Learning Structured Reactive Navigation Plans from Executing MDP Navigation Policies

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. Autonomous robots, such as robot office couriers, need navigation routines that support flexible task execution and effective action planning. This paper describes XFRMLEARN, a system that learns structured symbolic navigation plans. Given a navigation task, XFRMLEARN learns to structure continuous navigation behavior and represents the learned structure as compact and transparent plans. The structured plans are obtained by starting with monolithic default plans that are optimized for average performance and adding subplans to improve the navigation performance for the given task. Compactness is achieved by incorporating only subplans that achieve significant performance gains. The resulting plans support action planning and opportunistic task execution. XFRMLEARN is implemented and extensively evaluated on an autonomous mobile robot.

1 Introduction

Robots operating in human working environments and solving dynamically changing sets of complex tasks are challenging testbeds for autonomous robot control. The dynamic nature of the environments and the nondeterministic effects of actions requires robots to exhibit concurrent, percept-driven behavior to reliably cope with unforeseen events. Moreover, acting competently often requires foresight and weighing alternative courses of action under uncertainty.

Different approaches have been proposed to specify the navigation behavior of such service robots. A number of researchers consider navigation as an instance of Markov decision problems (MDPs) [TBB+98, KCK96]. 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 discretized robot poses into fine-grained navigation actions.

MDPs form an attractive framework for navigation because they use a uniform mechanism for action selection and a parsimonious problem encoding. The navigation policies computed by MDPs aim at robustness and optimizing the average performance. One of the main problems in the application of MDP planning techniques is to keep the state space small enough so that the MDPs are still solvable. This limits the number of contingent states that can be considered.

Another approach is the specification of environment- and task-specific navigation plans, such as ***structured reactive navigation plans (SRNPs)*** [Bee99]. SRNPs specify a default navigation behavior and employ additional concurrent, percept-driven subplans that overwrite the default behavior while they are active. The default navigation behavior can be generated by an MDP navigation system. The activation and deactivation conditions of the subplans structure the continuous navigation behavior in a task-specific way.

SRNPs are valuable resources for opportunistic task execution and effective action planning because they provide high-level controllers with subplans such as traverse a particular narrow passage or an open area. More specifically, SRNPs (1) can generate qualitative events from continuous behavior, such as entering a narrow passage [BBFC98]; (2) support online adaptation of the navigation behavior (drive more carefully while traversing a particular narrow passage) [Bee99], and (3) allow for compact and realistic symbolic predictions of continuous, sensor-driven behavior [BG00]. The specification of good task and environment-specific SRNPs, however, requires tailoring their structure and parameterizations to the specifics of the environmental structures and empirically testing them on the robot.

We propose to bridge the gap between both approaches by learning SRNPs from executing MDP navigation policies. Our thesis is that a robot can autonomously learn compact and well-structured SRNPs by using MDP navigation policies as default plans and repeatedly inserting subplans into the SRNPs that significantly improve the navigation performance. This idea works because the policies computed by the MDP path planner are already fairly general and optimized for average performance. If the behavior produced by the default plans were uniformly good, making navigation plans more sophisticated would be of no use. The rationale behind requiring subplans to achieve significant improvements is to keep the structure of the plan simple.

We have implemented XFRMLEARN a realization of this learning model and applied it to learning SRNPs for an autonomous mobile robot that is to perform office courier service. XFRMLEARN has learned compact and transparent SRNPs, which improved the navigation behavior substantially. In our view, the performance gains over MDP navigation policies indicate that the syntactic structure of the plans represents an advantageous discretization of the continuous navigation behavior.

In the remainder of the paper we proceed as follows. The next section gives a glimpse of the XFRMLEARN learning system. After this section SRNPs are described and discussed. The subsequent section details the operation of XFRMLEARN. We continue with our experimental results, a discussion of our approach, a summary of related work, and our conclusions.

2 An Overview on XFRMLEARN

XFRMLEARN is embedded into a high-level robot control system called ***structured reactive controllers*** (SRCs) [Bee99]. SRCs are controllers that can revise their intended course of action based on foresight and planning at execution time. SRCs employ and reason about

plans that specify and synchronize ***concurrent percept-driven*** behavior. Concurrent plans are represented in a transparent and modular form so that automatic planning techniques can make inferences about them and revise them.

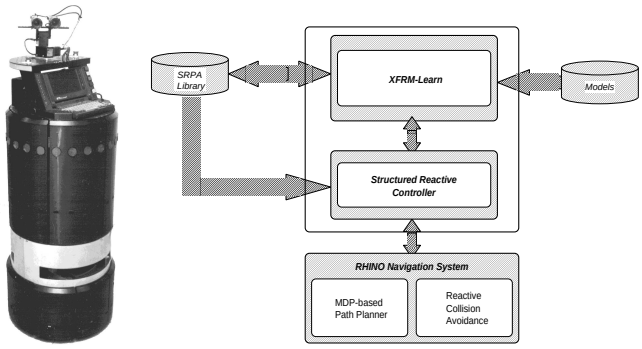

Figure 1. The autonomous mobile robot RHINO (left). The software architecture that employs XFRMLEARN.

XFRMLEARN is applied to the RHINO navigation system [TBB+98], which has shown impressive results in several longterm experiments [TBB+99]. Conceptually, this robot navigation system works as follows. A navigation problem is transformed into a Markov decision problem to be 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].

The RHINO navigation system can be parameterized in different ways. The parameter PATH is a sequence of intermediate points which are to be visited in the specified order. COLLI-MODE determines how cautiously the robot should drive and how abruptly it is allowed to change direction. The parameter SONAR specifies whether the sonar sensors are to be used in addition to laser sensors for obstacle detection.

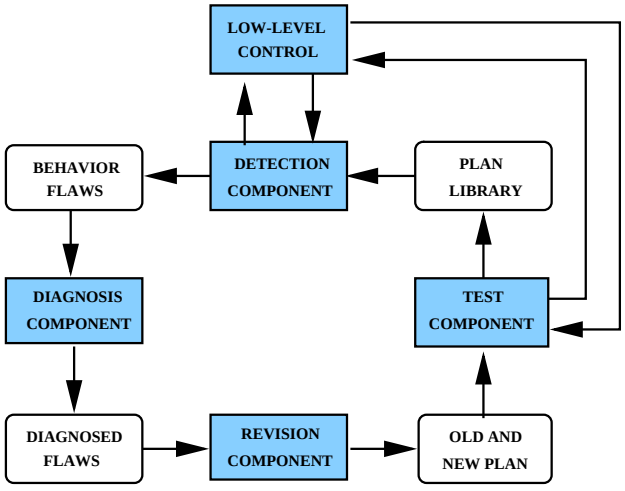

Figure 2. XFRMLEARN's "analyze, revise, and test" cycle.

RHINO's navigation behavior can be improved because RHINO's 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.

We propose an "analyze, revise, and test" cycle as a computational model for learning SRNPs (see Figure 2. XFRMLEARN starts with a default plan that transforms a navigation problem into an MDP problem and passes the MDP problem to RHINO's navigation system. After RHINO's path planner has determined the navigation policy the navigation system activates the collision avoidance module for the execution of the resulting policy. XFRMLEARN records the resulting navigation behavior and looks for stretches of behavior that could be possibly improved. XFRMLEARN then tries to explain the improvable behavior stretches using causal knowledge and its knowledge about the environment. These explanations are then used to index promising plan revision methods that introduce and modify subplans. The revisions are then tested in a series of experiments to decide whether they are likely to improve the navigation behavior. Successful subplans are incorporated into the symbolic plan.

3 Structured Reactive Navigation Plans

We will now take a more detailed look at SRNPs. SRNPs have the following syntax.

navigation plan *(start,dest)*
  with subplans *subplan**
  DEFAULT-GO-TO (*dest*).

with *subplan*s of the form

SUBPLAN-CALL(*args*)
  parameterizations $\{ p \leftarrow v \}^*$
  path constraints $\{ \langle x,y \rangle \}^*$
  justification *just*

where $p \leftarrow v$ specifies a parameterization of the subsymbolic navigation system. In this expression p is a parameter of the subsymbolic navigation system and v is the value p is to be set to. The parameterization is set when the robot starts executing the subplan and reset after the subplan's completion. The path constraints are sequences of points that specify constraints on the path the robot should take when carrying out the subplan. The subplan call together with its arguments specifies when the subplan starts and when it terminates (see below). The last component of the subplan is its justification, a description of the flaw the subplan is intended to eliminate. To be more concrete consider the following SRNP (See Figure 9c).

navigation plan *(desk-1,desk-2)*
  with subplans
    TRAVERSE-NARROW-PASSAGE($\langle 635,1274 \rangle$,$\langle 635,1076 \rangle$)
      parameterizations *sonar* $\leftarrow$ *off*
      colli-mode $\leftarrow$ *slow*
      path constraints $\langle 635,1274 \rangle$,$\langle 635,1076 \rangle$
      justification *narrow-passage-bug-3*
    TRAVERSE-NARROW-PASSAGE(...)
    TRAVERSE-FREE-SPACE(...)
  DEFAULT-GO-TO (*desk-2*)

The SRNP above contains three subplans: one for leaving the left office, one for entering the right one, and one for speeding up the

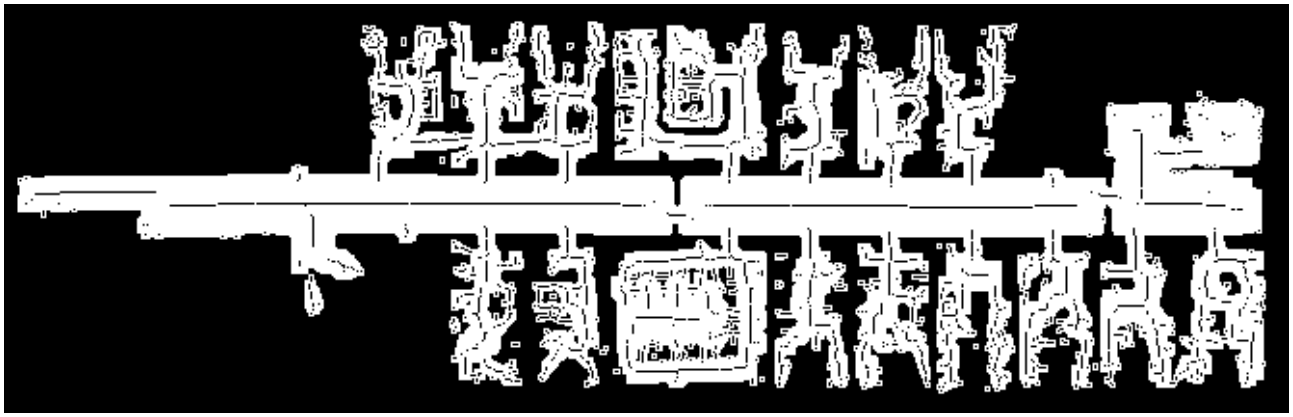

Figure 3. A learned environment map with the main axes Voronoi skeleton.

traversal of the hallway. The subplan for leaving the left office is shown in more detail. The path constraints are added to the plan for causing the robot to traverse the narrow passage orthogonally with maximal clearance. The parameterizations of the navigation system specify that the robot is asked to drive slowly in the narrow passage and to only use laser sensors for obstacle avoidance to avoid the hallucination of obstacles due to sonar crosstalk.[1]

SRNPs are called ***structured*** because the subplans explicitly represent task-relevant structure in continuous navigation behavior. They are called ***reactive*** because "perceived" qualitative events, such as entering or leaving a narrow passage, trigger the activation and termination of subplans.

The plan interpreter expands the symbolic navigation plan macros into procedural reactive control routines. Roughly, the plan macro expansion does the following. First, it collects all path constraints from the subplans and passes them as intermediate goal points for the MDP navigation problem. Second, the macro expander constructs for the subplan call a monitoring process that signals the activation and termination of the subplan. Third, the following concurrent process is added to the navigation routine: wait for the activation of the subplan, set the parameterization as specified, wait for the termination of the subplan, and reset the parameterization.

4 XFRMLEARN in Detail

In this Section we will describe the three learning steps, ***analyze***, ***revise***, and ***test*** in more detail.

4.1 The "Analyze" Step

A key problem in learning structured navigation plans is to structure the navigation behavior well. Because the robot must start the subplans and synchronize them, it must be capable of "perceiving" the situations in which subplans are to be activated and deactivated. Besides being perceiveable, the situations should be relevant for adapting navigation behavior. We use the concept of ***narrowness*** for detecting the situations in which the navigation behavior is to be adapted. Based on the concept of narrowness the robot can differentiate situations such as free space, traversing narrow passages, entering narrow passages, and leaving narrow passages.

[1] The programmers can specify in which areas it is safe to ignore sonar readings for obstacle avoidance.

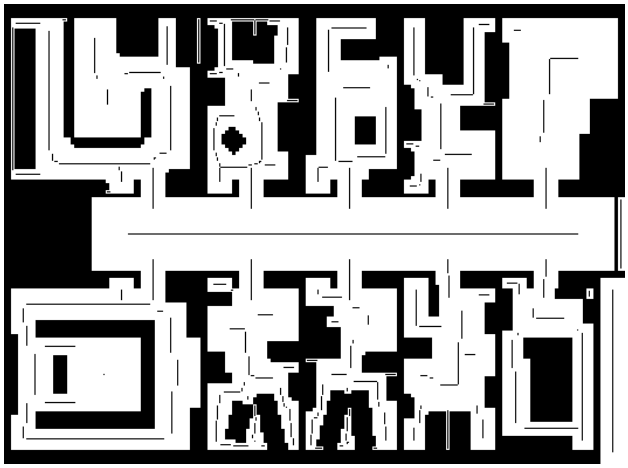

Figure 4. A CAD map with the main axes Voronoi skeleton.

We compute the perceived *narrowness* as a function of the robot's position. To do so, we use a *main axes Voronoi skeleton*, a particular environment representation that can be computed from autonomously learned environment maps. Figure 3 shows such a learned map of the robot's operating environment and Figure 4 shows a CAD map of the same environment. Both maps contain the main axes Voronoi diagram. Given a map, we can compute its Voronoi diagram, a collection of regions that divide up the map. Each region corresponds to one obstacle, and all the points in one region are closer to the corresponding obstacle than to any other obstacle. The main axes Voronoi skeleton are the set of borderlines between the Voronoi regions that are parrallel to the environment's main axes (Figure 4). Main axes Voronoi skeletons are more robust against inaccuracies of maps that are caused by sensor noise and inaccuracies in the robot's position estimation than their ordinary counterparts.

Based on the Voronoi skeleton we define perceived narrowness as follows. Every point on the Voronoi skeleton has a clearance, its distance to the next obstacle. The narrowness of points is the clearance of the closest point on a Voronoi skeleton. Here, we define that a location is to be perceived as narrow if the distance to the next obstacle is smaller than one robot diameter; but, of course, this value should be learned from experience. Based on these definitions the robot has

the following perceptions of narrowness on the path depicted in Figure 6(right).

Diagnosis of Conspicous Subtraces. If robots had perfect sensing capabilities and effector control and there were no inertia in the robot's dynamics, then the robot would perform exactly as specified in its navigation plans. In practice, however, robots rarely satisfy these criteria and therefore the behavior they exhibit often deviates substantially from what is specified in the control programs. Typically, the effects of the robot's deficiencies depends on the surroundings of the robot. As a rule of thumb: the more challenging the surroundings the larger are the effects of the robot's deficiencies. For example, in wide open areas the deficiencies do not cause substantial problems. In narrow and cluttered surroundings, however, they often cause the robot to stop abruptly, turn in place, etc.

Unfortunately, the navigation behavior is the result of the subtle interplay of many complex factors. These factors include the robot's dynamics, sensing capabilities, surroundings, parameterizations of the control program, etc. It is impossible to provide the robot with a deep model for diagnosing navigation behavior. Instead, we equip the robot with simple heuristic rules such as narrow passages sometimes provoke the robot to navigate with low translational velocity or to stop frequently. Then, the robot is to detect which narrow passages might cause such effects and estimates their extent from experience and by analyzing its behavior.

Let us consider the process of behavior diagnosis more carefully. Upon carrying out a navigation task, RHINO produces a behavior trace like the one shown in Figure 5. The robot's position is depicted by a circle where the size of the circle is proportional to the robot's translational speed. Figure 6(left) shows those behavior stretches where the robot's translational velocity is low. This is an example of a ***behavior feature subtrace***, a subtrace that represents a feature of the robot's behavior. Figure 6(right) shows the behavior stretches that are classified as traversing a narrow passage. This is an ***environment feature subtrace***.

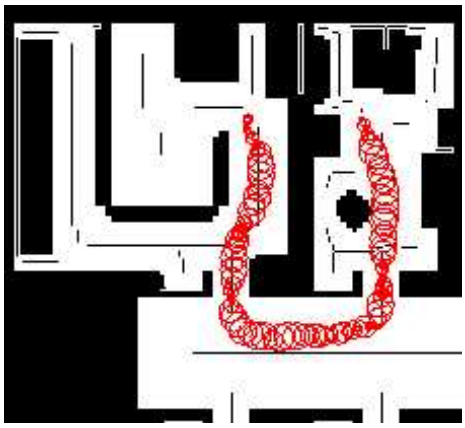

Figure 5. Visualization of a behavior trace.

Behavior feature subtraces such as low and high translational speed, turning in place, and frequent stopping often hint at which behavior stretches can be improved. To infer how the improvements might be achieved, XFRMLEARN first tries to explain a behavior feature subtrace by finding environment feature subtraces that overlap with it. Environment features include being in a narrow passage, in free space, or close to an obstacle. We define the percentage of the behavior subtrace that lies in the environment feature subtrace the degree to which the environment feature explains the behavior. If the percentage of overlap is high enough, then the behavior is considered to be a ***behavior flaw***. Behavior flaws have a ***severity*** that is an estimate of the performance gain that could be achieved, if the flaw would be completely eliminated. Finally, we use the predicate MAYCAUSE(F1,F2) to specify that the environment feature F1 may cause the behavior feature F2 like narrow passages causing low translational velocity.

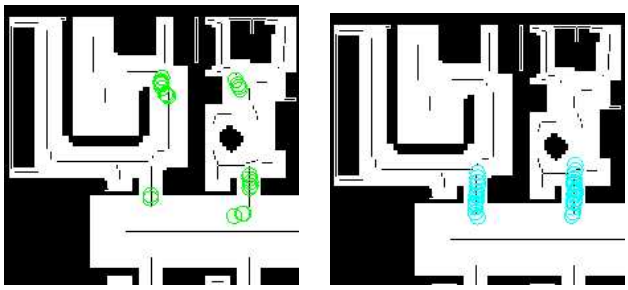

Figure 6. Visualization of a LOW-T-VEL subtraces (left), and the subtraces that correspond to the traversal of narrow passages (right).

Based on the vocabulary introduced above, the diagnosis step is realized through a simple diagnostic rule shown in figure 7. Depending on the instantiation of MAYCAUSE(?F1,?F2) the rule can diagnose different kinds of flaws:

D-1 Low translational velocity is caused by the traversal of a narrow passage (MAYCAUSE(NARROWPASSAGE,LOW-T-VEL)).
D-2 Stopping is caused by the traversal of a narrow passage.
D-3 Low translational velocity is caused by passing an obstacle too close (MAYCAUSE(CLOSEOBSTACLE,LOW-T-VEL)).
D-4 Stopping is caused by passing an obstacle too close.
D-5 High target velocity is caused by traversing free space.

if *BehaviorTrace(?trace)* $\wedge$ *MayCause(?f2,?f1)*
$\wedge$ *BehaviorFeatureSubtrace(?f1,?s1,?trace,?severity)*
$\wedge$ *EnvironmentFeatureSubtrace(?f2,?s2,?trace)*
$\wedge$ *Overlap(?s1,?s2,?degree)* $\wedge$ *?degree* $> \theta$
then *infer* *BehaviorFlaw* *with*

CATEGORY	*= CausedBy(?f1,?f2)*
PLAN	*= ?srnp*
SUBTRACE	*= ?s1*
SEVERITY	*= ?severity*
EXPLAINABILITY	*= ?degree*
EXPLANATION	*= ?s2*

Figure 7. Diagnostic rule that realizes the diagnosis step.

4.2 The "Revise" Step

The "revise" step uses programming knowledge about how to revise navigation plans and how to parameterize the subsymbolic navigation modules that is encoded in the form of plan transformation rules. In their condition parts transformation rules check their applicability

and the promise of success. These factors are used to estimate the expected utility of rule applications. XFRMLEARN selects the rule with a probability proportional to the expected utility and applies it to the plan.

transformation rule *traverseNarrowPassageOrthogonally*
to eliminate *?behavior-flaw* *with*
CATEGORY = *CausedBy(LowTVel, NarrowPassage)*
PLAN = *?srnp*
SUBTRACE = *?subtrace1*
SEVERITY = *?severity*
EXPLAINABILITY = *?degree*
EXPLANATION = *?subtrace2*
if *NavigationStep (?subtrace2,traverseNarrowPassage(?p1,?p2))*
∧ *applicability(traverseNarrowPassageOrthogonally, CausedBy(LowTVel,NarrowPassage), ?applicability),*
then *with* *expected-utility = ?severity * ?degree * ?applicability*
insert subplan
TRAVERSE-NARROW-PASSAGE*(?p1,?p2)*
path constraints { *?p1,?p2* }
justification *?behavior-flaw*

Figure 8. Transformational learning rule **R-1**.

Figure 8 shows a transformation rule indexed by a behavior flaw *CausedBy(NarrowPassage)*. The rule is applicable if the subtrace *?subtrace* traverses a narrow passage *traverseNarrowPassage(?p1,?p2)*. The application inserts a subplan into the SRNP that constrains the path to cause the robot to traverse a narrow passage orthogonally and with maximal clearance.

The main piece of work is done by the condition NAVIGATIONSTEP(?SUBTRACE,TRAVERSE NARROWPASSAGE(?P1,?P2)), which computes a description of the narrow passage that ?SUBTRACE traverses. ?P1 is then instantiated with the point of the main axes Voronoi skeleton that is next to the first entry in ?SUBTRACE and ?P1 with the one that corresponds to leaving the narrow passage. Using the lines of the main axes Voronoi skeleton to characterize navigation steps achieves more canonical representations of the discretized subtraces and thereby facilitate the identification of identical subtraces.

For the purpose of this paper, XFRMLEARN provides the following revisions:

R-1 If the behavior flaw is attributed to the traversal of a narrow passage (**D-1**) then insert a subplan that causes the robot to traverse the passage orthogonally and with maximal clearance.
R-2 Switch off the sonar sensors while traversing narrow passages if the robot repeatedly stops during the traversal (indexed by **D-2**).
R-3 Insert an additional path constraint to pass a closeby obstacle with more clearance (indexed by **D-3,D-4**).
R-4 Increase the target velocity for the traversal of free space where the measured velocity almost reaches the current target velocity (indexed by **D-5**).
R-5 Insert an additional path constraint to avoid abrupt changes in the robot's heading (indexed by **D-2**).

Estimating the Expected Utility of Revisions. Because XFRMLEARN's transformation rules are heuristic, their applicability and the performance gain that can be expected from their application is environment and task-specific. Therefore an important aspect in behavior adaption is to learn the environment and task specific expected utility of rules based on experience.

The expected utility of a revision *r* given a behavior flaw *b* is computed by

$$EU(r|diagnosed(b)) = P(success(r)|correct(b)) * s(b),$$

where b is a diagnosed behavior flaw, $s(b)$ is the severity of the behavior flaw (the performance gain, if the flaw would be completely eliminated), $P(success(r)|correct(b))$ is the probability that the application of revision r improves the robot's performance significantly given that the diagnosis of b is correct. This formula can be derived from the equation

EU(r|diagnosed(b))
*= EU(r|correct(b)) * P(correct(b)|diagnosed(b))*

In the current version of XFRMLEARN we assume that the diagnoses are correct. In this case, we can transform the equation into

EU(r|diagnosed(b))
*= P(success(r)|correct(b)) * U(success(r))*
*+ P(¬success(r)|correct(b)) * U(¬success(r))*

Because XFRMLEARN rejects revisions that do not significantly improve the robot's behavior, the utility of unsuccessful revisions is 0. Furthermore, we assume that $U(success(r))$ is proportional to the severity of the behavior flaw b. This leads to the first equation.

For computing the probability $P(success(r)|correct(b)) = P(success(r)|diagnosed(b))$ (under our assumption above) XFRMLEARN maintains a simple statistic about successful and unsuccessful rule applications. Let $h^+(r,b)$ be the number of times that r successfully revised a SRNP given the behavior flaw b and $h^-(r,b)$ the number of times that the revision was not successful. Then

$$\frac{h^+(r,b)}{h^+(r,b) + h^-(r,b)} \quad (1)$$

is a crude measure for $P(success(r)|correct(b))$. $h^+(r,b)$ and $h^-(r,b)$ are initialized to 1.

In general, the first equation overestimates the expected utility of a revision but tends to select, with respect to our practical experience, the same revisions that programmers propose when visually inspecting the behavior traces.

The estimate of the expected utility of the second most promising revision is used as an estimate of how interesting the current navigation task or the SRNP for this task respectively is for future examinations by XFRMLEARN.

4.3 The "Test" Step

Because plan transformation rules check their applicability and parameterization with respect to idealized models of the environment, the robot, the perceptual apparatus, and operation of the subsymbolic navigation system, XFRMLEARN cannot guarantee any improvements of the existing plan. Therefore, XFRMLEARN tests the resulting candidate plans against the original plan by repeatedly running the original and the revised plan several times[2] and measuring the time performance in the local region that is affected by the plan

[2] In our experiments we run both plans 7 times.

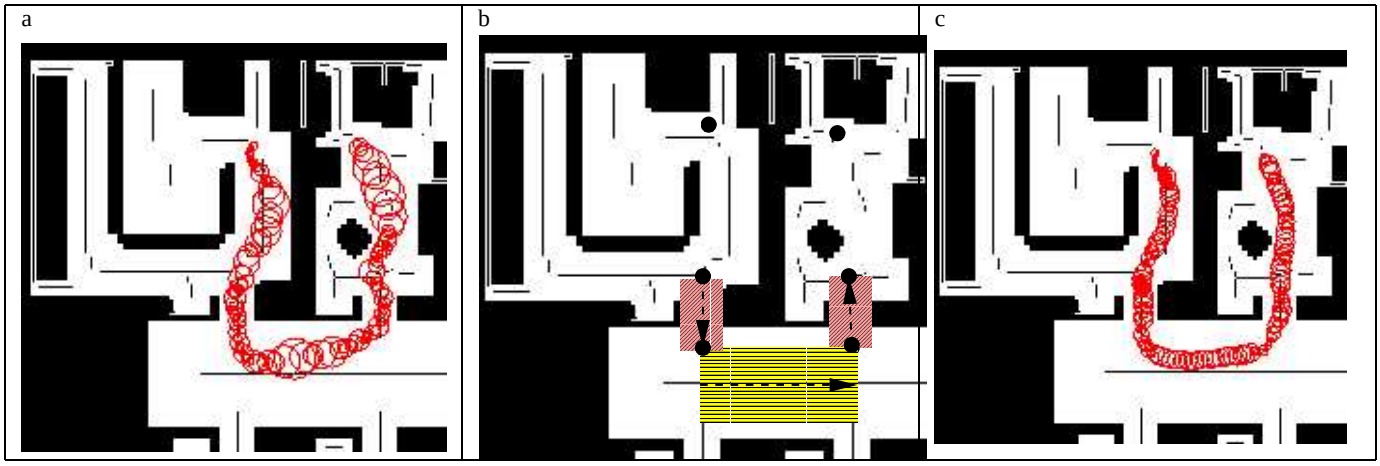

Figure 9. Behavior trace of the default plan (a). Low T-Vel subtraces (b). Learned SRNP (c).

transformation. The new candidate plan is accepted, if based on the experiments there is a 95% confidence that the new plan performs better than the original one. This test only checks whether a revision has locally improved the performance in a statistically significant way. Note that we also use a t-test to show that a sequence of revisions globally improves the robot's performance for a given navigation task with statistical significance.

5 Experimental Results

To empirically evaluate XFRMLEARN we have performed two long term experiments in which XFRMLEARN has improved the performance of the RHINO navigation system, a state-of-the-art navigation system, for given navigation tasks by up to 44 percent within 6 to 7 hours. Our conclusions are based on these two learning sessions because current robot simulators, which would allow for more extensive experimentation, do not model robot behavior accurately enough to perform realistic learning experiments.

	ORIGINAL	LEARNED
Min.	85.2	67.5
Max.	156.7	92.6
Range	71.5	25.0
Median	111.0	76.5
Mean	114.1	78.2
Var.	445.2	56.5
Dev.	21.1	7.5

Figure 10. Descriptive statistics for the first learning experiment.

A summary of the first session is depicted in Figure 9. Figure 9(a) shows the navigation task (going from the desk in the left room to the one in the right office) and a typical behavior trace generated by the MDP navigation system. Figure 9(b) visualizes the plan that was learned by XFRMLEARN. It contains three subplans. One for traversing the left doorway, one for the right one, and one for the traversal of the hallway. The ones for traversing the doorways are TRAVERSENARROWPASSAGE subplans, which comprise path constraints (the black circles) as well as behavior adaptations (depicted by the region). The subplan is activated when the region is entered and deactivated when it is left. A typical behavior trace of the learned SRNP is shown in Figure 9(c). We can see that the behavior is much more homogeneous and that the robot travels faster. This visual impression is confirmed by the descriptive statistics of the comparison (Figure 10). The t-test for the learned SRNP being at least 24 seconds (21%) faster returns a significance of 0.956. A bootstrap test returns the probability of 0.956 that the variance of the performance has been reduced.

Step	Rule	Applied To	Accepted?
1	R-1/2	door right	yes
2	R-1/2	door left	yes
3	R-4	hallway	yes
4	R-4	room left	no
5	R-3	room left	no

Figure 11. Summary of the revisions performed by XFRMLEARN in the learning experiment.

In the session XFRMLEARN has proposed and tested five plan revisions, three of them were successful. The unsuccessful ones proposed that the low translational velocity at the beginning of the behavior trace was caused by the robot passing an obstacle too closely. Indeed it was caused by the robot not facing the direction it was heading to and the inertia of the robot while accelerating. The second unsuccessful revision proposed to go faster in the open area of the left office. The performance gain obtained by this revision was negligible. Figure 11 summarizes the proposed revisions and whether they have been successful or not.

Figure 12 summarizes the results from the second learning session as descriptive statistic. The average time needed for performing a navigation task has been reduced by about 95.57 seconds (44%). The t-test for the revised plan being at least 39 seconds (18%) faster returns a significance of 0.952. A bootstrap test returns the probability of 0.857 that the variance of the performance has been reduced. The successful revisions of the second learning session are visualized in Figure 13.

We consider the performance gains that XFRMLEARN has

	ORIGINAL	LEARNED
Min.	145.3	86.1
Max.	487.6	184.5
Range	342.2	98.4
Median	181.5	108.6
Mean	212.6	119.0
Var.	9623.1	948.0
Dev.	98.0	30.7

Figure 12. Descriptive statistics for the second learning session.

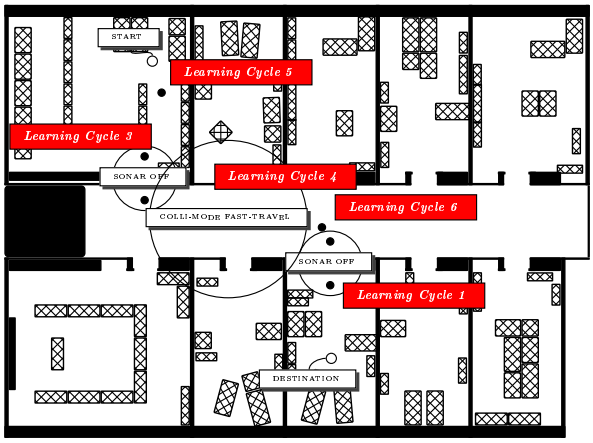

Figure 13. The transformations that were considered in the learning cycles of the learning session.

achieved in the learning sessions described in this section as being very impressive, for several reasons: First, the diagnosis rules as well as the transformation rules are general and are not specifically tailored to a particular system, e.g. the RHINO system. Furthermore, XFRMLEARN has only used observable symptoms such as driving slowly or fast for the detection of behavior flaws. It did not have access to the (system specific) execution states of the collision avoidance module which would have provided more informative and reliable evidence for the diagnosis of behavior flaws. Second, we have applied XFRMLEARN to a low-level navigation system which is well tuned and has been developed in the same environment and which has performed impressively in challenging longterm navigation experiments [TBB+99]. We expect the performance gains for insufficiently configured navigation systems to be much higher.

6 Discussion

In this section we discuss three questions that we feel critical for the evaluation of the XFRMLEARN approach. These questions are: (1) Are the time resources needed for the learning sessions justified by the performance gains obtained through them? (2) Is the performance gain not only due to a small set of heuristic and highly tailored rules? (3) How can generalization in learning be achieved?

6.1 Time Requirements

In the two learning sessions reported in Section 5 XFRMLEARN improved the average navigation performance by 29 % and 44 %, respectively. While these performance gains are quite impressive, the time required for each learning step appears to be very high: to test one SRNP revision takes about 60 minutes and a complete learning session for one navigation task takes about 6-7 hours. This is due to the use of a t-test as the criteria for deciding whether or not a given revision improves the robot's performance. The time is mainly consumed by the repeated and alternated execution of the original and the new candidate plan that resulted from a plan revision. Are such time consuming tests for making single learning steps really necessary?

In our view, the answer is yes, and the reason why is the enourmous variance in the robot's behavior. In our experiments, the time the robot takes to accomplish a given navigation task depends strongly on the occurrence of sonar cross talk while traversing doorways. Whether or not this sonar cross talk really occurs depends in turn on the exact trajectory of the robot's path, dynamic obstacles, and other subtle interfering factors. Even worse, the variance, at least in our control architecture, is not only caused by factors that can, in principle, be controlled by the robot but also by factors like the distribution of control processes over a local network, the loads on the computers, and the traffic on the network. The amount of variance that can be caused this way is shown in two test series of the same two SRNPs that we have performed at different days (see Figure 6.1).

DAY 1		DAY 2	
ORIGINAL	LEARNED	ORIGINAL	LEARNED
127.3	107.8	265.9	184.5
172.2	112.9	148.5	97.9
131.4	110.8	487.6	108.6
169.2	108.0	195.3	138.3
122.0	106.8	167.4	95.6
220.2	106.8	145.3	93.6

Figure 14. Variance of RHINO's performance controlled by the same SRNP on two different days.

Therefore, we believe that a learning system that aims at improving a state-of-the-art robot navigation system for a real world application must take this variance explicitly into account. In such settings competing learning approaches such as reinforcement learning will have trouble to converge to a better policy when features that cause variance in the robot's behavior are not represented in the state space. This, however, is often impossible or, if possible, leads to extremely huge state spaces.

Note that we restrict the test of the success of a revision only to a local envelope of the region that is directly affected by the transformation. This allows us to ignore the variance in the robot's behavior in other regions that could not have been affected by the current revision. Furthermore, it is sometimes necessary to eliminate a superposition of behavior flaws that occurred in the same region by applying a combination of revision rules (in our example **R-1** together with **R-2**). Our experiments have shown that in many cases neither rule **R-1** nor rule **R-2** alone were capable of improving the traversal of doorways significantly (using 7 samples) whereas the revision that applied both rules simultaneously could do so with very high significance.

We hope that we can reduce the time requirements for a single learning session, possibly by a factor 3-4, in the future if (1) a navigation task and its inverse task are learned together and (2) several hypotheses are tested simultaneously, which is possible when using local tests.

6.2 Heuristic Rules

The diagnosis and the revision rules we have used are quite simple and heuristic in nature. Definitely, more sophisticated rules could be specified, which could probably achieve even higher performance gains. However, the specification of such sophisticated rules is not the main focus of this paper. We rather introduce a general framework in which heuristic rules can be specified easily and used for improving a mobile robot's navigation performance and simultaneously learn navigation plans that can be subject of symbolic reasoning. This task is very hard to achieve without using prior knowledge as specified in the heuristic rules.

In future work, we will investigate more generic ways of specifying such diagnosis and transformation rules by using the parameters that specify MDP (navigation) problems and their states. We expect these methods to achieve even higher performance gains at the cost of making more assumptions about the robot's control software. In this paper, however, we use only rules that are not specifically tailored for a particular system. These general rules assume little knowledge about the low-level navigation system and can therefore be adapted with little effort to other environments, robots, and tasks.

6.3 Generalization in Learning

An important design criterion for learning systems is how much they should generalize from their experiences. There is a fundamental trade-off between the possible benefits of generalizing from past experiences to possible future experiences and the danger of overgeneralization. In this paper we have restricted ourselves to considering the problem of learning SRNPs for instances of navigation tasks instead of learning e.g. a universal SRNP for all possible tasks in the environment. XFRMLEARN applies revisions to SRNP instances and tests their success by comparing SRNP instances. This way XFRMLEARN avoids the problem of overgeneralization but also cannot exploit the advantages of generalization, that is apply revisions that have proven to be successful for some kind of behavior stretches to similar behavior stretches.

To achieve a better generalization instead of SRNPs rules together with their conditions of successful application could be learned. So far, we haven't got a good enough understanding of how much testing is necessary to limit the risk of overgeneralization to a reasonable amount. We intend to research these issues more carefully in our future work.

7 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 step towards learning default plan libraries. XFRMLEARN differs from other 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 but (partly) generates the symbolic plans itself.

A number of researchers consider navigation as a (Partially Observable) Markov decision problem (MDPs) [TBB+98, KCK96] and compute *policies*, mappings from discretized robot poses into fine-grained navigation actions, that cause the robot to reach its destination quickly and reliably. This approach has been successfully applied to the problem of robot navigation. Though MDP path planners produce a behavior that is robust and optimized for average performance, they are not suitable for handling exceptional situations. We have also discussed in the Introduction several advantages of hierarchically structured plans for high-level control of complex continuous navigation behavior.

In our research we combine (PO)MDP planning and transformational planning of reactive behavior. MDP path planning is used for generating robust navigation behavior optimized for average performance and transformational planning for predicting and forestalling behavior flaws in contingent situations. XFRMLEARN bridges the gap between both approaches: it learns symbolic plans from executing MDP policies and it could be used to decompose MDP problems and specify the dimensions and the grain size of the state space for the subproblems [Moo94]. Our approach has been inspired by Sussman's thesis that failures (bugs) are valuable resources for improving the performance of problem-solving agents [Sus77].

Our work is also related to the problem of learning actions, action models or macro operators for planning systems. Schmill *et al.* recently proposed a technique for learning the pre- and postconditions of robot actions. Sutton *et al.* [SPS98] have proposed an approach for learning macro operators in the decision-theoretic framework using reinforcement learning techniques. 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.

8 Conclusions

We have described XFRMLEARN, a system that learns SRNPs, symbolic behavior specifications that (a) improve the navigation behavior of an autonomous mobile robot generated by executing MDP navigation policies, (b) make the navigation behavior more predictable, and (c) are structured and transparent so that high-level controllers can exploit them for demanding applications such as office delivery.

XFRMLEARN is capable of learning compact and modular SRNPs that mirror the relevant temporal and spatial structures in the continuous navigation behavior because it starts with default plans that produce flexible behavior optimized for average performance, identifies subtasks, stretches of behavior that look as if they could be improved, and adds subtask specific subplans only if the subplans can improve the navigation behavior significantly.

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. Our approach also takes a particular view on the integration of symbolic and subsymbolic control 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 specifications symbolically, transparently, and modularly. In our approach, (PO)MDPs are viewed as a way to ground symbolic representations.

Acknowledgments. The research reported in this paper is partly funded by the Deutsche Forschungsgemeinschaft (DFG) under contract number BE 2200/3-1.

REFERENCES

[BBFC98] M. Beetz, W. Burgard, D. Fox, and A. Cremers. Integrating active localization into high-level control systems. *Robotics and Autonomous Systems*, 23:205–220, 1998.

[Bee99] M. Beetz. Structured reactive controllers — a computational model of everyday activity. In *Proceedings of the Third International Conference on Autonomous Agents*, 1999.

[BG00] M. Beetz and H. Grosskreutz. Probabilistic hybrid action models for predicting concurrent percept-driven robot behavior. In *Proceedings of the Fifth International Conference on AI Planning Systems*, Breckenridge, CO, 2000. AAAI Press.

[FBT97] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 1997.

[KCK96] L. Kaelbling, A. Cassandra, and J. 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.

[LMA98] N. Leash, N. Martin, and J. Allen. Improving big plans. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[LRN86] J. Laird, P. Rosenbloom, and A. Newell. Chunking in soar: the anatomy of a general learning mechanism. *Machine Learning*, 1:11–46, 1986.

[McD92] D. McDermott. Transformational planning of reactive behavior. Research Report YALEU/DCS/RR-941, Yale University, 1992.

[Mit90] T. Mitchell. Becoming increasingly reactive (mobile robots). In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 1051–1058, Boston, MA, 1990. MIT Press.

[Moo94] A. Moore. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 711–718. Morgan Kaufmann Publishers, Inc., 1994.

[SPS98] 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.

[Sus77] G. Sussman. *A Computer Model of Skill Acquisition*, volume 1 of *Aritficial Intelligence Series*. American Elsevier, New York, NY, 1977.

[TBB+98] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt. 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.

[TBB+99] S. Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Haehnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Minerva: A second generation mobile tour-guide robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'99)*, 1999.