Towards High-performance Robot Plans with Grounded Action Models: Integrating Learning Mechanisms into Robot Control Languages

Alexandra Kirsch

Abstract

For planning in the domain of autonomous robots, abstraction of state and actions is indispensable. This abstraction however comes at the cost of suboptimal execution, as relevant information is ignored. A solution is to maintain abstractions for planning, but to fill in precise information on the level of execution. To do so, the control program needs models of its own behavior, which could be learned by the robot automatically. In my dissertation I develop a robot control and plan language, which provides mechanisms for the representation of state variables, goals and actions, and integrates learning into the language.

Motivation

A key challenge for the next generation of autonomous robots is the reliable and efficient accomplishment of prolonged, complex, and dynamically changing tasks in the real world.

One of the most promising approaches to realizing these capabilities is the plan-based approach to robot control. In the plan-based approach, robots produce control actions by generating, maintaining, and executing plans that are tailored for the robots' respective tasks. Plans are robot control programs that a robot can not only execute but also reason about and manipulate. Thus a plan-based controller is able to manage and adapt the robot's intended course of action — the plan — while executing it and can thereby better achieve complex and changing goals. The use of plans enables these robots to flexibly interleave complex and interacting tasks, exploit opportunities, quickly plan their courses of action, and, if necessary, revise their intended activities.

Making plan-based controllers effective requires programmers to abstract away from details of the physical world. In order to reduce the size of the state space, the robot's belief state is described in terms of abstract situations and events. Similarly actions are described as discrete, instantanious events. As an example let us consider an autonomous household robot. A situation can be described as the robot being "near the door". When someone wants to enter the room the robot should perform the action "clear the door". A description like this totally disregards the actual position of the robot, the actual distance to the door and the precise position to where the robot should go in order to clear the door. These are the kinds of abstraction that make automatic planning feasible.

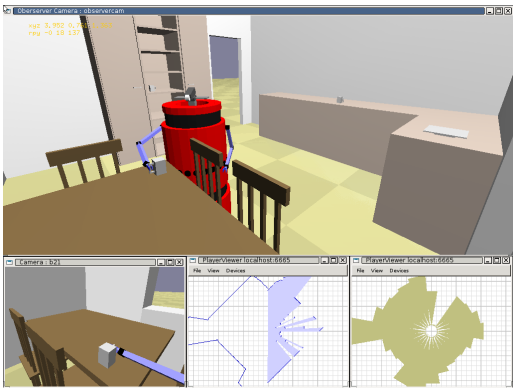

Figure 1: Realistic simulation for a household robot.

However, abstracting away from the low-level state description often yields suboptimal behavior. In our example where the robot is blocking the door, it might move away from the door so that it can be opened, but it might still be in the way of the person entering the room. Or the robot might have been standing near the door, because it was stirring the soup on the hearth. If it moves away to make room for someone to enter, the soup might be burning. In this situation, the robot should have looked for an alternative position that still allowed it to reach the hearth. The problem here is that the action "clear door" deliberately ignores the robot's precise start and goal positions. For the planner, this is fine, since the actions should be kept simple. But when it comes to executing the plan, the robot should consider its current situation, goals and possible outcomes of its action.

In my research I develop mechanisms that allow the programmer to keep a high degree of abstraction for planning. Only during the execution of the abstract plans, the low-level details are taken into account and the plan steps are optimized with respect to the current context. To perform an action in a certain situation the control program needs information about (1) why the action is to be performed, (2) other goals that might interfere, and (3) the behavior of the procedures used for achieving the action.

The information about the current program state can be

provided by a declarative structure of the control program making concepts such as goals, procedures and belief state explicit. Knowledge about the interference of goals or the behaviour of routines is provided by models. In the case of interfering goals the models might be provided by the programmer. But it would be a hard job to predict the behavior of every available control routine in every conceivable situation. Here automatic learning techniques are indispensable.

In this framework an action in a plan doesn't necessarily correspond to a certain control routine. In many cases, there are several routines, the performance of each varying in different contexts. The choice of which routine to call in the given situation is based on models. Although it is possible to program all these routines by hand, program development could be advanced by learning routines automatically.

Unfortunately, the performance of learned routines often drags substantially behind those of programmed ones, at least if the control tasks are complex, interact, and are dynamically changing. In our opinion this is not due to a poor performance of learning algorithms, but to the insufficient integration of learning into control languages. With a synthesis of learning and programming, parts adequate for learning need not be programmed explicitly, while other parts can be implemented manually. In order to get a smooth transition between the two worlds, the learning process must take place inside the controller. This means that the robot has to acquire training experiences, compute a suitable feature language representation, execute the learning process and integrate the result into the control program.

Contributions

The aim of my dissertation is the development of an extension of a robot control and plan language, which provides mechanisms for

- modelling interaction with the physical environment;
- the representation, inference and execution of abstract modalities like state variables, goals and actions;
- the smooth interaction of programming and learning.

For the first point I develop representations that provide the program with information about the physical meaning of state variables. Inference mechanisms can use this information for example to generate abstract feature languages that are needed for automatic learning.

An explicit representation of state variables, goals and actions provides knowledge about the execution state of the program. Thus when executing an action, the program can find out why this action is to be performed and use this information in choosing appropriate parameterizations.

In order to integrate learning into a programming language, we need an explicit and declarative representation of learning problems, as well as mechanisms for executing the learning steps and embedding the resulting procedure seamlessly into the code.

For the empirical evaluation of the language I develop, we have two testbeds. One is a simulated household robot that has to perform sophisticated tasks in a kitchen (figure 1). The other one is our autonomous robot soccer team, where real robots have to be controlled in highly dynamic situations.

Realization

The concepts for declaratively describing the physical world, for representing beliefs, goals and procedures, and the learning mechanisms are implemented as an extension to RPL, which is a plan language implemented as LISP macros.

Interaction with the physical world

The representation of the robot's belief is implemented by *state variables*, which don't only contain the robot's current belief about the world, but includes models about the physical meaning of each state variable. So, when specifying a state variable, we give the unit of measurement and the physical dimension of each value.

Changes in the values of physical quantities are propagated by fluents, variables that vary over time. Using fluents, the robot can wait for events and react accordingly.

For a more high-level description of the robot state we have the concept of *derived state variables*, which are a composition of other state variables. In the robot soccer environment, such a derived state variable could be the robot's current distance to the ball.

Goals, Actions and State

The robot control program uses explicit representations of the robot's state, its goals and the procedures for fulfilling the goals (figure 2).

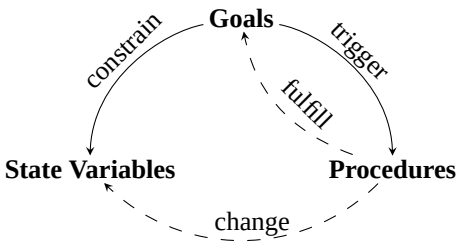

Figure 2: Interconnections between state variables, goals and procedures. The drawn through arrows denote mechanisms that are explicitly represented in the language. The dashed lines show interactions that are not mirrored in language constructs.

A goal class is defined as a restriction over a state variable. Such a state variable is called *controllable*. For a goal class the programmer must also specify a procedure to fulfill the goal. A goal can be *achieved*, where the adequate control procedure is invoked in order to fulfill some success criterion, or *maintained*, where the success criterion is constantly checked and if required, restored.

For a goal class the programmer has to state which procedure is to reach the goal. In many cases there are several possible routines with different characteristics in different situations. If we know models of these routines, we can choose the best according to the circumstances. For this we introduce the concept *control task*. The control task can choose between different *control routines*, given the current belief and models of the control routines. A control routine can be

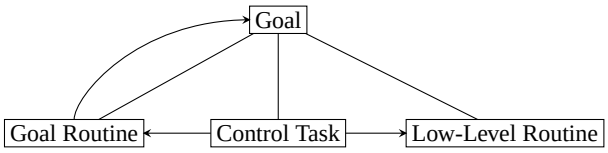

Figure 3: Calling hierarchy of goals and procedures.

either a *goal routine*, which is the only kind of procedure that can set new goals, or a *low-level-routine*, which controls the robot directly by giving commands to the robot architecture (figure 3).

Integrating Learning into Robot Control

Considering the current state-of-the-art, developing robotic agents that learn autonomously is rather an opaque art than an engineering exercise. One of the main reasons is that modern robot control languages do neither enforce nor strongly support the rigorous design of learning mechanisms. With the language ROLL (formerly called RPL_{LEARN}) (Beetz, Kirsch, & Müller 2004), we attempt to improve this situation by extending a robot control language with constructs for specifying experiences, learning problems, exploration strategies, etc. Using these constructs, learning problems can be represented explicitly and transparently and become executable.

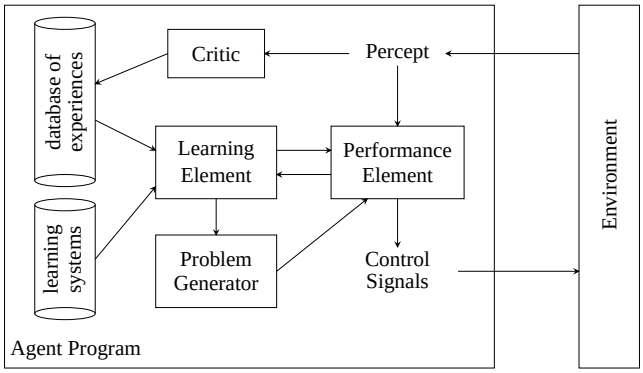

Figure 4: Learning agent after (Russell & Norvig 1995).

Figure 4 shows the parts of a learning agent. Every aspect of a learning problem is represented explicitly within ROLL.

The *performance element* realizes the mapping from percepts into the actions that should be performed next. The control procedures therein might not yet be executable or optimized. These are the procedures we want to learn.

The *critic* is best thought of as a learning task specific abstract sensor that transforms raw sensor data into information relevant for the learning element. To do so the critic monitors the collection of experiences and abstracts them into a feature representation that facilitates learning.

The *learning element* uses experiences made by the robot in order to learn the routine for the given control task.

The *problem generator* generates a control routine that is executed by the performance element in order to gather useful experiences for a given learning problem. The problems are generated according to a probability distribution as given in the learning problem specification.

is called with an experience class and returns a control routine that, when executed, will generate an experience of the respective class. The new parameterizations are generated as specified in the distribution of parameterizations of the experience class.

The language constructs for learning described here have been applied to reconstruct large parts of the control program of our soccer robots (Beetz *et al.* 2003).

Progress

In the current state, mechanisms for the representation of physical knowledge inside the state variables are implemented. Also goals, procedures and state variables are represented explicitly. The mechanisms shown in figure 2 are now reflected in programming constructs.

A first version of the language ROLL has been implemented independently of the other model-based concepts. We applied this language to several learning problems in the context of robot soccer. The constructs were also used by students in a practical course. We are integrating a revised version of the learning constructs into the model-based language context. For this purpose we store experiences in a database, so that data mining techniques for data cleaning can be applied easily to the training examples.

We used an earlier implementation (Müller, Kirsch, & Beetz 2004) of the language for implementing the control program of our soccer robots for the 2004 world championship in Lisbon. Also, learned routines for navigations tasks were included, whose performance reached the level of programmed ones (Kirsch, Schweitzer, & Beetz 2005).

A more recent version was employed for controlling the simulated kitchen robot. In this context we haven't performed any learning yet.

References

Beetz, M.; Stulp, F.; Kirsch, A.; Müller, A.; and Buck, S. 2003. Autonomous robot controllers capable of acquiring repertoires of complex skills. In *RoboCup International Symposium 2003*, Padova.

Beetz, M.; Kirsch, A.; and Müller, A. 2004. RPL-LEARN: Extending an autonomous robot control language to perform experience-based learning. In *3rd International Joint Conference on Autonomous Agents & Multi Agent Systems (AAMAS)*.

Kirsch, A.; Schweitzer, M.; and Beetz, M. 2005. Making robot learning controllable: A case study in robot navigation. submitted to Nineteenth International Joint Conference on Artificial Intelligence.

Müller, A.; Kirsch, A.; and Beetz, M. 2004. Object-oriented model-based extensions of robot control languages. In *27th German Conference on Artificial Intelligence*.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall.