Experience- and Model-based Transformational Learning of Symbolic Behavior Specifications

Michael Beetz and Thorsten Belker
University of Bonn, Dept. of Computer Science III, Roemerstr. 164,
D-53117 Bonn, Germany, beetz,belker@cs.uni-bonn.de

Abstract

This paper describes XFRMLEARN, a system that learns symbolic behavior specifications to control and improve the continuous sensor-driven navigation behavior of an autonomous mobile robot. The robot is to navigate between a set of predefined locations in an office environment and employs a navigation system consisting of a path planner and a reactive collision avoidance system. XFRMLEARN rationally reconstructs the continuous sensor-driven navigation behavior in terms of ***task hierarchies*** by identifying significant structures and commonalities in behaviors. It also constructs a statistical behavior model for typical navigation tasks. The behavior model together with a model of how the collision avoidance module should "perceive" the environment is used to detect behavior "flaws," diagnose them, and revise the plans to improve their performance. The learning method is implemented on an autonomous mobile robot.

Introduction

Many state-of-the-art navigation systems consider navigation as an instance of Markov (TBB+98) or partially observable Markov decision problems (KCK96; SK95; NPB95). They model the navigation behavior as a finite state automaton in which navigation actions cause stochastic state transitions. The robot is rewarded for reaching its destination quickly and reliably. A solution for such problems is a *policy*, a mapping from states to actions that maximizes the accummulated reward.

While (PO)MDPs aim at computing "optimal" actions they are not necessarily designed to generate the "best" behavior. This seeming contradiction has several reasons. First, the choice of the state space and transitions underlying a (PO)MDP might not be optimal. Second, solving them accurately might require too much computational resources. Third, often aspects of (PO)MDPs change over time. Fourth, their solutions suggest optimal actions only wrt. those pieces of evidence that are represented in the probabilistic state and transition models.

As autonomous robots operate in human working environments and are to solve dynamically changing sets of complex tasks, the requirements for adequate behavior, the ability to adapt quickly to changing circumstances, and competence in dealing with exceptional situations become more important. Moreover, the variety of events and entities that might have significant impact on the robot's performance yield decision problems with huge state spaces.

(PO)MDPs use a uniform mechanism for action selection and a parsimonious problem encoding comprising a state space, a probabilistic state transition model, and probabilistic perception model. In this encoding essential information for planning for *exceptional* situations and reasoning *about* situation-specific behavior is inaccessible. Therefore, the MDP formalism cannot sufficiently satisfy the requirements listed above. A more promising approach is to equip controllers with explicit models of control programs, their applicability, and the effects of running them.

But how can such explicit models be obtained? In the classical setting the answer is: they are specified by programmers. So far, this approach has not been very successful and controllers need sophisticated subsystems to bridge the gap between the assumptions of highlevel representations and the operation lowlevel control routines (BFG+97). As a result, high-level controllers often work with symbols that are not grounded and action representations that do not capture the temporal structure of the robot's behavior.

We propose to equip the robot controller with a learning component that builds up explicit models automatically. This learning component, called XFRMLEARN, constructs models of the typical behavior based on statistics collected from executing subsymbolic MDP controllers. It also reconstructs the continuous sensor-driven navigation behavior in terms of ***task hierarchies*** by identifying significant structures and commonalities in behaviors. XFRMLEARN computes symbolic behavior specifications that can approximately reproduce the behavior of the task from these task models. The symbolic behavior specifications represent the task structure and control parameters explicitly, abstractly, and modularly. This explicit representation allows symbolic inference processes to reason about behavior specifications, their applicability, and effects. They are therefore valuable resources for action planning and skill acquisition. Primitive behavior specifications are transformed into sequences of MDPs in order to compute optimal policies for achieving them.

We develop our approach to transformational learning of symbolic behavior specifications in the context of the RHINO navigation system (TBB+98). Conceptually, this navigation system works as follows. A navigation prob-

lem (a pair of locations ⟨s,d⟩ where s is the start location and d the destination) is transformed into a Markov decision problem solved by a path planner using a value iteration algorithm. The solution is a policy that maps every possible location into the optimal heading to reach the target. This policy is then given to a reactive collision avoidance module that executes the policy taking the actual sensor readings and the dynamics of the robot into account (FBT97).

RHINO's navigation behavior can be improved because the path planner solves an idealized problem that does not take the desired velocity, the dynamics of the robot, the sensor crosstalk, and the expected clutteredness fully into account. The reactive collision avoidance component takes these aspects into account but makes only local decisions. Our aim is to improve the behavior of the original navigation system significantly but not to make it optimal by fine-tuning all possible parameters.

Given an environment model and control over an autonomous robot equipped with a subsymbolic navigation system XFRMLEARN (1) computes a descriptive model of the robot's navigation behavior including a symbolic task model that represents the structure of navigation behavior for typical navigation tasks; (2) transforms the tasks in the task model into symbolic structured behavior specifications that can reproduce the behavior of the task; and (3) empirically tests the symbolic behavior specifications and revises them in order to improve the robot's navigation behavior. The next three sections will describe these three learning tasks in more detail. We conclude with a discussion of related work.

Model Construction

This section describes MC, XFRMLEARN's subsystem for constructing descriptive models of the robot's behavior. Fig. 1 pictures the input, output, and computational structure of MC. The input given to MC is an environment model and a set of predefined destinations (see Fig. 3(left)). The environment model is a 3D grid-based map that represents the structure of the environment (rooms, doorways, walls, ...) and the static task-relevant pieces of furniture (desks, shelves, ...). The second input is a set of destinations specified through their coordinates which define the set of standard navigation tasks. In addition, MC uses the RHINO navigation system as a resource to acquire data about the navigation behavior to be modelled.

MC constructs three different models: (1) a ***local clutteredness map*** that measures the clutteredness of the surroundings of a location; (2) a ***behavior map*** which is a very simple statistical model of the behavior generated by the subsymbolic navigation system; and (3) a ***task model*** that describes the decomposition of navigation tasks into subtasks. These models allow XFRMLEARN to reason about and revise the behavior specifications to improve the robot's navigation behavior.

MC performs two steps: the ***exploration step*** acquires behavior traces for learning behavior and task models and the ***model construction step*** generates descriptive models for the behavior traces. To collect data for model construction MC performs each predefined navigation task *n* times. It performs the navigation tasks by passing them to the RHINO navigation system, steadily records the robot's state (comprising its position, translational and rotational velocity) while executing the tasks, and stores the state trajectories as behavior traces.

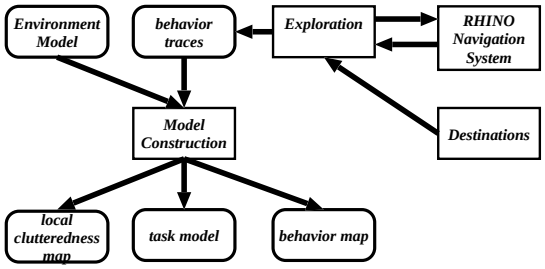

Figure 1: Computational structure of MC.

Local Clutteredness Maps. The performance of RHINO's navigation behavior is partly determined by the collision avoidance module resetting the robot's heading and velocity based on the immediate perceived obstacles, the robot's dynamical state, and its next destination. In a nutshell the expected interference increases with the clutteredness, the translational velocity, and the rotational velocity where the translational and rotational velocity can be partly controlled. The translational velocity can be changed by setting the robot's travel mode (fast travel, standard, and doorway mode) the average amount of rotation can be reduced by guiding the robot on paths that have maximal clearance.

MC constructs a ***local clutteredness map*** as an environment representation that provides for every location a coarse-grained measure of how an ideal sensor model of the range sensor used for collision avoidance would perceive the clutteredness around the location. The local clutteredness maps tesselate the environment into grid cells (in our case 75×75cm) and stores for each cell a rough qualitative measure of clutteredness (high, medium, and low).

The local clutteredness of a grid cell is computed in a three-step process. In the first step, we use the models of the sensors used by the collision avoidance and the static 3D model of the environment to compute the expected readings generated by the ideal sensor model (see (BFH97)). Fig. 2(left) shows the result of this step for two locations and a sonar sensor at table height. Of course, the sensor data are too finegrained to take them as a basis for characterizing the situation for the purpose of collision avoidance behavior. These readings are then condensed into a compact signature consisting of the minimal distance measured north, west, south, and east of the robot (Fig. 2(middle)). Finally, we classify the signatures into three predefined classes. The first class contains the locations where the sum of the opposite readings are smaller than two meters. The second class contains those for which one pair of opposite readings is smaller and one pair larger than two meters, and the last class contains those in which both pairs are larger than two meters. Thus the first class are those locations the robot perceives as highly cluttered, the second as medium, and the last one as lowly cluttered. Fig. 2(right) shows the local clutteredness map that is computed this way.

The local clutteredness map serves as a resource for ex-

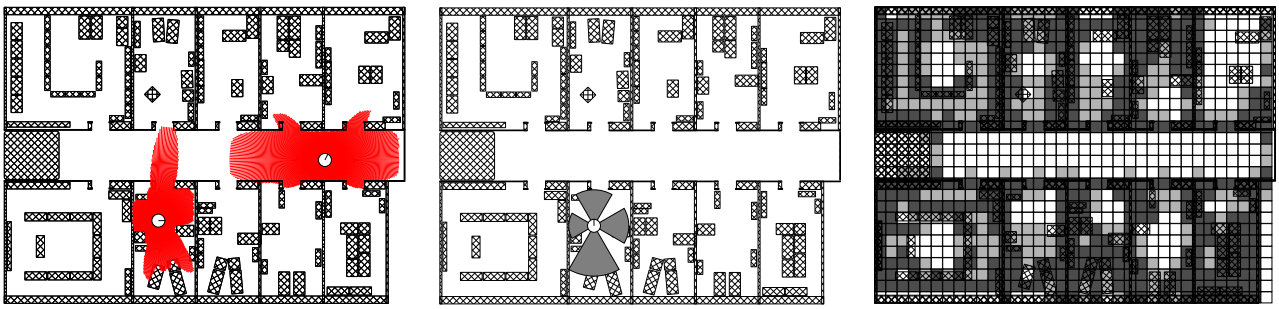

Figure 2: Idealized sonar sensor reading (left). Signature of a sonar reading (middle). Local clutteredness map (right).

plaining strong deviations from expected navigation behavior. It is also the basis for computing the ***travel mode area map***, a mapping from grid cells to travel mode areas, which are regions that the robot should be able to traverse well without changing its travel mode. A region coloring algorithm computes the travel mode area map from the local clutteredness map by naming the connected grid cells with the same local clutteredness in the same room with a name. The highlevel navigation system provides a global register CURR-BEH-MODE-AREA that contains the name of the travel mode area that the robot currently traverses and that is updated automatically. Using the register the robot can run control routines that adapt the travel mode of the robot when entering particular travel mode areas.

Behavior Maps. The second model constructed by XFRM-LEARN is the ***behavior map*** that represents the typical navigation behavior as a function of the robot's location. The behavior map also tesselates the environment into 75×75cm grid cells where each grid cell stores the average translational and rotational speed that the robot has when it traverses the grid cell. These values constitute the running average over previous navigation tasks weighted by their recency.

Fig. 4 shows a behavior map for a set of navigation tasks. The size of the circles represent the translational and their brightness the rotational speed.

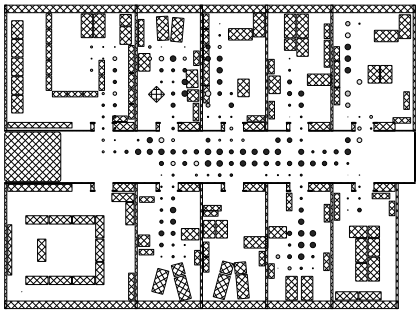

Figure 4: A behavior map for a set of tasks.

The **Task Model** describes the decomposition of the (predefined) navigation tasks into subtasks and is used for generating symbolic behavior specifications (see (Sus77)). The criteria that govern the decomposition of navigation tasks into subtasks are analogous to those that are applied in structured programming. Problems that cannot be sufficiently optimized using a single objective function are decomposed until they can. Problems that occur in multiple tasks are turned into subtasks so that their solutions can be used as subroutines in different plans.

A hierarchical task model that is reconstructed from behavior traces after these criteria can be used to construct plan libraries that avoid redundancies and aim at minimizing the number of parameters for controlling the behavior, reducing the complexity of learning as well as planning.

The predefined navigation tasks that ask the robot to get from one place to another one are shown in Fig. 3(left). This figure also shows a trajectory in the state variables X and Y that store the robot's coordinates within the environment. The trajectory can be explained in terms of getting to the destination quickly, avoiding perceived obstacles, and passing narrow passages.

The construction of a task model consists of two main steps: (1) the approximation of trajectories through sequences of line segments and (2) the identification of common structures in sets of related navigation trajectories.

Piecewise linear trajectory approximations. Office environments are designed with navigation being one of their main functions. Office environments have hallways for fast navigation, furniture pieces are often arranged that there are pathways to all desks in the offices, etc. If there were no obstacles, trajectories for navigation tasks would be almost straight lines. In office environments with typical furnishing trajectories can be adequately approximated through piecewise linear trajectories. The line segments and their connections prove themselves as good hints where we can look for qualitatively different behaviors.

The input of the approximation step is a recorded trajectory produced by RHINO's subsymbolic navigation system (Fig. 3(left)). The approximation is determined by a simple recursive algorithm that computes a piecewise linear function such that no point on the trajectory has a distance to the line greater than a given threshold θ. For the experiments in this paper we have set θ to 20cm. A trajectory and its approximation is shown in Fig. 3(middle).

Finding Common Subtasks. To find commonalities in navigation behavior, MC retrieves sets of *related* navigation tasks, that is tasks that share a common destination or starting point, from the recorded behavior traces.

To compute the subtasks of $\langle s, d \rangle$ where s and d are predefined destination points we first compute the common sub-

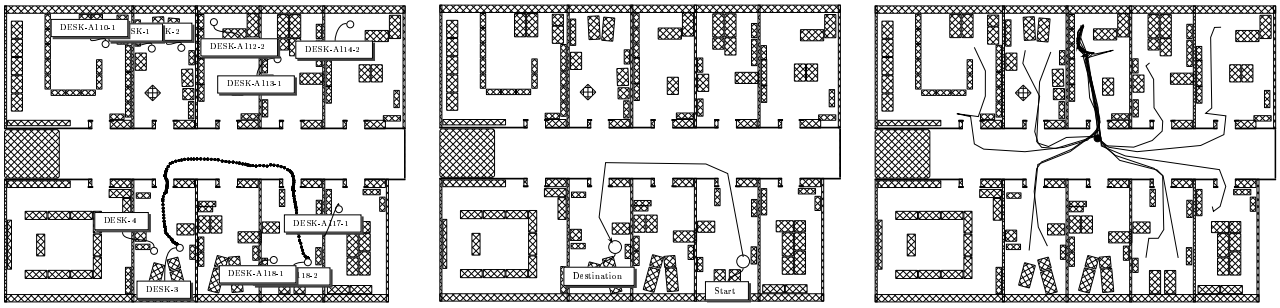

Figure 3: Navigation tasks and trajectories (left). Trajectory approximation (middle). Common subtask (right).

task $\langle s, \gamma_s \rangle$ of all tasks $\langle s, d_i \rangle$ and then the common subtask $\langle \gamma_d, d \rangle$ of all tasks $\langle s_i, d \rangle$. We then add the decomposition seq($\langle s, \gamma_s \rangle$,$\langle \gamma_s, \gamma_d \rangle$,$\langle \gamma_d, d \rangle$) of $\langle s, d \rangle$ to the task model. The γ_ss and γ_ds are called ***junction points***.

In our application, the task hierarchy has depth one. In general, deeper task hierarchies are generated by applying the algorithm above to the set of all navigation tasks defined by the cross product of the junction points. One can think of the task model as a DAG where the edges represent the subtask relationship.

To describe the computation of junction points we have to introduce the notion of *distinctive points*. Given approximations of k related trajectories (Fig. 3) distinctive points are points that (1) connect two subsequent line segments on an approximation and (2) at least $d \times k$ approximated trajectories intersect its r neighborhood. XFRMLEARN uses 0.6 for d and 50cm for r, which means for a given distinctive point p 60% of the k trajectories have a point that is at most 50cm away from p.

Given the distinctive points, *junction points* are computed by the following algorithm: (1) sort the distinctive points according to their distance to the common start respectively destination point; (2) choose the point p with maximal distance; (3) compute the center of mass of the distinctive points in the r neighborhood of p and return it as the junction point γ of the related tasks.

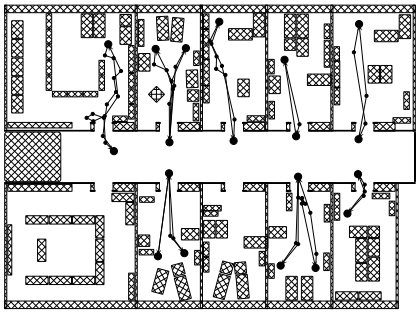

Figure 5: Task model for the navigation tasks.

A primitive subtask is a task without further decomposition. With each primitive task $\langle s_{sub}, d_{sub} \rangle$ is associated a concise trajectory description, a connected sequence of line segments from s_{sub} to d_{sub} defined by the start point, a sequence of *intermediate points* and the destination point. It is computed as the piecewise linear approximation of a trajectory generated by executing the task $\langle s_{sub}, d_{sub} \rangle$ with the low-level navigation system.

Fig. 5 pictures the task model generated by XFRMLEARN for the given set of navigation tasks where the tasks for going from one junction point to another one are not displayed. For the 110 predefined tasks XFRMLEARN has identified 78 actions. 22 of them are non-trivial (the ones displayed in Fig. 5) meaning that they contain intermediate points that specify the paths to be taken. Each of the depicted subtasks, which are also the ones that can be expected to require the most adaptations, are shared by ten tasks.

The task model has several flaws. The first one is that it overgeneralizes. For example, the tasks for going from a desk to another desk in the same room goes via the common junction point. Second, the task for leaving the upper left room contains a loop. Rather than fine tuning the algorithms for constructing the task model we believe that these problems should be eliminated in the transformational learning step because any algorithm that needs fine tuning for the construction of the task model will fail when applied to other problem instances.

Generating Symbolic Behavior Specifications

Our ultimate goal is improving the robot's navigation behavior by revising the behavior specifications producing the behavior. We have already argued in the Introduction that the representations that MDPs are stated with are too fine grained for fast transformational learning. XFRMLEARN makes up more coarse grained symbolic behavior specifications that can compactly represent various aspects of behavior and be used to explain behavior. In this Section we will describe the representation and the construction of these symbolic behavior specifications. We will do so by answering two questions: (1) how can the generalizability of transformational learning steps be achieved? and (2) what are good behavior specification to start transformational learning with?

The behavior specification generation step supports generalizability primarily by modularizing behavior specifications. By modularity we mean that the structure of the (prescriptive) behavior specifications reflects the structure of the (descriptive) task model. Modular behavior specifications are constructed by generating individual behavior specifications for the primitive subtasks and combining them as described by the subtask structure of the model. Thus the revision of a behavior specification associated with a par-

ticular task *t* is inherited by all behavior specifications that accomplish *t* as a subtask.

Behavior specifications for primitive tasks $\langle s_{sub}, d_{sub} \rangle$ are called ***actions*** and represented as

action PRIM-TASK-I($\langle s_{sub}, d_{sub} \rangle$)
1 **PATH-SPECIFICATION** $\langle p_1 = s_{sub}, p_2, ..., p_n = d_{sub} \rangle$
2 **TRAVEL-MODE-ADAPTATIONS** $\langle a_1 : m_1, ..., a_k : m_k \rangle$

where $\langle p_1 = s_{sub}, p_2, ..., p_n = d_{sub} \rangle$ are points that constrain the path to be taken and $\langle a_1 : m_1, ..., a_k : m_k \rangle$ specify the set of travel mode adaptations $a_j : m_j$ which means that the robot is to adapt its travel mode to the mode m_j when entering the behavior mode area a_j.

We initialize actions by setting $p_2, ..., p_{n-1}$ to the intermediate points computed for the primitive subtask and the adaptations list as being empty. An empty adaptation list means that the robot navigates through the environment using the default travel mode only. The rationale behind these choices are: (1) The approximations of the trajectories generated by the RHINO navigation system (in particular, the number of intermediate points) give us a good guess how complex a path-specification for an action should be. XFRMLEARN tries to keep the number of path constraints small to reduce the number of possible transformations in a single transformation step. (2) The RHINO navigation system already works reliably and well in office environments. We expect performance gains mainly by tailoring the navigation behavior for particular environment and sets of tasks.

Before behavior specifications are executed they are transformed into RPL plans. RPL (Reactive Plan Language) (McD91) provides conditionals, loops, program variables, processes, and subroutines. RPL also places high-level constructs (interrupts, monitors) to synchronize parallel physical actions and make plans reactive and robust by incorporating sensing and monitoring actions, and reactions triggered by observed events at the programmer's disposal (see (McD91) for details). The symbolic behavior specification specifies a sequence of Markov decision problems and a parameterization of the collision avoidance module that are solved and executed by the RHINO navigation system. The pseudo RPL code describes an RPL plan that is constructed from a symbolic behavior specification.

```
symbolic-behavior-specification TASK-25()
 1  with policy whenever    CHANGED?
 2                                (curr-beh-mode-area)   do
 3              sequentially do
 4                if curr-beh-mode-area = 51
 5                  then SET-TRAVEL-MODE(office)
 6                if curr-beh-mode-area = 63
 7                  then SET-TRAVEL-MODE(office)
 8                if curr-beh-mode-area = 59
 9                  then SET-TRAVEL-MODE(doorway)
10    do sequentially do
11         START-NAV-PLAN
12              (   ⟨⟨1100.0, 1390.0⟩, ⟨975.0, 1675.0⟩⟩   )
13         wait for (navigation-completed?)
```

The RPL behavior specification consists of a body that sends a path to be followed to the RHINO navigation system and starts the system. After the activation of the navigation process the body waits for the completion signal. The execution of the body is subject to a policy, a constraint on its execution. This policy monitors the global register ***curr-beh-mode-area*** for changes. If the register changes to ***51*** or ***63*** the robot's travel mode is set to *office*, if it changes to ***59*** then it is set to doorway.

Transformational Learning

After the initial plan library has been build up transformational learning uses the local clutteredness map and the behavior map as resources and proceeds in a loop with four steps: ***explore***, ***analyze***, ***revise***, and ***test***. In the ***explore*** step XFRMLEARN performs a set of experiments in order to detect unexpected patterns of behavior. These unexpected patterns that might hint behavior flaws are then detected and diagnosed in the second, the ***analyze*** step. The ***revise*** step applies plan revision rules to the action that produced the unexpected behavior pattern that modify the behavior specification of this action.

The learning knowledge is encoded using predefined predicates: AVG-TRANS-SPEED(*p,v*) represents that the average translational speed at location *p* is *v*. Distinctive points *p* of an action *a* are represented by the predicate DISTINCTIVE-POINT(*a,p*) and the local clutteredness by LOCAL-CLUTTEREDNESS(*p,c*). These predicates are implemented using the behavior map, plan library, and clutteredness map.

The Transformation Step

XFRMLEARN will be equipped with a set of transformational learning rules that are applied to possible behavior flaws. Up to now we have experimented only with a few individual rules in isolation. One of these rules is shown below. The rule determines an alternative pair (distance from the obstacle, speed) for going around a set of obstacles. It can do this because going around is associated with a distinctive point.

IF DISTINCTIVE-POINT(*p*)
∧ ¬ LOCAL-CLUTTEREDNESS(*p,low*)
∧ ¬ AVG-TRANS-SPEED(*p,high*)
∧ CLOSEST-OBSTACLE(*?p,?obst*)
∧ POINT-WITH-MORE-CLEARANCE(*?p',?p,?obst*)
∧ BEHAVIOR-MODE-CLUSTER(*?p,?c-name*)
THEN **MOVE** *?p* **TO** *?p'*
ADD ?c-name : *medium* **TO TRAVEL-MODE-LIST**

This learning rule is applicable if the candidate opportunity is a distinctive point *?p* in a cluttered area where the robot drove slowly. The condition also determines the behavior mode area *?c-name* of *?p* and a point *?p'* with more clearance than *p*. The first three condition test whether a distinctive point might be an opportunity for improving the robot's behavior; the remaining three check whether a move of the distinctive point might be promising. The rule performs two transformations: one transforms the behavior specification of the corresponding action the other one revises the coordinates of a symbolic distinctive point. The modification of the behavior specification is the addition of

a new navigation mode adaptation: if the robot enters the behavior mode area *?c-name* it increases the desired speed.

A second rule specifies: if a waypoint of the toplogical navigation plan lies in a narrow passage and in the passage the average rotational speed is high then delete that way point and substitute it by two other ones: one before and one after the passage and so that the robot has maximum clearage in the passage.

Another category of revision rules are triggered by the task model and the initial behavor specification being faulty. For example, using the initial plan library derived from the task model depicted in Fig. 5 the robot goes from *DESK-1* in *A-111* to *DESK-2* in the same room via the junction point near the door of *A-111*. This happens because XFRMLEARN overgeneralizes the plans for getting to *DESK-2*. Such flaws are eliminated by rules like this one: if a navigation plan results in a behavior that is twice as bad as the plan without intermediate points then conditionalize the stored plan so that it does not apply to the critical situations.

XFRMLEARN chooses an applicable rule, applies it, and compares the performance of the revised plan with the original one. If the revised plan is better than the original plan, then the revision is incorporated into the plan library.

First preliminary tests with a variation of the first rule have shown that placing intermediate points such that the robot passes obstacles with more clearance can reduce the average time the robot is stuck and has to move away from the obstacle without making progress towards its destination by a factor two. We do not want to make claims about performance gains based on such isolated rule applications. In our view, revisions performed by XFRMLEARN cannot be evaluated by comparing the performance of the behavior produced by the specification for a given task before and after a single modification and in isolation. We believe that the overall performance gain has to be tested more globally, for example by performing a t-test on the behavior map before and after a series of transformations. At the current stage of implementation we do not have a library of transformational learning rules that has enough coverage to perform such tests.

Related Work

XFRMLEARN can be viewed as an extension of the transformational planning system XFRM (McD92). Using a library of modular default plans, XFRM computes a plan for a given compound task and improves the plan during its execution based on projections of its execution in different situations. XFRMLEARN is a first step towards automatically learning such default plan libraries. XFRMLEARN differs from many approaches that build up plan libraries by plan compilation/explanation-based learning (LRN86; Mit90) in that it does not require the existence of symbolic action representations. In contrast, XFRMLEARN makes up symbolic behavior specifications by analyzing the behaviors generated from MDP policies and abstracting them. There is an enormous amount of work on computing optimal decisions for Markov decision problems (KCK96; KLC95). Recently, Sutton *et al.* (SPS98) have proposed an approach for learning macro operators in the decision-theoretic framework using reinforcement learning techniques. However, as we have already stated we believe that the approach based on solving MDPs is more tailored for selecting optimal decisions in standard situations than for making control decisions based on projecting courses of action in specific situations.

The IMPROVE algorithm (LMA98) runs data mining techniques on simulations of large, probabilistic plans in order to detect defects of the plan and applies plan adaptations to avoid these effects. Dependency interpretation (HC92) uses statistical dependency detection to identify patterns of behavior in execution traces in order to detect possible causes of plan failures. The main differences to our approach include that our approach also learns the symbolic actions and adapts and improves continuous control processes. Haigh and Veloso propose a method for learning the situation-specific costs of given actions (HV98) whereas our approach aims at learning and improving the actions themselves.

Our ultimate goal is to develop a system for skill acquisition on autonomous mobile robots inspired by HACKER (Sus77). As Sussman believes that skill acquisition is a process of compiling abstract declarative knowledge into executable procedures. The last twenty years have shown that "executable" should mean "executable on physical systems." Therefore, we investigate the problem of skill acquisition in the context of controlling an autonomous mobile robot.

Conclusions

This paper describes XFRMLEARN, a learning component of an autonomous mobile robot that employs a navigation system consisting of a path planner and a reactive collision avoidance system. XFRMLEARN learns symbolic behavior specifications to control and improve the continuous sensor-driven navigation behavior. XFRMLEARN detects behavior "flaws," diagnoses them, and revises the behavior specifications to improve their performance. Our next steps in the further development of XFRMLEARN include the specification of a library of transformation rules.

The learning method builds a synthesis among various subfields of AI: computing optimal actions in stochastic domains, symbolic action planning, learning and skill acquisition, and the integration of symbolic and subsymbolic approaches to autonomous robot control.

An important issue that still needs to be discussed is that of generality. In this paper we have only addressed the navigation problem in indoor environments, first because it builds the basis for most other tasks for mobile robots and second because the problem is well understood and solved in a variety of ways. We believe that our approach carries over to other tasks in which Markov decision problems are solved and executed by a reactive execution component. MDPs will try to compute policies that generate the fast trajectories through the state space to the goal state. Where the real trajectory significantly deviates from the optimal ones there are often reasons that force the reactive execution component to correct the precomputed trajectory and which have been idealized away in the formulation of the Markov deci-

sion problem. To apply our approach to other problem areas is on our agenda for future research.

Our approach also takes a particular view on the integration of symbolic and subsymbolic reasoning processes, in particular MDPs. In our view symbolic representations are resources that allow for more economical reasoning. The representational power of symbolic approaches can enable robot controllers to better deal with complex and changing environmants and achieve changing sets of interacting jobs. This is achieved by making more information explicit and representing behavior specifi cations symbolically, transparently, and modularly. In our approach, (PO)MDPs are viewed as a way to ground symbolic representations.

References

P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(1), 1997.

Wolfram Burgard, Dieter Fox, and Daniel Hennig. Fast grid-based position tracking for mobile robots. In *Proceedings of the 21th German Conference on Artificial Intelligence (KI 97), Freiburg, Germany*. Springer Verlag, 1997.

D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 1997.

A. Howe and P. Cohen. Isolating dependencies on failure by analyzing execution traces. In J. Hendler, editor, *AIPS-92: Proc. of the First International Conference on Artificial Intelligence Planning Systems*, pages 277–278. Kaufmann, San Mateo, CA, 1992.

K. Haigh and M. Veloso. Learning situation-dependent costs: Improving planning from probabilistic robot execution. In *Second Int. Conf. on Autonomous Agents 98*, 1998.

Leslie Pack Kaelbling, Anthony R. Cassandra, and James A. Kurien. Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. Technical report, Brown University, 1995.

N. Leash, N. Martin, and J. Allen. Improving big plans. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

J. Laird, P. Rosenbloom, and A. Newell. Chunking in soar: the anatomy of a general learning mechanism. *Machine Learning*, 1:11–46, 1986.

D. McDermott. A reactive plan language. Research Report YALEU/DCS/RR-864, Yale University, 1991.

D. McDermott. Transformational planning of reactive behavior. Research Report YALEU/DCS/RR-941, Yale University, 1992.

T. Mitchell. Becoming increasingly reactive. In *Proceedings of the Eight National Conference on Artificial Intelligence*, pages 1051–1059. AAAI Press/MIT Press, 1990.

Illa Nourbakhsh, Rob Powers, and Stan Birchfield. DERVISH an office-navigating robot. *AI Magazine*, 16(2):53–60, Summer 1995.

Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *Proc. International Joint Conference on Artificial Intelligence*, 1995.

R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: Learning, planning, and representing knowledge at multiple temporal scales. *Journal of AI Research*, 1998.

G. Sussman. *A Computer Model of Skill Acquisition*, volume 1 of *Aritficial Intelligence Series*. American Elsevier, New York, NY, 1977.

S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schimdt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R.P. Bonasso, and R. Murphy, editors, *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press, Cambridge, MA, 1998.

Remark. *Due to the fact that our hardware platform has been broken for several weeks the experiments have been carried out on our robot simulator. If the paper gets accepted they will be replaced by runs on the real robot.*