume that multiple rules are induced independently, and that they are subsequently chained in a reasoning sequence. Consider a rule \( (\varphi) \equiv x_i \) in the last layer of the constructed reasoning sequence. Although during the learning phase \( (\varphi) \equiv x_i \) faced as inputs only the sensor readings of the agent, during the reasoning phase it faces the updated beliefs of the agent, as those are determined by the application of rules in the earlier layers of the reasoning sequence. This change in the rule’s inputs invalidates any guarantees that we may have had on the appropriateness of this rule during its induction. It can be shown, in fact, that the rule may suffer an arbitrary loss in accuracy as a result of this change.

This impossibility result suggests that rules in higher reasoning layers should be induced by facing the same inputs that they will face during the reasoning phase. This can only be achieved if when learning \( (\varphi) \equiv x_i \), all rules in the previous layers have been already learned, and are used to draw conclusions that are to be given as input to \( (\varphi) \equiv x_i \). Thus, an iterative learning strategy is necessitated, one that interleaves learning and reasoning. This prescribes that an agent should use reasoning not only as a means to make predictions about unseen parts of its environment, but also as a means of learning. We use the term interim reasoning to denote such type of reasoning that happens during the learning phase, and for the benefit of learnability.

The necessity of interim reasoning for inducing multi-layered reasoning sequences is accompanied by a sufficiency property.

6 Conclusions

A unified semantics for the processes of learning and reasoning was presented. Special emphasis in our proposed theory has been placed on the absence of any assumptions on the existence of a teacher that facilitates the agent’s interaction with its environment. We believe that the development of agents that follow such an autodidactic approach to learning and reasoning can find numerous applications, ranging from the extraction of knowledge from text, where a sentence can be naturally viewed as a partial appearance of some underlying scenario; to the design of unmanned robots for space exploration, where a robot should be able to cope with novel situations that it will face by learning and reasoning about the structure of its environment; and even as a means to understand certain aspects of human intelligence, especially with respect to the acquisition of commonsense knowledge from our everyday interactions with our environment.
We have focused in this work on a passive learning setting, where the agent does not affect its environment through actions, and does not actively choose which environmental states and attributes to observe (e.g., by incurring some cost). Investigating such extensions of our theory is a promising venue for future research, and in the spirit of developing a complete cognitive theory of autodidactic agency.

Acknowledgments

The author is grateful to Leslie Valiant for his advice, and for valuable feedback on this line of research.

REFERENCES

Combining appearance and behavioral cues for the purpose of person recognition

Fiora Pirri and Matia Pizzoli
Sapienza Università di Roma, email: {pirri,pizzoli}@dis.uniroma1.it

Abstract. The recognition of a person is based not only on the person appearance but also on several other aspects that depend on the person facial expressions and his/her way of moving the head or directing the gaze. The more we know a person the better we are able to recognize he/her even in a crowd of people or in unusual and unexpected circumstances. These are the cognitive aspects of recognition that we would like to capture by classifying both the space of the appearance features together with the rigid head motions, and the space of facial actions, generating the person peculiar expressions. To represent the facial actions we take also into account their dynamics and the sequence of transformations that specifically characterize an individual. At this stage of our research we emphasize that the advantage of our approach is mainly in the effort of providing a new perspective for people recognition in terms of classification strategy. However, we also provide some test results that show the efficacy of our method in terms of generality.

1 INTRODUCTION

Identifying known individuals is a crucial skill of humans and other animals. The term known individuals is, however, too general, because we cannot specify which are the minimal set of features, properties and other peculiarities that characterize the knowledge of a person face.

Indeed several sources of information are connected to activate re-cognition and thus retrieval of names, places and other memories associated with a person; yet the computational process taking place in humans for the recognition of a person is not clearly understood, as many neurological and psychological studies witness. It is still controversial whether face perception is a specific brain functionality and how it relates to object categorization and recognition (e.g. [23, 14, 12]).

Anyway, several experiments on face visual perception have shown that there is a complex structure that underlies the recognition process, including structural and semantic information, like displacements of spatial configuration of inner and outer features, psychological correlates of familiarity and emotional information induced by expressions (see e.g. [35, 31]).

According to these findings visual recognition of faces should rely on a structured knowledge of the person to be recognized that must be based on different aspects that emerge while interacting and communicating or just observing the other person acting (e.g. in TV). This structured knowledge is the outcome of a learning process able to extract the peculiar aspects and define the person’s signature. This coded representation of a subject’s behavior allows subsequent recognition under different circumstances strongly varying the subject’s appearance: changes in head pose and illumination conditions but also aging, hair, eye-glasses, make up or other kind of visibilia and camouflage. On this basis, we propose to characterize a person face according to diverse aspects, initially considering the following:

1. the person appearance, in terms of spatial configuration of features;
2. the way the person move and scroll the head and directs the gaze;
3. expressions, involving eyes, eyebrows, mouth deformations and the induced face transformations.

These aspects are supposed to be captured and classified in a video sequence while the subject is conversing.

The model we propose is the following. We identify a set of fiducial points on the face that can be tracked under different poses. We compute the head rotations in terms of pitch, roll and yaw with respect to the camera reference frame. The sequence of the RPY triples is smooth because the rotations are computed for each frame. Then, using the extracted fiducial points, we define a hierarchical graph in which the nodes labelled by the fiducial points are the roots of sub graphs denoting the metric of the features we consider: mouth, eyes, nose and eyebrows. We track the transformations of the hierarchical graph in terms of distances. These are the observable features hiding the person typical expression, which are estimated as states of a Hidden Markov Model whose transitions denote the face actions. The number of states is fixed so that the correspondence between states of different subjects can be established, again in terms of a distance between observations, which are multivariate Gaussians. In other words each state represents a typicality of a person. In this way it is possible to estimate the subject’s appearance for each of the computed states. Clearly the approach is feasible as far as the number of states is limited. Then recognition amounts to finding, according to the defined aspects, the corresponding states and identifying the subject that minimizes the distance between the states computed and memorized or predicted by the model.

Finding and identifying a human face is a very successful application in the field of computer vision and pattern recognition and related problems have been studied since early sixties [5]. For recent reviews we address the interested reader to [37, 19].

In order to compute the transformations, we make use of some approaches available in literature for detecting and tracking fiducial points, while the hierarchical graph construction, the states and observations identification is a novel view in modeling face perception and recognition.

The paper is organized as follows: in the next section we describe how we compute the head-face rotations and translations and how we extract the fiducial points. Further, we describe the measurement pro-
cess that leads to the building of the hierarchical graph and finally we
discuss some preliminary experiments. This paper is intended as the
abstract of a research work that certainly needs further development
but seems very promising according to the first results described in
the following.

2 THE MEASUREMENT MODEL

For each subject we are given a video sequence composed of pairs of
frames acquired at 25 Hz framerate, one frame for the RGB image
and the other frame for the corresponding distance image, i.e. the
XYZ map.

The XYZ map provided by the used stereoscopic device is only
partially dense and, in order to obtain a dense head map (see Fig-
ure 2), we make use of bilinear interpolation. The head is tracked
throughout the frame sequence by a feature tracking routine based on
the early work by [24], modified in [32] and fully explained in [27].
Each frame is normalized and scaled into a fixed size face image.
Then, a hierarchical topological representation is associated to each
identity. Faces are represented by weighted graphs, whose nodes are
labeled by a Gabor feature representation of some fiducial points,
e.g. corners of eyes, mouth, and eyebrow (see Figure 1).

Figure 1. The deformation parameters extraction is based on the
recognition of the fiducial points encoded by Gabor wavelet transform, a
convolution with a set of wavelet kernels. The set of 40 coefficients encoding
an image point is referred to as a jet. In the figure only 2 frequencies and 3
orientations are shown.

Edges are weighted with the relative distances between fiducial
points so that face deformations are modeled by the changing in time
of the edge labels in the graph representing a face. As usual, the
whole system description goes through the feature extraction, learn-
ing and classification steps.

Modeling a human face as a graph requires a robust encoding
of the chosen fiducial points. In a way that is similar to the one in
[36, 33], the magnitudes of a set of multifrequency and multiorien-
tation Gabor wavelets are used as a representation for each fiducial
point [21]. Such a codification is widely used in computer vision as
it provides robust performances even under illumination and appear-
ance variations. Moreover, it is biologically plausible, as the filter
resemble the receptive field profile of simple cells in the primary vi-

sual cortex of vertebrates [25, 17].

In literature, the set of complex coefficients describing a small
patch of grey values in an image \( I(x, y) \) transformed by multifre-
quency and multiorientation Gabor filtering is called a jet and is de-

defined by the convolution

\[
J(x, y) = \int_{\infty}^{\infty} \int_{-\infty}^{\infty} I(x + r \cos \theta - y + r \sin \theta; f_0, \theta) dx \, dy
\]

(1)

with a family of Gabor kernels

\[
\psi(x, y; f_0, \theta) = \frac{1}{\pi \gamma ^2} e^{-\left( \frac{x^2 + y^2}{\gamma ^2} \right)} e^{i(2\pi f_0 x')}
\]

(2)

that implements a 2D filter made up of a complex exponential, rep-
resenting a sinusoidal plane wave, restricted by a Gaussian envelope
function. The sharpness of the filter is controlled by \( \gamma \) along the major
axis and by \( \eta \) along the minor axis. The \( f_0 \) and \( \theta \) parameters
respectively express the central frequency of the filter and the rota-
tion angle of the Gaussian major axis and the plane wave. Several
Gabor filters are combined to form a filter bank, which is composed of
filters with different orientations and frequencies, with equal ori-
entation spacing and octave frequency spacing. The relative widths
of Gaussian envelope \( \gamma \) and \( \eta \) are kept constant. Our system encodes
the probe faces by filtering them with a bank characterized by a set
of 5 frequencies and 8 orientations.

2.1 Feature selection

In order to quickly and robustly detect fiducial points in the probe
face image we select features by an AdaBoost based method ([9]).
This classification technique boosts the classification performance of
a simple learning algorithm (the weak learner, whose accuracy on
the discrimination task are slightly better than chance) by combining
a collection of weak classification functions to form a stronger clas-
sifier. AdaBoost is widespread in literature for feature selection and
has been reported to achieve outstanding results if compared to other
supervised classifiers [2].

To use AdaBoost for feature selection, we treat each Gabor filter
as a weak classifier and, for each round, the best of those classifiers
is select as the one who provides the best performance on the errors
committed by the previous.

The system requires a first learning step in which the feature selec-
tors are trained on manually extracted fiducial points. Given the face
image, a bank of multifrequency, multiorientation Gabor filters is ap-
pied to obtain a feature representation based on jets. An AdaBoost
classifier is trained for each fiducial point through a general purpose
learning mechanism that uses jets from the learning sets representation of known fiducial points. Therefore feature selection is performed for every fiducial point, resulting in a linear combination of filters that are the most characterizing for the specific fiducial point.

The graph structure is superimposed to each acquired frame limiting the search to a neighborhood of each previous observed fiducial point. For this reason, it must be deformed, scaled and translated according to the current estimated rotation and the face detector output.

3 TRACKING HEAD ATTITUDES AND INCLINATIONS

Given the sequence of maps (XYZ values) at time $t_0, \ldots, t_m$ and the fiducial points (as explained in the previous section), the rotation and translation of the head between two frames, with respect to the camera reference frame (note that in this preliminary version the camera is fixed) can be easily estimated as a rigid transformation between two 3D point sets on the face, as they appear in the sequence. Given matched point sets $X \in Y$, $|X| = |Y| = N$, $R$, rotation matrix and $t$, displacement vector, we formalize the problem as a 3D point registration problem:

$$\min_{R, t} \sum_{i=1}^{N} ||x_i - (Ry_i + t)||^2$$

(3)

where $x_i \in X$ and $y_i \in Y$ are corresponding points. Centering the points we obtain $\bar{x}_i$ and $\bar{y}_i$, and the following is equivalent to expression (3):

$$\min_{R, t} \sum_{i=1}^{N} ||\bar{x}_i - R\bar{y}_i||^2$$

(4)

which is minimized when $\text{tr}(R^\top K)$, $K = \sum_{i=1}^{N} \bar{x}_i\bar{y}_i^\top$ is maximized. This is true ([18]) when, being $K = VD U^\top$ the singular value decomposition of $K$, $(U^\top R^\top V) = I$. Therefore, the solution is

$$R = VU^\top$$

(5)

$$t = \frac{1}{N} \left( \sum_{i=1}^{N} x_i - R \sum_{i=1}^{N} y_i \right)$$

(6)

By hypothesis the rotations are very small, therefore we can easily use Euler angles and assume that each computed rotation can be transformed into a fixed sequence of pitch, roll and yaw movements of the head. Thus, we let $R = R_{roll}R_{pitch}R_{yaw}$, with $R$ the matrix obtained as above, and the single angles $(\psi, \theta, \phi)$ can be easily computed, with respect to the fixed reference frame, using a four quadrant inverse tangent function. The sequence of rotations will be used in the computation of the states.

4 COMBING APPEARANCE AND BEHAVIORAL CUES

Since we are interested in characterizing a subject making use of behavioral cues, our system stores relative distances between fiducial points extracted from a sequence. A subject behavior is modeled by the change in time of a multivariate $W \in \mathbb{R}^{m \times 3}$ specified by the following observed data: distances $\{d_1, d_2, \ldots, d_m\}$ between fiducial points, and the head RPY angles $(\psi, \theta, \phi)$.

For each subject $P$ and for each time step $t$, the multivariate $W$ accounts for the hierarchical graph (which we do not describe here). Figures 3 and 4 refer to the whole structure normalized by projecting the data, i.e. distances between fiducial points expressed by the edge labels, into a 2-dimensional space making use of Principal Component Analysis. It is worth noting that such a dimensionality reduction has been performed for the sake of data visualization.

The structure $W$ will be used to estimate the hidden states corresponding to the subject expressions via the Hidden Markov Models (HMM).

The use of HMMs for classification relies on a number of parameters whose values are set so as to best explain training patterns for the known category. In our system, training patterns are sequences $S = \{W_1, W_2, \ldots, W_n\}$, where each sequence is a matrix $k \times m$ of $k$ observations and $m$ distances, according to the fiducial points modelled in the hierarchical graph. That is, $W_i = x_1, \ldots, x_N$ and we shall index the $W_i$, eventually with the subject the test pattern refers to. For each subject to be recognized a test pattern is acquired; this sequence is then classified by the most likely model. Classification amounts to single out hidden states as specific expressions of a subject. Each expression is a configuration of both head orientations and face deformations. Therefore observations, i.e. state emissions, are the multivariate obtained by measuring distances between fiducial points. We chose to model emissions with Mixtures of Gaussians. To train these models, the Baum-Welch algorithm is used [3]. In this approach, the parameters of every Gaussian mixture representing the observation of each state of the HMM have to be estimated and, for each iteration of the EM algorithm, the HMM transition matrix and the prior probabilities have to be re-estimated. Training requires providing the coarse structure of the HMM, i.e. the number of states and
the number of visible states. We use the same number of states and Gaussians for every mixture, relying on Mean Shift [11, 7] to output such a parameter and initialize the iterative estimation (see section 5). Our system trains an HMM for every training set, that is, one model for each known person.

![States and emissions](image)

**Figure 3.** States and emissions: states represent clusters of parameter configurations; emissions are modeled by mixtures of Gaussians. Given the number of hidden states, training an HMM for every subject produces parameters for the mixtures and the transition matrix. In the figure, only 3 states and 3 Gaussians per mixture are showed; data are reduced to 2-dimensional by PCA.

We are given a model \( \Omega_B = (S, \Theta, W) \), over a domain \( (S, W) \) of states and observations, for each subject \( B \), where \( S \) is the transition matrix and \( \Theta \) mentions all the parameters of the Gaussian mixtures modeling the observation at each state. The Gaussian at each state \( S \in S \), encodes the person appearance under the face action aspects. That is, if for example a specific state \( s_m \) specifies a certain typical grimace of \( B \) then, associated with such a state, there will be a mean face (we do not describe this here) of that grimace and the mixture represents the average deformations of the \( B \)'s face when he or she makes that grimace. Therefore the mixture encodes exactly the spatial configuration of the features, however we intend to extend this set with other appearance features capturing the mean face of the specific state.

Now, given a video sequence of a subject \( B \), i.e. a set of observed face actions \( W_B = \{x_1, \ldots, x_N\} \), and the set of models \( \Omega = \{\Omega_1, \ldots, \Omega_N\} \) of the known persons, the recognition of \( B \) amounts to determine the model of the observed face actions, among those in \( \Omega \), that maximizes our belief that this model explains the subject \( B \)'s behaviour:

\[
P(\Omega_i | W_B) \propto P(\Omega_i) \prod_{n=1}^{N} P(x_n | \Omega_i)
\]

The model \( \Omega_i \) that maximizes the above posterior, will identify the person having a behaviour most similar to \( B \). This means that a best model is always found, even if \( B \) has not been classified, hence the choice will be committed to a threshold. However to be able to define a collection of models, it is necessary to preliminary define a correspondence and hence an ordering between states related to each person \( B \). Since states are hidden and only observations can be measured, and since we model observations as mixtures of Gaussians, the Bhattacharyya distance, that measures the similarity of two probability distributions, can be used to determine the state ordering.

In other words, given two models for subjects \( A \) and \( B \) and given that \( g_A(x|\Theta, S) \) is the observation at state \( S \in S_A \) of subject \( A \), conditional to the model parameters \( \Theta \), that best explains the aspect action face of person \( A \), and given that \( f_B(x|\Gamma, Q) \) is the analogous function but related to person \( B \), then state \( S \) and state \( Q \) are in correspondence if, given that \( D \) is the set of states already ordered, then

\[
\forall S' \in S_A, S' \notin D, \forall Q \in S_B, Q \notin D :

\sum_{x \in W} (g_A(x|\Theta, S)) f_B(x|\Gamma, Q))^{1/2} < \sum_{x \in W} (g_A(x|\Theta, S') f_B(x|\Gamma, Q))^{1/2}
\]

It is easy to see that the above condition ensures a total ordering on states. The total order does not change the transitions nor the Gaussian emission, hence given a new video sequence and a sequence of states and emissions estimated by a given model, it is possible to suitably compare the states sequences and determine the underlying model of the observed person. Furthermore it is possible to label the observations and characterize the aspect action faces.

5 EXPERIMENTS AND ANALYSIS

Experiments have involved about 30 volunteer students, whose 3D faces have been collected to form 25 Hz data sequences using a Dual Core Pentium based PC and both a Bumblebee, by Point Grey Research, and two calibrated Marlin, by Allied Vision. Most of the software was in the form of Matlab scripts and this prevented us from executing it on-line, even if we think that a C++ implementation would execute in almost real time. It should be noticed that the number of test cases is inadequate to experimentally validate our system but, since the approach described in this paper is somehow preliminary to a more thorough research, we understand the need to carry out extensive experimenting. For the acquisition of the sequences, the subjects have been asked to behave naturally in front of the stereo rig as if they were introducing themselves to a human. Every subject’s behaviour is encoded by a 30 seconds sequence.

Training the fiducial point detector required manually labeling about 70 frames. AdaBoost classifier selected a combination of filters capable of good detection rates. As a result, Gabor based fiducial points extraction has proven accurate and even robust against change in illumination conditions, besides being quite fast. Superimposing the graph structure prunes the search space providing a consistent speedup to overall performances. Correct detection rates vary strongly among the fiducial points, with an highest rate of 92% for frontal images, drastically decreasing as the head is rotated.

Classification of behaviors through the Hidden Markov Models was quite successful too. Even with subject speaking and naturally acting in front of the camera, the correct model was selected with a percentage varying with the chosen number of hidden states. Figure 3 and 4 show a three states model. Nevertheless, the system has been tested on video sequences making use of different numbers of states for each model. Experiments have demonstrated that HMMs can capture behavioral cues useful for the purpose of discrimination. To better investigate the contribution of information related to head rigid motion and face deformation our experiments included
two sessions. In the first, volunteers were asked to perform a sequence of face deformation trying to keep the head pose fixed.

In the second session probe subjects were allowed to act spontaneously, thus performing head pose variations and face deformation simultaneously. Table 1 reports the recognition rates for models characterized by different numbers of states.

### Table 1. The number of states which characterizes the models is crucial for recognition. Rates obtained for different numbers of hidden states are reported.

<table>
<thead>
<tr>
<th>Number of States</th>
<th>Recognition Rates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fixed Head pose</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>53.4</td>
</tr>
<tr>
<td>6</td>
<td>83.4</td>
</tr>
<tr>
<td>8</td>
<td>86.7</td>
</tr>
<tr>
<td>10</td>
<td>83.4</td>
</tr>
<tr>
<td>12</td>
<td>80</td>
</tr>
</tbody>
</table>

Experiments showed that models with too few states are unable to provide information useful to discriminate between known individuals, as demonstrated by the very low recognition rates reported in the first rows of Table 1. On the other hand, too many states can even lead to worsening recognition performances. Given the number of states, we use \( k \)-means clustering to initialize the iterative density estimation and, if the given states are too many, some of them may represent very few samples. Note that the reported states are only representative of the data set used for our experiments and are not intended as generally applicable to different data sets.

We chose to address the number of hidden states problem detecting the modes of the data distribution by a Mean Shift based procedure [11, 7]. This nonparametric technique is widely known for the analysis of complex multimodal feature spaces and we apply it setting the window dimension to the value providing the best decomposition stability, i.e. choosing the center of the largest operating range over which the same number of clusters are obtained for the given data [10]. Results from Mean Shift based analysis of 2 data sets are depicted in figure 6. Not surprisingly, the experiments underlined the meaningful contribution of head pose variation, as reported by Table 1 whose third column values report higher rates than the ones from the second column.

In experiment sessions which included the automatic estimation of the number of hidden states, the patterns of behavioral data extracted from the sequence were assigned to the right model with rates of 90 and 96.7 respectively, for fixed head pose and unconstrained small motions, thus proving the efficacy of the proposed approach.

### 6 COMPARISON WITH THE LITERATURE AND CONCLUSIONS

To address the recognition problem several approaches have been proposed, both statistical and neural based. Recent experiments report excellent results in the cases of frontal views by applying statistical analysis of subspaces techniques [34, 29, 1]; a certain robustness against expression changes has been achieved through the DLA/EBGM [20, 36] neural architecture, while the pose and illumination change problems have been successfully addressed estimating pose and light source [13] or by 3D model fitting. A very popular approach to face modeling based on computer graphics techniques is described in [4]. Principal Component Analysis (PCA) is applied in the face space to learn deformation manifolds from a set of 3D face models acquired by laser scanning. In [15] deformable models similar to the ones used by [4] are aligned to faces in 2D pictures by estimating deformation and pose through an Expectation Maximization based procedure. A more comprehensive survey can be found in [38]. Nowadays, face recognition systems are capable of considerable hit rates, yet we think a biologically motivated approach to people identification should somehow catch behavioral peculiarities, from gestures, expressions and head movements, leading to a novel approach to perception in which information coming from such different sources is collected and classified.

On the other hand, most of the research on facial expression analysis has been focused on recognizing basic emotions as happiness, sadness, surprise, disgust, fear and anger. The work by Ekman and Friesen [8] suggested to code facial expressions by decomposing them in action units (AUs), related to the contraction of specific set of facial muscles. Recent approaches address the problem of replacing the original manual coding with a more desirable automatic, real time AU recognition system. The proposed methods differ: 1) for the techniques used in the feature extraction or AU classification stage; 2) for treating the face as a whole rather than considering only a specific set of features; 3) for analyzing AUs frame by frame or mind the evolution in time. In the work described in [33] a Dynamic Bayesian Network (DBN) is used to overcome some limitations of the static approaches, mainly concerned with modeling the syntactic relationships among different AUs.

HMM have found greatest use in problems involving processes that unfolds in time, for instance speech [26] or gesture [28] recognition. Our approach shares with the work described in [16] the use of HMMs to represent and recognize complex human actions in a sequence. Nonetheless, 2D data are used and motion estimation relies on flow fields computation, thus consisting in a holistic approach, while our is a 3D, feature based one.

However most of these studies on facial expressions and induced deformations do not explicitly connect them to the subject individuality for recognition purposes. Inspired by the human brain ability to discriminate on the basis of recurrent expressions, attitudes and ges-
turing, we think that a way to improve such impressive classification performances is considering all the information humans do usually consider, even if unconsciously (see e.g. [22, 30]). As a consequence, we do not rely on the extraction of Action Units, that are commonly used as features to perform expression classification, as we do not consider nominal expressions such as anger, fear, disgust, surprise, happiness, sadness, etc. Instead, we consider typical individual expressions as representing the subject personality; these states in our approach are hidden and they represent a set of transformations that can certainly be reduced to specific facial nerve and associated function, such as platysma and frowning, but they also might appear in combination, according to the deployed states.

In this sense, while this paper does not introduce any substantial contribution to face appearance recognition, our attempt to extract information useful to discriminate person faces by modeling head pose and face deformation has not yet been covered by literature.

In the recent work described by [6] advantages introduced by stereoscopic information to the problem of generalizing across different views of faces are investigated. Experiments demonstrate that viewpoint costs are reduced by depicting the face with stereoscopic three-dimensionality, with respect to synoptically presented faces. We also have used a stereopsis approach to capture typical head movements and to obtain a reliable set of distance measures.

We expect to extend the model with more data relative to the appearance and to better specify the relationship between states and expressions for each subject, besides extending the experiments to cope with a much larger database of people acquired under different light conditions. Moreover, we are investigating the benefits of an approach based on flow computation instead of relying on the extraction of a specific set of fiducial points.

To summarize, we have presented a 3D, feature based approach to head motion modeling through a set of pose and deformation parameters whose evolution in time retains information useful to characterize a subject from a behavioral point of view.

References


Figure 5. Images from 2 sequences used in experiments. Face expression and pose changing provides information useful to discriminate among known individuals.
Figure 6. Mean Shift analysis of multimodal densities. Figures (a) and (c) depict two distributions of data reduced by PCA to 2 principal components. Figures (c) and (d) show the maximum modes detected by the Mean Shift procedure.
Computational Framework for and the Realization of Cognitive Agents Providing Intelligent Assistance Capabilities

Marcin Skowron, Joerg Irran, Brigitte Krenn

Abstract. The scope of the presented research covers virtual agents providing intelligent assistance capabilities for accessing and processing information from the Internet, domain specific databases and knowledge repositories. They receive natural language inputs and communicate findings to their users via a set of task oriented interfaces. Cognitive agents are conceived to evolve in response to the changes of interests, needs and preferences of the users and the alterations in their environment. We present a virtual embodied cognitive agents architecture, and a computational framework that allows their modular and flexible creation, based on a set of components. The framework supports the creation of an environment for multiple agents and provides communication mechanisms, used to share knowledge between the agents. The exemplary assembly of these building blocks to realize smart assistance applications further demonstrates the platform’s capacity to support development, instantiation and evaluation of collaborative cognitive agents.

1 MOTIVATION AND OBJECTIVES

Our motivation is to create virtual agents that provide personalized assistance to their users in finding and retrieving information from the Internet and other resources such as domain-specific databases and knowledge repositories. The developed virtual agents represent a growing class of cooperative agents that do not have a physical presence, but nevertheless are equipped with major ingredients of cognition including situated correlates of physical embodiment to become adaptive, cooperative and self improving in a virtual environment, given certain tasks. The design of the agents is partially inspired by embodied cognition originating from interaction-based robotics, transferred to a virtual context[5, 4]. This is persuade in the projects RASCALLI (Virtual Artificial Situated Cognitive Agents that Live and Learn on the Internet) and SELF (Advanced Knowledge Technologies: Grounding, Fusion, Applications). The presented work covers the following topics: development of a computational framework for realization of cognitive agents providing intelligent assistance capabilities, cognitive architecture and modeling, perception and action, reasoning, learning, communication, agent-to-agent and agent-to-user interfaces.

In the symbolic AI tradition, there has been a substantial work on Internet agents that perform pre-defined tasks on the Internet, including information retrieval and extraction tasks[7], checking for website updates, doing price comparisons etc. However, such Internet agents relied rather on extensive statistical analysis and on existing search engines than on the usage of cognitive architectures in an attempt to develop adaptive and flexible virtual agents. The Rascalli project aims at extending the state of the art by developing situated cognitive agents which inhabit the Internet environment. The investigation of the added value through the usage of cognitive architectures for improving the agent’s capabilities and the user experience in interaction with that agent is a further objective of the presented work.

This is feasible due to the modular approach used in the Rascalli platform, which supports multiple agent architectures and agent definitions, and therefore serves as a testbed for the evaluation and comparison of the agents. The major standards organization in the area of Agent Oriented Software Engineering (AOSE) is FIPA[3], which is concerned with the standardization and interoperability of multi-agent systems. JADE[4] and similar FIPA compliant agent platforms offer a strong middleware layer for distributed multi-agent systems, including agent lifecycle management, agent communication, as well as rich graphical tools for agent development but they do not meet major requirements of the RASCALLI platform. While the RASCALLI platform supports the execution of multiple agents, it is not a multi-agent system in the traditional sense, where agents are independent components of a larger application. Instead, Rascalli are complete individual entities that simply happen to exist in the same environment and are intended to communicate with each other. Furthermore, none of the aforementioned agent platforms supports the development style targeted by the RASCALLI platform, where multiple agent architectures and agent definitions, as well as multiple versions of agent components co-exist in a single platform instance. This development style is specifically geared towards research projects experimenting with alternative cognitive architectures, and combining them with a variety of action and perception tools as it is the case with the exemplary applications (see section 6) created and integrated in the Rascalii platform. The RASCALLI approach differs also from existing software systems for cognitive modeling such as AKIRA VI[5] or AmonI[6]. While the latter two provide specific means for modeling cognitive processes, the RASCALLI platform is a more general framework for implementing a variety of different models and architectures.

The rest of the paper is organized as follows: section 2 describes concepts and design principles of the cognitive agents providing intelligent assistance capabilities. Section 3 introduces the computational

1 Austrian Research Institute for Artificial Intelligence, Austria, email: marcin.skowron,joerg.irran,brigitte.krenn@ofai.at
2 http://www.cs.washington.edu/research/softbots
3 http://www.fipa.org
4 http://jade.tilab.com
5 https://sourceforge.net/projects/a-k-i-r-a/
6 http://www.cs.bath.ac.uk/ai/AmonI-sw.html
framework for realization of the cognitive agents as implemented in the Rascalli platform. Section 4 provides an overview of a bottom-up and top-down cognitive components used in the system. Section 5 introduces the concepts and components implemented in the platform, that are used to create different incarnations of Rascalli agents, presented in section 6. The last section provides the conclusions and presents the future work.

2 CONSTITUTION OF THE AGENTS

In the RASCALLI project the creation of virtual agents that provide personalized assistance to users in finding and retrieving information from the Internet and other resources such as knowledge repositories and domain-specific databases is pursued. The set of actions the agents are able to perform includes web and database search mechanisms, techniques for extracting information from social networks and news-feeds, techniques for extracting information from texts, logging of user interactions and recommendation of related content. The agents evolve in response to the changes of interests, needs and preferences of the users and to the alterations in the agents environment. Therefore the agents need a certain degree of autonomy and flexibility in their behavior. To achieve this, the agents have a biologically inspired virtual embodiment consisting of three layers (see Fig. 1), a perception layer with which they perceive (analyse) their environment, a cognitive layer with learning and reasoning capabilities, and an actuator layer which allows them to act on the environment employing a selection of processing tools. Everything outside the agent, including the user, Internet resources, local databases and other Rascalli agents is considered as a part of the environment. Due to the modular design of the platform (refer to section 3), a cognitive component can be realized using various approaches (refer to section 4) and different agents can be assembled using a selected set of action-perception and user-agent interface components (refer to section 5).

3 RASCALLI PLATFORM - A COMPUTATIONAL FRAMEWORK FOR COGNITIVE AGENTS

In the scope of the Rascalli project, various sets of action-perception tools as well as knowledge acquisition, reasoning and decision making mechanisms are being conceptualized, developed and tested. The Rascalli development platform[10] supports a flexible creation of individual virtual agents via assembly from a set of existing software components. The platform allows also multiple instances of agents to be run on a single platform instance, thus different incarnations of virtual agents can be exposed to the same environmental conditions, as well as developed and evaluated in parallel. The Rascalli project provides also several interface modules for the realization of agent-human interaction, including an Embodied Conversational Agent interface and other interfaces described in section 5.7. The interfaces are treated in the Rascalli platform in the same way as the other components. Since the components are isolated building blocks, they can be separately evaluated, according to what information they have gathered, what outputs they have produced to certain tasks or according to the the user satisfaction. Consequently, different system incarnations (including various implementations of a cognitive component) can be assembled and evaluated to find an optimal set of system components for a given objective (see section 6).

The design of the software architecture of the RASCALLI platform is determined by a number of requirements. They stem from the need for distributed development of individual components, the wish to run multiple Rascalli on a single platform, the need for a platform that supports a plug and play approach, and that supports the realization of and experimentation with different cognitive architectures/models/components. We additionally aim at a research setting, where the integration of existing components is preferable to reimplementation, and where system integration is likely to be a technically non-trivial task.

The RASCALLI platform supports the implementation of agents of different constitutions, structured in several layers (see Fig.2 for an overview):

**Figure 1.** Layers of the agents virtual embodiment and the interaction loop with the environment

**Figure 2.** Rascalli platform and its layers

- **Infrastructure Layer:** The infrastructure layer contains basic technologies and components the RASCALLI platform is build upon.
- **Framework Layer:** The framework layer contains general platform services and components available to all RASCALLI agents.
- **Agent Architecture Layer:** This layer defines the constitution of the general agents architecture (e.g. Mind-Body-Environment).
- **Agent Component Layer:** The Agent Components layer contains implementations of the components/interfaces defined by the Agent Architecture Layer.
- **Agent Definition Layer:** An agent configuration is an assembly of specific components of the Agent Component Layer.
- **Agent Layer:** The agent layer contains the actual individual agents.
Each agent is an instantiation of a specific agent configuration, based on a set of agent components defined within a specific agent architecture. An individual agent might have additional configuration beyond what is provided on the agent configuration layer such as an individual agent’s name, name of its user, etc. A more detailed description of the agents cognitive and other components is provided in the following sections.

4 TOP-DOWN AND BOTTOM-UP COGNITIVE CAPABILITIES

Considering the agents’ cognitive component, we follow two strands of knowledge acquisition and action selection for the agents: a low level bottom-up approach where an agent acquires knowledge based on logs from its interaction with the user and the other entities from its environment, and a theory driven (top-down) approach based on the DUAL/JAMBR cognitive model[8]. To realize an interaction based knowledge acquisition architecture for virtual environments, a virtual embodiment for our agents is created. They are equipped with a collection of sensor channels geared towards the particular environment, and a set of specialized software tools (actions) through which they interact with the environment. These are described in more detail in section 5. Both approaches have a specific characteristic and predispositions in the scope of their applications. While the data-driven approach is more flexible and robust, the theory-driven approach provides more fine-tune capabilities and accuracy for restricted domains. Our aim is to take advantage of the best properties from both approaches and employ the above presented features of the Rascalli platform, reconciling both strands in a hybrid model, by matching the structures emerging in the bottom-up approach with the ones implemented in a top-down fashion by the cognitive model[15].

The research on the cognitive component includes the following topics: knowledge acquisition process, impact of the design of the environment, the design and constitution of the agent, and the training situation, flexibility/robustness in task solving strategies, dealing with changing situations (dynamic environment, novel input, sensor channels, tools), scalability (with increasing complexity of sensor channels, tools and tasks).

5 BUILDING BLOCKS OF THE SYSTEM

In this section the concepts and implemented components are presented in more detail, including the principles of the agents cognitive components, their relations to the agents action-perception capabilities, strategies used for knowledge acquisition, user profiling and user-agent interfaces. In section 6 it is demonstrated how the selected building blocks can be assembled to achieve a particular goal.

5.1 Agent environment - virtual entities

The agent environment includes the Internet, databases, knowledge resources, the user and other Rascalli agents. The Rascalli agent perceives its environment (via a set of perception sensors, implemented as software tools - see 5.2) as a set of unique virtual entities with their own characteristics and properties. These include strings of written language originating from the user or extracted from HTML documents, markup tags, meta-data information about binary files, information about the accessibility of various Internet and local tools and resources, user feedback, etc. Therefore, the agent has to deal with a dynamic environment i.e. evolving content of the websites, permanent or temporary inaccessibility of Internet services, appearance of a new content or services, changes in the user preferences and interests, as well as the unrestricted natural language input from the user and the web-pages. Similarly, all the agents actions in its environment are performed on the above introduced set of entities (refer 5.4 for an extensive overview of the agents’ actuators). Since interaction-based learning is a bottom-up driven learning approach of increasing complexity, it necessitates a simplification of the initial environment of the Rascalli. This is achieved by providing a reduced set of resources, e.g. closed domain databases, selected RSS feeds and websites, as well as a reduced and simplified instruction-driven user input.

5.2 Perception layer - sensor channels of the agent

The aim of the perception layer is to provide the agents with the capabilities to perceive an input situation and to distinguish a set of features necessary for selecting an appropriate action (tool application), based on the similarities between input situations and the episodes, which the agent encountered before.

5.2.1 Concept of the classification driven perception

The perception layer contains all components that are used by agents to sense their environment. Those components are designed to allow classification driven perception. The perception layer implementation in the Rascalli platform includes the Input-Processing Tool, which is based on the classification of input data and recognition of the source of an input (e.g. user utterance, pdf document, web page, user feedback). Various classes and categories are for example assigned to the data originating from the user utterance such as named entities (locations, organizations, person names), part-of-speech tags, coarse-grained utterance classes (greeting, question, acceptance, rejection, etc.), question categories, syntactic parse trees. Similarly, various classes are provided for the other types of virtual entities such as file extension and size, the content of an HTML file (title, headers, links, etc.), the meta-data related to binary files (title, artist, album, width, frame rate, etc.), input class (user utterance; praise/scolding button value, data input from the Internet/databases), IO error message status from the action application, user feedback. The classes are assigned based on:

- information perceived directly from the environment (e.g. file extension, name of an artist originating from a music file header, input class, feedback)
- machine learning based classifiers developed or adopted for the agents requirements(Maximum Entropy[6] based utterance classification, Supported Vector Machines[19] based question classification[17])
- available NLP tools (e.g. parsers, pos-taggers, named entity classifiers)

5.2.2 Natural Language Input Processing

Natural Language Input Processing (T-IP) involves a set of sensor channels each of which allows the agent to perceive certain aspects of a natural language input situation. T-IP performs actions on entities from the agent’s environment such as part-of-speech tagging the user utterance, finding the question focus word or segmenting text. At the current stage of development, the tool provides the following information to the agent:
● utterance class (greeting, question, agreement, rejection, find-similar, other),
● question class (6 coarse grained categories: abbreviation, description, entity, human, location, number; and 50 fine-grained categories [12]),
● POS tags [1],
● question focus word [17],
● instances, concepts, relations from the Rascalli ontology,
● NP chunks,
● Minipar parse [13],
● Wordnet entries (antonyms, synonyms, hypernyms, coordinate terms, polysemy count, etc.) [2],
● DUAL interest [9],
● DUAL question [9],
● DUAL free-question flag [9].

5.3 Cognitive layer - from knowledge acquisition to action selection

To achieve intelligent assistance capabilities, an agent must have abilities to select appropriate actions in a given situation. It has to be able to find cues in an input situation that can be related to one or more possible action applications. In the current Rascalli system, these cues can be related to action selection via a rule based system, a top-down driven cognitive architecture, a bottom-up based knowledge acquisition and classification driven action selection.

5.3.1 Rule based action selection

The rule based action selection component 'Simple Mind' is a trivial implementation of a mind component, based on hard-coded action selection rules. These rules match to specific cues in the input data arriving from sensor channels. 'Simple Mind' extracts relevant information and passes this information on to the appropriate effector tool. Even though seemingly non-trivial behavior can be accomplished through a series of interactions of the Simple Mind and the available tools, the Simple Mind does not contain any cognitive aspects such as memory or learning.

5.3.2 Top-down driven action selection

The top-down action selection mechanism is implemented in the current version of the platform as the DUAL/AMBR cognitive architecture [8]. It includes a long term memory (LTM) where general and episodic knowledge is stored, and a working memory (WM) constituted by the active part of LTM, perceptual input and goals. The DUAL mind operates only on represented knowledge and has only a mediated connection to the body and the environment. Thus it contains a partial, selected representation of the environment at an abstract conceptual level and experiential memories related to specific episodes like organization of the interaction of a Rascalli agent with its environment, see [15] for more details.

5.3.3 Bottom-up driven knowledge acquisition

In the bottom-up approach, the Rascalli agents acquire knowledge about their environment by relating the input to their own action-perception capabilities. In this process the agents build their own "understanding" about the properties of the virtual entities that constitute the environment. The agents ground the knowledge about the entities by relating them to a set of actions applicable to those entities in a given input situation, as well as to the consequences of a given action application.

With the bottom-up knowledge acquisition, we aim at a robust mechanism which to a possible extent autonomously endows the agent with the knowledge necessary to perform its tasks. Since the agent deals with a dynamic environment as described in section 5.1, it is infeasible to provide ex-ante (e.g. through human expert knowledge) a full spectrum of knowledge required by the agent to perform its wide range of tasks and to adapt to the ever-occurring changes. Therefore we propose a learning mechanism that is sensitive to its environment, including the user activities within the system.

The major objective is the development of a cognitive component that enables the agents to gain knowledge, clearly different from human knowledge but grounded in interaction based self experience. This kind of grounded knowledge forms the basis for action selection within a Rascallo and allows modeling information exchange between Rascalli (see 5.5), as well as knowledge exchange between the Rascalli and their users (see 5.7).

Depending on the developmental stage of an individual Rascallo, tools are selected and executed arbitrarily, motivated by drives, or deliberately chosen based on the given input and previously made experience. The outcomes of the tools application on the environment (see 5.4) are also treated as an input to the sensor channels (see 5.2). In a way this is similar to a robot’s perception of the consequences of an action application. For each action tool ti an application space app is created over time that contains tool related interaction episodes. These episodes include the input situation short before the tool was applied and the outcome situation as perceived by the agent via its sensor channels. Feedback provided by the user (if available) is part of the outcome. Steps of the learning approach as ‘partitioning the episodes’, ‘extraction of relevant sensor channels’, ‘feature extraction’ and ‘characterizing involved entities’ [4] are applied to the application spaces to derive generalized input (ci0) and outcome characterizations (ci0) from the sets of stored episodes. The generalized input characterizations are - related to the affordances theory - cues for possible action applications. If a given input situation matches to one ore more of those generalized input characterizations (ci0), then the assumption can be made that this situation affords the application of the related tool ti resulting in the expected outcome (ci0). The set of input characterizations (I), the set of tools (T), the set of outcome characterizations (O) and their interrelation form the knowledge repository I-T-O. As this repository is growing during further exploration of the environment including user feedback, it allows the Rascallo to act more purposefully on future input.

5.3.4 Classification driven action selection

The classification driven approach is a foundation of the agent action selection mechanism. Based on the available knowledge, including the perception of an input situation (task description, set of available actions and resources, user feedback received previously, etc.) the agent finds a set of actions that can be applied to this input situation. The selection of a particular action is based on the similarities with other actions the agent had successfully performed in the past, i.e. received positive feedback from the user. The action selection classifiers are implemented as Maximum Entropy [6]. The models used for training the classifiers represents an input situation in terms of entities perceived by the agent and an action associated with it. The examples of the classification driven action selection application are e.g. a choice of a particular website which is returned as an answer to the user, based on the similarities between the current input and
The concept of active perception, includes tool usage to sense the agent previously experienced inputs that led the agent to decide to access a given website (e.g. wikipedia.org, news.google.com, youtube.com); a selection of a particular communication channel with the user (e.g. email, ECA, web-browser, music player) based on the type of data to be presented, or the user’s preferences and his/her status (online, offline).

5.3.5 Bottom-up knowledge acquisition scenarios

The Rascalli agent acquires the knowledge about its environment, the agents own capabilities, task solving strategies and the user preferences via the following mechanisms:

- learning via self-experience about the entities that constitute the agent environment in the relation to the agent capabilities,
- learning via exploration of the agents environment and the interaction with the user,
- learning in the training mode from examples provided by an expert (human or other Rascalli agent),
- learning via the communication with other Rascalli agents using grounded and agreed upon symbols.

5.4 Action layers

The Rascalli agents actuators are geared towards performing actions necessary to assist the user in accessing information from the Internet, knowledge databases and communicating the findings to the user. The agent actions are realized as software tools the agent is equipped with, including:

- actions related to accessing information from the Internet and the databases - Question Answering System (see 5.4.2), Natural Language Database Query Interface (see 5.4.1), various Internet site specific accessing tools (wikipedia.org, dictionary.com, youtube.com, news.google.com, etc.),
- actions related to communication with the user - Multi-Modal Generation Component (see 5.4.5),
- actions related to the agents perception7 (equivalent to active perception in robotic and human)- the extensive set of components integrated in the Perception Layer, (see 5.2).

5.4.1 Natural Language Database Query Tool

The Natural Language Database Query Tool (T-Nalqi) is used in the Rascalli platform for querying the databases accessible to the Rascalli agents, in a search for instances and concepts that can provide answers to the user questions. The component analyses a user’s natural language questions and retrieves answers from the system’s domain-specific databases. The tool comprises the following three sub-components:

1. A relation extraction component, which finds patterns of pos-tags that represent relational structures and their arguments.
2. A relation-to-DB mapping component which identifies relations or concepts that are contained in a relational DB.
3. A query generation component which generates SQL queries and retrieves results from the database, and post-processes the results.

If the mapping is successful a query is formulated and executed, retrieving results from the database, and post-processing the results. The tool triggers a signal which is perceived by the Rascalli system when an answer cannot be found or does not exist in the databases accessible to the Rascalli agents. The processing stages of the system include: question analysis, accessing the Internet resources, and analyzing the accessed documents. The system incorporates a number of natural language processing tools and resources (named entity recognition, part-of-speech tagger, text segmentation, chunker, stemmer, gazetteers, etc.), information retrieval tools (document indexing and querying engine) and machine learning based classifiers and clustering solutions. A set of tools usable to access a variety of Internet websites such as wikipedia.com, dictionary.com, howstuff-works.com, wordnet.org., and the Internet search engines such as Google, Yahoo, Altavista posses the capabilities to interpret the results of Internet resources, e.g. extract distinct definitions for a given term, report on ambiguity of a used term or possible misspellings, provide information on the number of available documents for a given query, distinct senses for a given term, the lack of term related documents or of a searched definition.

5.4.2 Question Answering System

The open-domain question answering system (T-QA) (which is based on the work described in [18], [16]) is used in the Rascalli platform to provide answers to the user factoid-type questions expressed in natural language. A typical use-case scenario involves a situation where an answer cannot be found or does not exist in the databases accessible to the Rascalli agents. The processing stages of the system include: question analysis, accessing the Internet resources, and analyzing the accessed documents. The system incorporates a number of natural language processing tools and resources (named entity recognition, part-of-speech tagger, text segmentation, chunker, stemmer, gazetteers, etc.), information retrieval tools (document indexing and querying engine) and machine learning based classifiers and clustering solutions. A set of tools usable to access a variety of Internet websites such as wikipedia.com, dictionary.com, howstuff-works.com, wordnet.org., and the Internet search engines such as Google, Yahoo, Altavista posses the capabilities to interpret the results of Internet resources, e.g. extract distinct definitions for a given term, report on ambiguity of a used term or possible misspellings, provide information on the number of available documents for a given query, distinct senses for a given term, the lack of term related documents or of a searched definition.

5.4.3 ChatBot

In the Rascalli project a wrapper to an existing ChatBot system was implemented. This component is based on the work undertaken in the A.L.I.C.E Artificial Intelligence Foundation8, which aims at the creation of a natural language ChatBot capable to engage in the conversation with a human. In the Rascalli platform the current JAVA based implementation of the ChatBot can be used to handle unspecific user utterances of the type ‘Are you a fish?’ or ‘Is it boring to be a computer?’.

5.4.4 RSS Feed Tool

The RSS Feed tool provides a mechanism for Rascalli agents to retrieve current information that might be of interest for the user. While technically RSS feeds have to be polled and retrieved (and thus obtaining information from an RSS feed is an active behavior), they can be easily modelled within the Rascalli system to be part of a dynamic and changing environment, with new feed items arriving in a manner that is temporally unpredictable for the agent. Thus a RSS feed tool which continuously (i.e. in very short intervals) polls and retrieves feeds that have been registered by a Rascalli agent without requiring any further intervention was implemented. As soon as new data is retrieved, the tool triggers a signal which is perceived by the Rascalli agent. The RSS feed tool includes a mechanism to filter news feeds for sets of keywords, allowing Rascalli to retrieve only relevant information of user interest based on user profiling (see 5.6).

5.4.5 Multi-Modal Generation Component

The Multi-Modal Generation Component provides a middle-ware functionality between generated agent output and the user interfaces. The generation component implements a template-based approach by encoding vocabulary, phrases, gestures etc. - which can be combined with the output of the Rascalli tools and context data - in the

7 The concept of active perception, includes tool usage to sense the agent environment.
8 http://www.alicebot.org
form of Velocity templates. The use of Velocity\(^9\), a template generation engine, allows to design and refine templates separately from the application code. For example the multi-modal generation component generates the speech, gestures and facial expressions for the Embodied Conversation Agent interface (see 5.7.1) used to communicate with the user. The output is encoded in an XML format that includes SSML (speech synthesis markup language) and BML\(^{[20]}\) (behavior markup language) markup, making it interpretable by the MARY speech synthesis and the Rascalli User Interface (Nebula Client\(^{10}\)).

5.5 Sharing knowledge - communication of grounded symbols between the agents

To enrich the task solving capabilities of the agents, which are equipped with symbol acquiring mechanisms, agent-to-agent communication was conceptualized in the Rascalli platform. This concept and its implementation allow agents to share their experiences (exchange knowledge) obtained through interaction with the environment[5]. Each agent has its own experience base comprising generalizations over the outcomes (O) of action applications, and generalizations of the input types (I) tools (T) that can be applied to. Due to the differences in their experience bases a negotiation process between Rascalli takes place establishing common labels for their individual knowledge of inputs and outcomes. Such an agreement process includes several cycles of exchanging prototypes and/or single episodes. For instance an agent A provides a prototype description \(d\) to another agent B, B tries to match \(d\) to its own experience base. Given a successful match, the agents choose a common label depending on the existence of labels from previous negotiations. In case the match at prototype level is not successful, the agents resort to instance level. This may lead to the creation of new prototypes in the agents which form a new basis for establishing a common label. If the negotiation at instance level fails, the agents’ current experience bases are too distant. This however may change with further acquisition of knowledge. Using a set of labels, agents can exchange information more efficiently than by exchanging prototype representations or instance data. Since the shared labels are grounded in each agent’s experience due to virtual embodiment and the affordance approach, knowledge exchange is possible even though the agents do not share the same internal representations. With the agreed labels, agents can exchange task solving strategies: which tools or tool chains to use in a given situation, how to react on a given input, or reach a desired outcome. For example, an agent may present a user input to another agent and ask for a recommendation what tool to use. This reduces the search space of individual agents for finding applicable tool or tool chain usage and increases the probability of satisfying the user.

5.6 User profiling

User profiling is based on logs from the user interactions with the system via selected user interfaces. The data are aggregated in a profiling component that allows the Rascalli agent to learn from the user actions about the interests and preferences of the respective user. These mechanisms are part of the agent assembly we term Smart Music Companion (6.3). The Smart Music Companion is an incarnation of a Rascalli agent that supports the user in finding and retrieving information on the popular music domain. Thus, the Music Companion is equipped with two special purpose tools/components which allow the user to browse a large music archive (currently up to 150 000 songs, see 5.7.2) and also access background information on the artists from a database comprising entries for approx. 15 000 individual artists and groups (5.7.3). Apart from these domain-specific interfaces, user action data are also collected from the ECA interface (5.7.1) which is the major user interface to the RASCALLI agent. Through this interface, the user may pose questions to her agent and evaluate the agent’s responses, as well as introduce specific URLs and RSS feeds the agent should monitor for the user. The logs of the user actions contain the following information: user X does Y on item Z at time T. They are part of the (implicit) user profile which is used by different generator components of the profiling system in order to provide the agent with up-to-date information of the kind USER X LIKES/IS INTERESTED IN Z. This is crucial information for the Rascalli to adapt to their users and provide their users with (new) information that might be relevant given the user profile. For instance, when retrieving information from the domain databases (5.4.1), the Rascalli agent first of all provides information that is high ranked according to the user’s current interest profile. Similarly, when applying the question answering system (5.4.2), first of all those web pages are analyzed which best fit the current user interests and/or have been explicitly introduced by the user to her agent. Combining the user interest profiles with an item-based recommender system that operates on the data from the music domain enables the Rascalli to propose new music/artists to the user. Moreover, the logs from the user actions may also be used as an input to the episode learning mechanism of the cognitive model DUAL (6.2).

5.7 Interfaces for agent-user interactions

In the following, we give an overview of the user interfaces which have been implemented up to date for the Rascalli agents.

5.7.1 Embodied Conversational Agent interface

While user input to the system via the ECA interface is constrained to free text input into a window of restricted size and pressing a praise and a scolding button (thumbs up/thumbs down), the system presents its output to the user via multi-modal, verbal and nonverbal behavior. See Figure 3, for a screenshot of the current design of the ECA interface, where the character sits in his fantasy room and waits for the user to type in her question. The output of the generation component is encoded in a BML-compliant format\([20]\), which is then interpreted by a 3D animation component built on top of the Nebula platform\(^{11}\). For speech synthesis the MARY TTS system\(^{12}\) is employed. Figure 3 shows the ECA interface together with the Rascalli web page through which the user may introduce URLs and RSS feeds to the agent, and from which the music exploration interface and the music browser interface may be accessed. Through these interfaces the following user actions are logged: 1. the user utterance (text string), 2. clicks on the praise and scolding buttons, 3. switches to the music exploration and browsing interfaces, 4. user specified URLs and RSS feeds.

5.7.2 Music exploration interface

Figure 4 shows the exploration interface to the music archive the Rascalli have access to. The user may enter her favorite artist, song

\(^{9}\) http://velocity.apache.org
\(^{10}\) http://nebuladevice.cubik.org

\(^{11}\) http://www.radonlabs.de/technologynebula2.html
\(^{12}\) http://mary.dfk.ie
or genre. The underlying application provides then a list of songs which belong to the selected artist or genre and computes for each song the user is interested in a list of songs that have similar acoustic properties. The best matching eight songs are displayed in a music map visualized in a 3x3 grid of album covers. Every suggested song can be listened to as a 30 second sample. The music player is invoked by either clicking on a cover or on an element in the play list. For each song played the user can state 'I like it!' (thumbs up) or 'No thanks!' (stop hand). From each song, there is a link to background information about the artist, which can be browsed via the music browser interface (5.7.3). All user clicks within the music exploration interface are assigned with a specific action label, logged and used as input to the profiling component (5.6).

5.7.3 Music browser interface

Figure 5 shows the music browser interface which allows the user to explore the background information on a certain artist or group available to the Rascalli. This information has been harvested from the Internet, employing a seed-based information extraction methods which uses the already known in order to extract new, related information for text documents[21]. The browser provides not only facts about artists but also a network view of people related to them. Again, all user clicks on the links in the interface are logged. Each link is assigned a specific action label. The logs are in put to the profiling component (5.6).

5.7.4 Jabber interface

A Jabber client interface (6) was developed as a simple user interface mechanism for the Rascalli system. It allows Rascalli agents to connect to the Jabber (an XML-based open source instant messaging protocol) network. Various Jabber clients provide an easy-to-use text-based interface by which the user can contact her Rascallo (the Rascallo can be added to the user’s contact list just like other - human - contacts), and can be notified or contacted by her Rascallo in an unobtrusive manner.

5.7.5 Command line client interface

A plain version of the agent-user communication interface is realized as a command line client. This type of interface is especially useful for system development and testing.

13 http://www.jabber.org
6 ASSEMBLING THE BUILDING BLOCKS - REALIZATION OF SPECIAL PURPOSE RASCALLI

Below we present how selected building block and interfaces presented in chapter 5 and section 5.7 were used and integrated in the Rascalli platform to create sample agents fulfilling particular goals.

Based on the Rascalli computational framework three different Rascalli configurations were conceptualized, developed and implemented. These are the initial implementation of a Rascalli agent with a basic set of tools (sandbox for testing the system components and their integration), a Rascalli agent utilizing an implementation of the DUAL cognitive architecture as central control unit, and a Smart Music Companion.

6.1 Sandbox for testing the system components and their integration

The goal of the sandbox application is to provide a method that allows to evaluate single components with respect to their feasibility for integration and their interoperability in the platform. For this purpose the building block 'simple mind' - a rule based action selection mechanism (see 5.3.1) has been developed. It allows the definition and application of a set of rules that describe how to react on a certain input/cue. Interaction with the user is handled via the ECA interface which is connected with the natural language input processing tool to analyse the user input and the multi-modal generation tool to produce the multi-modal output specification for the ECA. The command line interface (see 5.7.5) provides a fast and sufficient capability for developing and testing the system.

As a case-study example a particular system incarnation was assembled with the following building blocks: T-IP (see 5.2.2), 'Simple Mind' (see 5.3.1), User Profiling Component (see 5.6), T-QA (see 5.4.2), T-Nalqi (see 5.4.1), T-MMG (see 5.4.5), ECA interface (see 5.7.1).

In addition to testing system integration, it was possible to assemble a system that was capable to realize a wide range of actions, providing a user with relevant information to her input. Due to the smart characteristics of the perceptor and actuator components (T-Nalqi, T-IP, T-QA) this was achievable even with a fairly simple rule-based action selection mechanism.

6.2 DUAL cognitive architecture as a control unit for a Music Information Assistant

As a cognitive enhancement to the initial Rascalli configuration, (see 6.1) a system incarnation was created that incorporates a proof-of-concept implementation of the DUAL cognitive architecture[8] as its central control unit. This implementation was accomplished for a very restricted set of show cases in the music domain. The implemented system incarnation is capable of providing answers to music and gossip related questions of the following kind: "Tell me something about Britney Spears", "Who are the children of Madonna?" or "Do you happen to know who is Madonna married to?".

A case-study example for the particular system incarnation was assembled with the following building blocks: T-IP (see 5.2.2), T-QA (see 5.4.2), T-Nalqi (see 5.4.1), T-MMG (see 5.4.5), ECA interface (see 5.7.1).

The goal of this Rascalli incarnation was to explore and further develop the cognitive model in order to prepare for reimplementation of selected cognitive functionality in a way that the system is able to scale up.

6.3 Smart Music Companions

The Smart Music Companion incarnation of the Rascalli agent provides information from the music domain, learns about user preferences, and uses this knowledge to present the user with new information that is related to her preferences[11]. The particular environment a Rascalli agent has to deal with in this application consists of: external knowledge sources, in particular the Internet and some domain-specific knowledge bases and the user. The application domain is popular music. Apart from the Internet, the Rascalli have access to the following two domain-specific knowledge resources: (1) a database with up to 150 000 tracks annotated with meta-information such as track name, album name, genre, artist or group name; (2) a database comprising background information of artists and groups such as band members, personal relations of artists, etc.

As case-study example for the particular system incarnation was assembled with the following building blocks: T-IP (see 5.2.2), 'Simple Mind' (see 5.3.1), User Profiling Component (see 5.6), T-QA (see 5.4.2), T-Nalqi (see 5.4.1), T-MMG (see 5.4.5), Music Exploration Interface (see 5.7.2), Music Browser Interface (see 5.7.3), ECA interface (see 5.7.1).

The user interacts with the companion via two kinds of interfaces: 1) The ECA interface where face-to-face dialog interaction between the embodied conversational companion and the user takes place. 2) The special purpose interfaces Music Explorer and Music Browser which support the user in exploring the Rascalli music databases. The special purpose interfaces are an addition to ECA for the following reasons: 1) easiness to obtain information about the user preferences; 2) easiness and naturalness in providing unambiguous information to the system in an explicit way, which would be hard to achieve via the ECA interface due to erroneous natural language processing.

7 CONCLUSIONS

The RASCALLI computational framework introduced in this paper allows to flexibly add, remove and modify components according to the needs of a particular assembly of a system application. Due to the platform modularity and its well defined interfaces the integration of existing components is possible, and even preferred to the creation and re-implementation of new ones. In this way the platform can be continuously updated and enhanced, matching the state-of-the-art developments in intelligent artifacts. As demonstrated above, the platform provides already a sandbox for testing new and existing components and their interoperability. This development style is specifically geared towards research projects experimenting with alternative cognitive architectures, and combining them with a variety of processing and generation tools. The presented work demonstrates that the platform and its components support a prompt and purpose-driven creation of various incarnations of agents, where different cognitive components, action-perception tools and interfaces can be realized. The cognitive component can incorporate a wide range of approaches, including bottom-up driven, top-down driven or combined cognitive models. The Multi-Modal Generation Component as a middle-ware layer allows the usage of various user-agent interfaces without the need of modifying the agents output. This allows to operate in different contexts and to realize various agent appearances, different interaction behavior capabilities, etc.
The presented computational framework also provides an environment for a simultaneous existence of multi-agents, ranging from multiple instances of one agent configuration to multiple instances of agents assembled with different components. The agents have individual users and due to their unique interaction history their experience bases deviate from each other. As a result the collective experience base is diverse and covers a broader range of knowledge. To enable an agent to take advantage of this distributed knowledge, a method for communication between the agents in the platform was conceptualized. In addition the multi-agent platform provides the basis for the evaluation of the fitness of particular incarnations of agents, and thus helps to identify those agents that solve a given problem more effectively and purposefully than others. It also helps to explain why some agent types are more successful in their environment. The presented action-perception components are used by the agents to perceive their environment and perform actions on their environment such as accessing and processing information from the Internet and knowledge repositories. The three presented application scenarios exemplarily demonstrate the modularity, flexibility and integration capabilities of the processing tools and the platform.

Future work includes further enhancement of the platform, extended integration between the top-down and bottom-up cognitive approaches at a conceptual level and at execution level. The characteristic of the presented computational framework also enables further research on the goal orientated collaboration of multiple-agents and the usage of various communication strategies that facilitate distributed task solving.

ACKNOWLEDGEMENTS

This research is supported by the EC Cognitive Systems Project IST-027596-2004 RASCALLI, by the national FFG programme 'Advanced Knowledge Technologies: Grounding, Fusion, Applications' (project SELP), and by the Federal Ministry of Economics and Labour of the Republic of Austria.

REFERENCES