Training on the Job — Collecting Experience with Hierarchical Hybrid Automata

Alexandra Kirsch and Michael Beetz

Technische Universität München

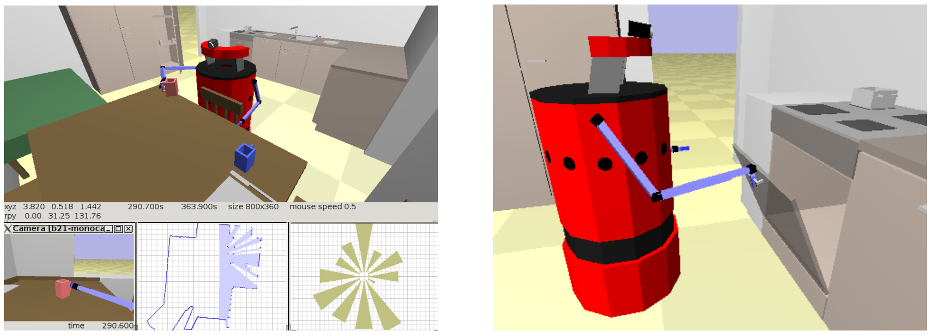
Abstract. We propose a novel approach to experience collection for autonomous service robots performing complex activities. This approach enables robots to collect data for many learning problems at a time, abstract it and transform it into information specific to the learning tasks and thereby speeding up the learning process. The approach is based on the concept of hierarchical hybrid automata, which are used as transparent and expressive representational mechanisms that allow for the specification of these experience related capabilities independent of the program itself. The suitability of the approach is demonstrated through experiments in which a robot doing household chore performs experience-based learning.

1 Introduction

Our vision is to build general purpose service robots that do not have to be preprogrammed for every task variation and every new environment. Such robots should be capable of improving and adapting themselves doing training on the job. If their jobs are complex activities such as household chore they have to do these optimizations in little training time and with sparse experience. In such settings being effective in the collection of experiences is an essential precondition for successful robot learning. To achieve training time efficiency robots must decompose their overall learning task into sets of modular and specific learning tasks that address detected flaws in their behavior. Learning task specific experiences for these subtasks that capture the interaction of the robot's behavior and the context conditions have to be collected concurrently. Finally, collected experiences must be stored, managed and transformed into abstract descriptions that speed up the learning process.

An instance of this vision is a household robot (see figure 1) equipped with default plans for setting the table, cleaning etc. It has basic skills for manipulating and navigating as well as default plans, general purpose plans that are developed to be applicable in a variety of environments and to be stable and robust in most cases, for its high-level tasks. However, the plans can fail or operate suboptimally if the robot's lower-level routines are not adapted to the specific environment. For example, when objects are to be taken out of cupboards or drawers the robot might encouter difficulties in a new kitchen, e.g. when it hasn't positioned itself appropriately for the manipulation task.

The robot can recognize such failures and locate its source and execution context inside the program. So when it detects that the manipulation task fails strikingly often, it puts it on the agenda of problems that have to be learned. The experience for all of those



```
xyz   3.820   0.518   1.442          290.700s      363.900s    size 800x360    mouse speed 0.5
rpy    0.00   31.25   131.76
```

Fig. 1. Kitchen scenario with a simulated B21 robot equipped with a camera, laser and sonar sensors.

unsolved learning problems are then gathered while it performs its household chores. One of the difficulties here is that experience sometimes is only desired in a certain context. For example, when the robot detects that the navigation routine performs different while it is cleaning the floor (presumably because the floor is wet), it is only interested in experience about the navigation routine while cleaning the floor. Other navigation tasks are of no interest at all in this situation.

As learning takes place at various levels of abstractions, it does not suffice to simply log all sensor data and control signals. The learning systems also need information about the program state, such as variable values. For example, to learn informative causal models of gripping actions the learning mechanism might need information about the scene — its clutteredness and the existence of reaching corridors. Or it might need information about which hand was used and why, or the position where the robot intended to grasp the cup. This data is learning task specific and typically not contained in the low-level sensor and control data.

To account for the robot's behavior being a result of its interaction with the environment, meaningful learning experience can only be obtained with a context-specific experience acquisition. We also need a compact and powerful behavior abstraction by segmenting continuous behavior into states that are meaningful with respect to the learning task. Furthermore, the abstraction of experience for learning should be possible on different levels.

In this paper we propose hierarchical hybrid automata (HHA) as a basic mechanism to fulfill these criteria. We propose HHAs to collect experience and learn from it that allow for

– the modular specification of experience collection independent of primary activity. This is necessary, because the interesting points in the control program can be distributed within the code and because our control program is modified by plan transformations.
– anchoring experience collection into the control program such that program execution automatically effectuates state transitions in the automaton.

– the specification of the problem specific data to be recorded, such as properties of objects being grasped or context conditions such as clutter.
– abstraction of behavior into learning problem specific information, for example the duration of the activation of a state or the number of failures encountered.

Our approach is implemented in RPL (Reactive Plan Language) (McDermott, 1993), a programming language for reactive plans implemented as a set of LISP macros. It provides high-level programming constructs like loops and conditionals as well as process control (parallel execution, mutual exclusion, etc.). The constructs for acquiring experience are embedded in a language called RoLL (Robot Learning Language), which is based on RPL.

The rest of the paper is organized as follows. First we give an informal overview of our procedure before we explain the more technical aspects of our approach, which comprises an introduction to hierarchical hybrid automata, the anchoring of these automata in the control program, the differentiation between "data " and "experience", and the integration of HHA in the whole learning process. After that we give experimental support for our proposition that experience acquisition independent of the primary control program plays a vital role in robot learning. Finally, we present related work and a conclusion.

2 Overview

In this section we describe our general approach of acquiring experience and learning in an informal way. Technical details are explained in the next section. For an illustration see Figure 2.

The first step is to define the required experience by specifying execution episodes and data associated with it. We model this information as an experience automaton, which is a hierarchical hybrid automaton where the episodes are modeled as discrete states with continuous behavior.

For not only specifying, but really getting the desired experience, the experience automaton must be anchored to the control program in such a way that it starts accepting an interpretation stream when the program task matching the automaton starts. Because the behavior that generates the episodes emerges from different parts of the program, it is not practicable to add the automaton code at the specific places in the program. Rather, we run a process parallel to the primary activity that is anchored in the control program and in this way can detect state transitions and observe the desired data.

The automaton returns experiences, i.e. data traces structured along the definition of the specified automaton. These experiences contain all information for understanding the interaction of the program and the environment needed for the respective learning task. In Figure 2 the experience data is structured according to the automaton structure and possible data sources are the robot's internal variables (indicated as bars) and external variables (indicated as zigzagging lines). After the acquisition, the experiences are stored in a database which facilitates the retrieval, but also purification of the data with data mining tools.

Before starting the learning process, the experience must usually be converted to more abstract features. This can be regarded as a transformation of the hierarchical

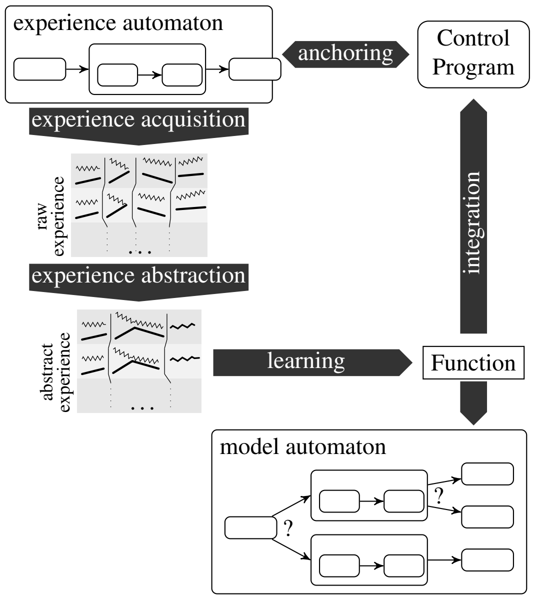


Fig. 2. Overall procedure for learning with RoLL.

hybrid automaton of the experience acquisition process to a more abstract form. Figure 2 illustrates this by showing the possible abstraction steps, changing the automaton structure (the data of the two automata in the middle are combined into one) and combining data (as in the data of the third automataon, where robot and environment data is merged).

Finally, the whole learning process can be regarded in the light of HHA. We can build models of the robot's control program with probabilistic hierarchical hybrid automata (PHHA). PHHA are HHA with the difference that state transitions are not activated by conditions, but rely on a probability distribution. After learning from the experience, the model experience can be replenished with the probability distributions of state transitions.

With the techniques presented here, we can for example observe the activity of lifting an object whose weight is greater than 2 kg. The respective automaton would be active as soon as a gripping action starts and the gripped object fulfills the specification. During the execution of the action, several interesting pieces of data can be observed, e.g. the required time, failures or if both grippers are used or only one.

3 Technical Approach

In this section we explain our approach in more detail. First, we define hierarchical hybrid automata. We then show how HHA are anchored to the control program, so that they can be used for defining experience. What we mean by experience compared to data is explained thereafter. We conclude by describing the rest of the learning process in the context of HHA.

3.1 Hierarchical Hybrid Automata

A hybrid system includes continuous change as it is observed in environment variables and discrete change, like the actions performed by a robot. The concept of hybrid systems and analytical methods are well-defined and understood(Alur et al., 1993; Alur, Henzinger, and Ho, 1996). We use a hierarchical notion of hybrid automata similar to the one presented by Alur et al. (2001).

Here we define a *hierarchical hybrid automaton (HHA)* as a tuple $\mathcal{H} = \langle V, flow, M, T, act, cond\rangle$, where V is a finite set of variables, $flow$ is a function describing the continuous change inside the automaton, M denotes a set of modes or sub-automata, T is a transition relation between modes, act is an action function effecting changes in the variable assignments, and $cond$ is function assigning a predicate to each transition in T.

The state of an automaton is described by the current program state (the set of currently active (sub-)automata) and the current data state (the current variable assignment).

Figure 3 shows a hybrid automaton $\mathcal{H}$ with two subautomata $\mathcal{M}_1$ and $\mathcal{M}_2$. The automaton has two variables $V = \{var_1, var_2\}$. The flow function is given for each automaton as well as the action and condition functions for the transitions.

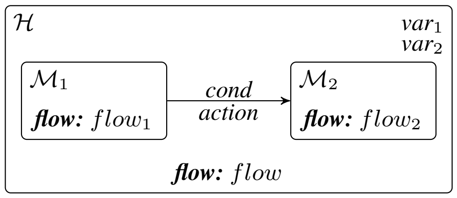


Fig. 3. Hierarchical hybrid automaton.

The hierarchical structure of the automata is not only an appropriate tool to describe robot behavior, but also enables us to provide this description on an arbitrary level of abstraction. The behavior of the automaton $\mathcal{H}$ can either be described by the compositional behavior of its subautomata or by its flow function alone. The flow function of an automaton can always be regarded as the result of the flow functions of its subautomata and their interactions, albeit it might not be computable.

3.2 Anchoring of HHA

In order to use HHA as a means of specifying experience, we need to anchor the automaton in the control program. This means that certain changes in the control program must effectuate a state transition in the automaton.

Figure 4 illustrates the two possible sources for state transitions in automata: (1) changes inside the program like the beginning or end of a subplan; and (2) changes in the environment. For implementing these anchoring mechansims we need means of detecting and reacting to environment changes and we must have insight into the control program and its current execution status.

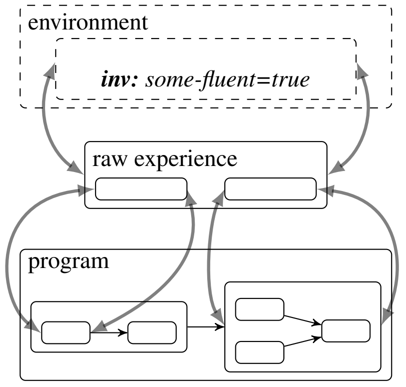


Fig. 4. Description of the HHA structure for raw experience acquisition. The arrows indicate how the experience automaton can be anchored in the automaton defined by the robot program and in the environment process.

Both requirements are provided by RPL. Continuous flow of environment variables is captured in so-called *fluents*. Fluents are variables whose continuous change can be noticed by program constructs such as (**wait-for** {conditions}), which blocks the process until the fluent *condition* is true.

RPL plans or programs run in tasks, which can have subtasks, for example the construct (**seq** a b), which executes a and b sequentially, represents a task with two subtasks. The task network is available in every part of the program and can be traversed at runtime by following links between sub- and supertasks. Thus, arbitrary tasks can be addressed by specifying the path from the current to the desired task. Another way is to label tasks with an explicit name and address them directly by this identifier.

In the language we propose, both kinds of anchoring — in the world and in the program — are supported. Anchoring in the world can be obtained direcly by specifying an RPL fluent, for example

```
(person-in-kitchen-automaton
  :invariant {person-in-kitchen})
```

For anchoring in the control program, the possibilities of RoLL exceed those of RPL in that not only unique tasks can be addressed, but also classes of tasks occurring several times, like the execution of a certain plan:

```
(plan-automaton
  :invariant (:plan-execution grip))
```

As we have defined hybrid automata in a hierarchical way, we can nest automaton specifications arbitrarily:

```
(place-object-automaton
  : invariant  (: plan-execution place-object)
  : children  (( grip-automaton
                 : invariant  (: plan-execution grip ))
               ( release-automaton
                 : invariant  (: plan-execution release ))))
```

3.3 Experience

We consider experiences not just as a set of data, but as a data trace of an HHA. In this way we preserve the execution context of the acquired data and thus make it more informative. After showing how to anchor HHA to control programs, we will now present how data is associated to HHA.

Figure 5 shows how data can be associated with a hierarchical automaton structure. Every automaton has two events: its beginning and its end. Data can be obtained at any of these events. Furthermore, we can record data continuously while an automaton is active.

The available data consists of the global variables including state variable fluents, local variables in active RPL processes and arbitrary combinations or abstractions of this data. For easier access later on, each recorded piece of data is provided with a unique name.

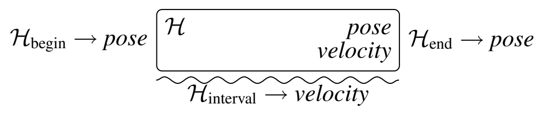

$\mathcal{H}_{\text{begin}} \rightarrow pose$
$\mathcal{H}$
$pose$
$velocity$
$\mathcal{H}_{\text{end}} \rightarrow pose$
$\mathcal{H}_{\text{interval}} \rightarrow velocity$

Fig. 5. Visual illustration of experience data acquisition. The robot's pose is to be recorded at the beginning and end of execution and during the execution interval the robot's velocity is recorded continuously.

As an example, let us have a look at the experience acquisition for learning the hand to be employed for gripping. Some relevant information might be the original robot pose, the object's original pose, the hand used by the robot, the goal pose of the object, the time needed for performing the sub-tasks and the whole task, and the accuracy the goal position of the object is reached with. We specify the data information within the hierarchical structure of the automaton specification:

```
(place-automaton
  : invariant  (: plan-execution place-object)
  :begin
    (( object-goal  (pose (: local  goal place-automaton)))
     (used-arm (:local  hand place-automaton)))
:children
  ((grip-automaton
    :invariant  (:plan-execution grip)
    :begin
      ((robot  (pose {robot}))
       (object  (pose (:local  object  place-automaton)))
       (time {timestep}))
    :end ((time {timestep})))
   (release-automaton
    :invariant  (:plan-execution release)
    :begin ((time {timestep}))
    :end
      ((reached (pose (:local  object  place-automaton)))
       (time {timestep })))))
```

3.4 Learning in the Context of HHA

After defining an experience automaton, its anchoring in the program and the data to be acquired, RoLL generates code that is run in parallel to the control program and experience is acquired every time the automaton is active, triggered by events in the control program and in the environment.

The experiences collected in this way, however, are usually not suitable for learning directly. In order to get a good generalization the data must be transformed into an abstract feature language. In the context of HHA this abstraction can be formulated as a transformation of automata. This contains splitting and combining modes from the original experience automaton and the calculation of feature values out of raw experience data (for an illustration see figure 2).

Another issue apart from abstraction is the storage of experiences. Experiences as we have seen can come in a variety of abstractions (there can be several abstraction steps before learning). Intermediate-level abstractions are often suited for different learning problems, so that it is often useful to store them for later use in different problems. Figure 6 illustrates this procedure.

We store experiences in a database (instead of a file for example) for several reasons. The first is that the data is better accessible, especially if only parts of the stored experience are required. Secondly, we can employ data mining algorithms for cleaning and filtering collected experience. Finally, in a database we can manage experiences over longer time scales. It is possible to add new experiences or remove old ones, and with the help of time stamps experiences can be classified by their reliability, giving new ones more weight than older ones.

After abstracting the experience, the learning process can take place. RoLL is not designed for a special learning algorithms, so that arbitrary learning systems like neural networks or decision trees can be used. The output of the learning process is a functions that is integrated into the control program and can be used at once.

The resulting function can also be interpreted as a state transition function of a probabilistic hierarchical hybrid automaton (see figure 2). A PHHA is the same as an HHA, but the transition condition function is replaced by a probability distribution giving the

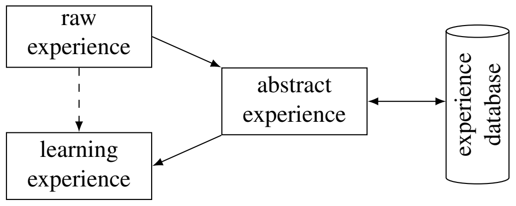


Fig. 6. Experience processing steps. Here only two abstraction steps are shown (raw to abstract and abstract to learning experience), but more steps are possible.

transition probabilities to the successor nodes. This PHHA is a model of the robot's behavior while acting in the determined environment.

4 Evaluation

In the introduction, we motivated our work on experience acquisition by the task of learning on a robot during its usual activity with scarce experience. In contrast to our proposed paradigm, most robot learning is done with artificially acquired data. One of the reasons is that special code for data acquisition must be added, so only a small piece of code is adapted to data acquisition, then the data is recorded and the learning is performed, so that the acquisition code can be removed and the control program remains readable.

We ran an experiment to show the benefits of learning from experience acquired during the robot's activity in contrast to artificially obtained data by learning the time prediction model of our kitchen robot's navigation routine. The experiment was performed in iterations defined in the following way:

1. perform the plan "set the table", thereby acquire learning experience (one run includes about seven navigation tasks)
2. record experience of 30 randomly generated navigation tasks
3. train two regression trees: one with the "natural" and one with the artificial experience
4. perform the plan "set the table" again, thereby comparing the two learned models with a programmed prediction function

Figure 7 shows the first 15 iterations of this procedure. The line with vertical bars shows the results of the programmed prediction function. Note that this line would be exactly horizonal if the robot's actions were deterministic. But we see that the navigation tasks and the time needed to fulfill them in the plan vary slightly. The curve with triangle-shaped dots indicates the result of learning with artificially generated experience. With a training set of 450 experiences, the learned model is significantly worse than the programmed model. The line with diamond-shaped dots indicates the result of learning with experience observed during plan execution. Here the quality of the prediction is about the same or slightly better than the programmed one after only 15 iterations, i.e. 15 observations of setting the table or 105 training examples.

After 15 iterations we replaced the set the table plan by a plan for cooking. This plan contains more navigation tasks of a very short range, which are uncommon in setting the table. As expected, the error of the regression tree that was only trained with experience from the set the table plan was very high when presented with the new plan. But after about 15 more iterations the experience of the new plan was integrated in the regression tree so that it could again beat the programmed model. Not surprisingly, the change in plan didn't cause any performance decay for the other regression tree. In the meantime, we had acquired 900 experience sets of randomly generated navigation tasks. But still the performance could not compete in the least with the programmed model or the learned model with realistic experience.

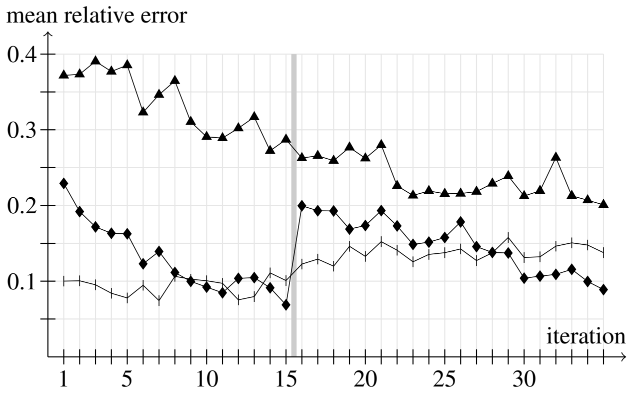


Fig. 7. Comparison of learning with artificial experience in contrast to experience acquired during normal execution. The line with the vertical bars indicates the result of a programmed function. The line with the diamond-shaped points is the result of learning with natural experience, whereas the line with triangle-shaped points gives the error of the learning process with artifically generated experience.

These results show that it is advisable to draw learning experience directly from the normal activity. Because the control programs are very complex for sophisticated robot activity it is not practicable to introduce special code for experience acquisition. The RoLL features for experience acquisition solve this problem by adding the acquisition code without change of the control program.

5 Related Work

Currently, data for learning is collected in two ways: (1) protocolling continuously every sensor input and control output during the program execution or (2) adding special code to the control program that specifies which data is to be recorded when.

The first approach is put into practice in the XAVIER project (O'Sullivan, Haigh, and Armstrong, 1997) of Carnegie Mellon University. Here all the data is recorded at runtime and replayed for analysis later. This approach is not feasible for agents with many state variables because of the high storage demands. More importantly, the context information and the robot's local variable values are lost, so that intermediate-level experience like which position the robot intended to head for in order to grasp an object.

The Common Lisp Instrumentation Package (CLIP) (Anderson et al., 1994) is a LISP package facilitating the second approach, i.e. adding extra code for every kind of experience. This approach is not adequate for our robot, which changes its control program by transforming its plans. The first problem would be that the program is harder to understand and therefore harder to transform because of the code pieces that don't have any influence on the plan's performance. The second drawback is that the acquisition code might be lost or mutilated during the plan transformation.

Reinforcement learning (RL) has been used to learn complex tasks embedded inside the agent program. In this paradigm complex problems are not analyzed and taken apart, but learned as one problem. This makes the solution an opaque part of the program that cannot be modified further. An instance of RL was presented in (Geipel and Beetz, 2006), where an attack of the opponent goal involving several players in robotic soccer is learned by means of RL. Because the RL algorithm has no semantic information about the actions involved, all actions are weighed equally. Therefore the learning process needed a very large number of experiences and the results varied considerably in different experimental runs. In contrast, we would collect experiences of situations when the robot has played passes and has scored goals successfully. With these experiences we would learn models that answer questions like "How should a pass be played?" or "In which situations can passes be played most effectively?"

The modeling of embedded systems with hybrid automata was explored by Williams et al. (Williams et al., 2003). In this work, the behavior of space explorers is modeled as a hybrid system, which is then used to detect and prevent failures during execution.

Fox and Long (2006) use hybrid automata for the definition of the plan language PDDL+, because they provide a rich, well-under stood framework for modeling continuous behavior.

6 Conclusion

Whereas a lot of work is done on robot learning, embedding learning into the operation of autonomous agents has received surprisingly little attention. It will not be possible to build autonomous robots acting for a longer period of time in demanding environments without prividing them with integrated learning mechanisms, so that they will be able to acquire their experiences when needed and to learn with little experience.

No one would attempt to solve a task like cooking pasta as one monolithic problem, because the state space is gigantic. But complex tasks in real-world scenarios cannot be achieved in a stable and robust way without learning. The solution is to devide the problem in small sub problems and learn those (or at least parts of them). A consequence of this procedure is that many learning problems have to be solved with a small number of experiences. However, solving large numbers of learning problems is only feasible by automating the process including experience acquisition and integrating the learning results into the control program. Besides, a service robot will not have the time to gather experience for every learning problem separately. Instead, the experience must be obtained during its normal operation.

These requirements make automatic experience acquisition a hard problem. The main challenges are recognition of interesting experience, specification of experience without modifying the main code, feature extraction for learning, and storage and management of experience.

In this paper we have presented an approach to specify experience in the framework of hierarchical hybrid automata. We have shown how automata are anchored in the control program so that there is no need to modify the main code. Nevertheless, the automaton has access to all local variables in the main program, thereby making it possible to access intermediate-level information about the robot's intentions and internal state and beliefs. Running in parallel to the primary activity, experience is gathered and stored automatically. Before the learning process starts, experience is abstracted, again in the context of HHA. The final process of learning and integrating the result has been mentioned briefly.

We have presented empirical evidence that learning small problems during the robot's normal activity leads to better learning results and thereby better robot behavior.

References

1993. Alur, R., C. Courcoubetis, T. A. Henzinger, and P-H. Ho (1993). Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. *Lecture Notes in Computer Science* 736: 209–229.

1996. Alur, R., T. Henzinger, and P. Ho (1996). Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering* 22(3): 181–201.

2001. Alur, Rajeev, Radu Grosu, Insup Lee, and Oleg Sokolsky (2001). Compositional refinement for hierarchical hybrid systems. *Lecture Notes in Computer Science* 2034.

1994. Anderson, Scott D., David L. Westbrook, David M. Hart, and Paul R. Cohen (1994). *Common Lisp Interface Package CLIP*.

2006. Fox, Maria and Derek Long (2006). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research* 27: 235–297.

2006. Geipel, Markus and Michael Beetz (2006). Learning to shoot goals, analysing the learning process and the resulting policies. In Lakemeyer, Gerhard, Elizabeth Sklar, Domenico Sorenti, and Tomoichi Takahashi, editors, *RoboCup-2006: Robot Soccer World Cup X*. RoboCup, Springer Verlag, Berlin. to be published.

1993. McDermott, Drew (1993). A reactive plan language. Technical report, Yale University, Computer Science Dept.

1997. O'Sullivan, Joseph, Karen Zita Haigh, and G. D. Armstrong (1997). *Xavier*.

2003. Williams, Brian C., M. Ingham, S. H. Chung, and Paul H. Elliott (2003). Model-based programming of intelligent embedded systems and robotic space explorers. *Proceedings of the IEEE: Special Issue on Modeling and Design of Embedded Software* 9(1): 212–237.