Learning Predictive Knowledge to Optimize Robot Motor Control

Freek Stulp, Alexis Maldonado and Michael Beetz

***Abstract*— By observing the execution of their actions, cognitive robots become aware of their behavior. We describe a system that acquires such knowledge, and uses it to reflect future actions to autonomously avoid failures and optimize future actions. The system has three types of motor control knowledge, which are represented at different levels of abstraction, and acquired in different ways: 1) declarative knowledge to select actions, 2) procedural knowledge to execute actions, and 3) predictive knowledge to optimize action executions with respect to execution duration and success. The robots acquire predictive knowledge autonomously, by learning it from observed experience. The generality of the approach is demonstrated by applying the methods to three robotic platforms: a Pioneer soccer robot, a simulated articulated B21 in a kitchen environment, and a PowerCube arm.**

I. Introduction

In robotics, it is common to provide robots with a library of durative actions relevant to the task domain. In soccer for instance, robots will typically have dribbling and passing actions. Actions constitute the procedural knowledge of a robot. Actions are also referred to as skills, behaviors, motor primitives, schemas, options, or macros [16].

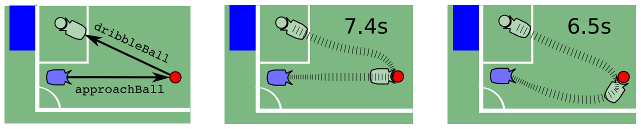
To reason about their actions, cognitive systems must have models of their actions. A model of an action predicts the outcome or performance of an action in a given task context. There is evidence for the widespread use of predictive action models in human motor control, for instance in state estimation and sensory cancellation, state prediction, context estimation, and optimal control [26], [13], [17].

In this paper, we consider declarative and learned action models, and describe how this knowledge is acquired, represented and applied in our robot control system. First of all, declarative knowledge about actions is used to reason about actions at an abstract level. By considering the effects of actions, actions can be selected, combined and concatenated, so as to achieve more complex tasks that individual action could not achieve alone. Due to its abstract nature, it is feasible to specify ('declare') declarative knowledge by hand. For instance, being in possession of the ball in front of the goal in the typical soccer scenario in Figure 1(a) can be achieved by "First, approach the ball, and then, dribble it towards the goal."

This declarative statement is executed by consequently executing the two corresponding actions `approachBall` and `dribbleBall` with the task-specific parameters. These actions represent the procedural knowledge of the robot. If the robot naively executed the first action `approachBall`,

Intelligent Autonomous Systems Group, Technische Universität München, Boltzmannstrasse 3, D-85747 Garching bei Munich, Germany, {stulp,maldonad,beetz}@cs.tum.edu

The research reported in this paper is partly funded by the CoTeSys cluster of excellence (Cognition for Technical Systems, www.cotesys.org)



(a) Selecting appropriate action. (b) A naive execution of the plan. (c) A time-optimal execution of the plan.

Fig. 1. Selecting, executing and optimizing actions with declarative, procedural and predictive knowledge respectively.

it might arrive at the ball with the goal at its back, as depicted in Figure 1(b). This is an unfortunate position from which to start dribbling towards the goal. An abrupt transition occurs between the actions, as the robot needs to brake to slowly and carefully dribble the ball towards the goal.

What we would like the robot to do instead is to go to the ball *in order* to dribble it towards the goal afterwards. The robot should, as depicted in the Figure 1(c), perform the first action sub-optimally in order to achieve a much better position for executing the second plan step. The behavior shown in Figure 1(c) exhibits seamless transitions between actions and has higher performance, achieving the ultimate goal in less time.

Apparently, mapping the declarative statement to the corresponding actions raises some questions: 'What is the best angle of approach?' or also 'Will I collide with the ball before achieving the desired angle?' The answers to these questions are not relevant at an abstract declarative level, but *are* relevant to robust and efficient execution.

We demonstrate that predictive knowledge, in the form of learned action models, enables robots to answer these questions autonomously in real-time. For instance, if the robot could predict that the execution as in Figure 1(c) is faster than Figure 1(b), it could commit to the action parameterization of the faster one. As it is difficult to hand-code action models, robots acquire action models by observing the executions of actions, and learning a general model from these observations with tree-based induction.

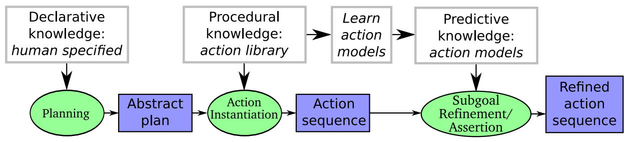

Declarative knowledge: human specified
Procedural knowledge: action library
Learn action models
Predictive knowledge: action models
Planning
Abstract plan
Action Instantiation
Action sequence
Subgoal Refinement/ Assertion
Refined action sequence

Fig. 2. Interactions between the three types of knowledge.

The interactions between the three types of knowledge are depicted in the system overview in Figure 2. When faced with a task, the first step is to use human-specified declarative knowledge to generate an abstract plan to solve the task. Then, this plan is mapped to the actions in the action library, which constitutes the procedural knowledge. Finally, with two procedures called subgoal assertion and subgoal refinement, the exact execution of the actions is refined and optimized with learned action models, which constitute the predictive knowledge. In a preceding off-line step, the action models are learned from observed experience.

The main contributions of this paper are: 1) demonstrating how action models enable robots to autonomously optimize plans that are generated with declarative knowledge. 2) proposing methods with which robots can learn to predict action performance and outcome based on observed experience 3) empirically verifying that this leads to more efficient, robust, and effective behavior.

We apply our approach to three robotic domains, depicted in Figure 3: 1) robotic soccer, using both the real and simulated Pioneer I robots of our RoboCup mid-size league team the 'AGILO RoboCuppers' [3]. 2) service robotics, with a simulated B21 in a kitchen environment. 3) arm control, using two degrees of freedom of a PowerCube arm from Amtec Robotics.

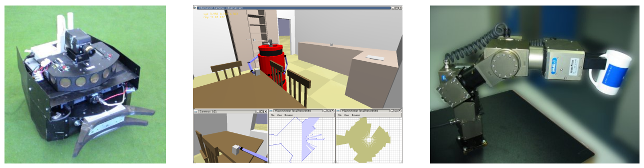


Fig. 3. The three robotic platforms used in this paper.

The rest of this paper is structured as follows. First, we describe how each of the three types of control knowledge is represented and acquired in Section II. In Section III, we described how declarative and procedural knowledge are combined to generate executable plans, and in Sections IV and V, we demonstrate how predictive knowledge enables robots to optimize plan execution with subgoal refinement and subgoal assertion. Related work is presented in Section VI, and we conclude with Section VII.

II. CONTROL KNOWLEDGE REPRESENTATION

In this section, we describe how our robots represent and acquire procedural, declarative and predictive control knowledge.

A. Procedural knowledge

Procedural knowledge consists of a library of durative actions. Actions generate streams of primitive motor commands, and encapsulate knowledge about how certain goals can be achieved in certain task contexts. For instance, human and robot soccer players will typically have dribbling, kicking, and passing actions, that achieve different goals within different soccer contexts. Because actions only apply to limited task contexts, they are easier to design or learn than a controller that must be able to deal with all possible contexts [13], [15]. Libraries of actions are used extensively in robotics control [16].

In cognitive science, the conceptual equivalent to actions are inverse models [26], [21]. In [13], human motor control is modeled as the interaction of a library of inverse models, paired with forward models.

In our system implementation, actions are hand-coded C++ or Python functions. As the approach described in this paper is independent of action implementations, we do not elaborate on them here, but refer to Appendix A of [24].

B. Declarative knowledge

Declarative structure captures abstract knowledge about tasks [12]. It is used to select the appropriate actions for a given task.

In cognitive science, the difference between declarative and procedural knowledge was dramatically witnessed in the patient H.M. [22] After a severe brain operation, H.M. was unable to store novel facts in declarative memory, but *was* able to acquire new skills. For instance, with training H.M. improves and becomes skilled at tracing geometrical shapes seen only in the mirror, but when asked, reports having no recollection of ever having done such as task before.

In our system implementation, the Planning Domain Description Language PDDL2.1 [10] is used to describe abstract actions, as well as goals, states and plans. The declarative knowledge about an action consists of its preconditions and effects (add- and delete-list), as depicted in Figure 4. The PDDL action library is specified manually by the designer. The main reason is that the high level of abstraction of PDDL enables us to easily transfer our knowledge about how we would solve a task to the robot controller. Also, such knowledge is difficult for robots to obtain autonomously. Other recent examples of robot controllers that use such abstract action descriptions for plan generation are [7], [5].

```
(:action approachball
   :parameters (?from ?to)
   :precondition (and (robot ?from) (ball ?to) )
   :effect (and (not (robot ?from)) (atball ?to) ))
```

Fig. 4. PDDL specification of an action.

C. Predictive knowledge

In cognitive science, there is a distinction between inverse models, which map desired states to motor commands, and forward models, which map motor commands to state changes [26]. Forward models are not entities that are fixed at birth, but that must rather be learned and updated through experience. This allows forward models to be learned for new action contexts, or for newly acquired actions. In this paper and in [9], actions and action models are similar to inverse models and forward models respectively.

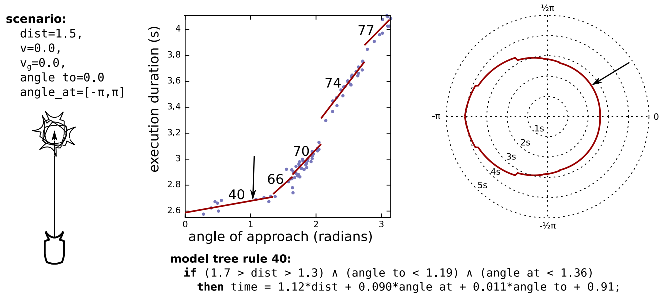
In the system implementation, action models take the same parameters as their corresponding action, and return the predicted execution duration or success. The advantage of learning action models over analytical methods is that it is based on real experience, and therefore takes all factors relevant to performance into account. Also, many hand-coded actions are difficult to formalize analytically, or analysis is impossible because the inner workings of the action are unknown [2].

Training examples for learning action models are gathered by executing an action, and observing the results. The running example in this section will be learning to predict the execution duration of the `goToPose` action for the AGILO robot. The robot first executes the `goToPose` action 290 times, for random positions on the field. Before learning, the data is transformed to a feature space appropriate for the action. Data acquisition continues until the error of the learned model on a separate test set stabilizes. How many executions are actually necessary to learn an accurate model is domain dependent, and will be presented in Section IV-A.

Then, decision trees and model trees are trained with this data to acquire a general model of the action. Decision trees are functions that map continuous or nominal input features to a nominal output value. The function is learned from examples by a piecewise partitioning of the feature space. One class is chosen to represent the data in each partition. Model trees are a generalization of decision trees, in which the nominal values at the leaf nodes are replaced by line segments. A line is fitted to the data in each partition with linear regression. This linear function interpolates between data in the partition, which enables model trees to approximate continuous functions. For more information on decision and model trees, and how they are learned with tree-based induction, we refer to [19], [24].

1) Action Model: Execution Duration Prediction: To visualize action models learned with model trees, an example of execution duration prediction for a specific situation is depicted in Figure 5. This model for the `goToPose` action in the soccer domain (real robots) is learned from 290 episodes.

In the situation depicted in Figure 5, the variables $dist$, $angle\_to$, v, and v_g are fixed to 1.5m, 0°, 0m/s, and 0m/s for visualization purposes only. For these values, Figure 5 shows how the predicted time depends on $angle\_at$, once in a Cartesian, once in a polar coordinate system. The plots consists of five line segments. This means that the model tree has partitioned the feature space into five areas, each with its own linear model. Below the two plots, one of the learned model tree rules that applies to this situation is displayed. Arrows indicate this linear model in the plots.



scenario:
```
  dist=1.5,
  v=0.0,
  vg=0.0,
  angle_to=0.0
  angle_at=[-π,π]
```

model tree rule 40:
```
  if (1.7 > dist > 1.3) ∧ (angle_to < 1.19) ∧ (angle_at < 1.36)
    then time = 1.12*dist + 0.090*angle_at + 0.011*angle_to + 0.91;
```

Fig. 5. An example situation, two graphs of time prediction for this situation with varying $angle\_at$, and the model tree rule for one of the line segments.

2) Action Model: Failure Prediction: The simulated soccer robots also learn to predict failures in approaching the ball with the `goToPose` action. A failure occurs if the robot collides with the ball before the desired state is achieved. The robots again learn an action model from experience. To acquire experience, the robot executes `goToPose` a thousand times, with random initial and goal states. The ball is always positioned at the destination state. The initial and goal state are stored, along with a flag that is set to `Fail` if the robot collided with the ball before reaching its desired position and orientation, and to `Success` otherwise.

The learned decision tree and a graphical representation, are depicted in Figure 6. The goal state is represented by the robot, and different areas indicate if the robot can reach this position with `goToPose` without bumping into the ball first. Remember that `goToPose` has no awareness of the ball at all. The model simply predicts when its execution leads to a collision or not. Intuitively, the rules seem correct. When coming from the right, for instance, the robot always clumsily stumbles into the ball, long before reaching the desired orientation.

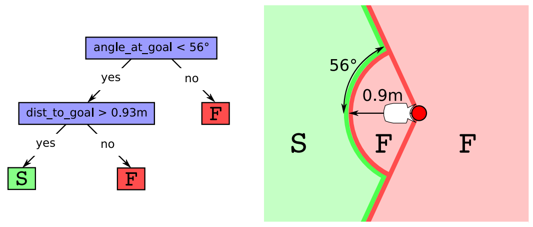


Fig. 6. The learned decision tree that predicts if a collision will occur.

3) Empirical Evaluation: The different domains and actions for which action models of execution duration are learned are listed in the first two columns of Table I. The subsequent columns list the number of episodes executed to gather data for the training set n, the mean execution duration per episode $\bar{t}$, the total duration of data gathering for the training set $\bar{t} \cdot n$, as well as the model's error (MAE) on a separate test set with $\frac{1}{3}n$ episodes. In the next sections, we demonstrate that these action models are accurate enough to enable a significant improvement of action execution.

Robot	Action	n	$\bar{t}$	$\bar{t} \cdot n$	MAE
			(s)	(h:mm)	(s)
Pioneer I	`goToPose`	290	6.4	0:31	0.32
(Real)	`dribbleBall`	202	7.7	0:26	0.43
Pioneer I	`goToPose`	750	6.2	1:18	0.22
(Simulated)	`dribbleBall`	750	7.4	1:32	0.29
B21	`goToPose`	2200	9.0	5:45	0.52
	`reach`	2200	2.6	1:38	0.10
PowerCube	`reach`	1100	2.9	0:53	0.21

TABLE I

LIST OF ACTIONS AND THEIR ACTION MODEL ACCURACIES.

To evaluate the accuracy of the ball approach failure model, the robot executes another thousand runs. The resulting confusion matrix is depicted in Table II. The decision tree predicts collisions correctly in almost 90% of the cases.

		Observed		Total
		Fail	Success	Predicted
Predicted	Fail	51%	10%	→ 61%
	Success	1%	38%	→ 39%
		↓	↓	↘
	Total Observed	52%	48%	89%

TABLE II
CONFUSION MATRIX FOR BALL COLLISION PREDICTION.

III. GENERATING EXECUTABLE PLANS

We now describe how executable plans are generated using declarative and procedural knowledge. In the subsequent two sections, we show how the robots optimize these plans using predictive knowledge.

Abstract plans are generated using the declarative knowledge about the actions. For this, we use the Versatile Heuristic Partial Order Planner [27][1]. Apart from the PDDL representations of the actions, an abstract goal and state represented in PDDL are also passed as an input to VHPOP. Usually, the abstract state is derived from the continuous belief state of the robot through a process called anchoring [8]. As in the International Planning Competition, we consider a limited number of scenarios, enabling us to specify the scenarios in PDDL in advance.

The output of VHPOP is a list of symbolic actions and causal links, as depicted in Figure 7. This plan corresponds to the example in Figure 1. Causal links specify which action was executed previously to achieve an effect which meets the precondition of the current action. A chain of such abstract actions represents a valid plan to achieve the goal.

```
Initial 0 : (robot pos1) (ball pos2) (final pos3)

Step 2    : (approachball pos1 pos2)
           0   -> (robot pos1)
           0   -> (ball pos2)

Step 1    : (dribbleball pos2 pos3)
           2   -> (atball pos2)

Goal      :
           0   -> (final pos3)
           1   -> (atball pos3)
```

Fig. 7. The output of VHPOP is a PDDL plan with causal links.

The chain of abstract actions in Figure 7 represents the declarative knowledge needed to achieve the goal. The next step is to map declarative knowledge to the executable actions in the action library, i.e. the procedural knowledge.

PDDL plans are instantiated with executable actions by first extracting symbolic actions and causal links in the plan, and then instantiating the symbolic actions one by one. For each symbolic action, the executable action is retrieved by its name. For instance, the PDDL action '`dribbleball`' is simply mapped to the C++ action '`dribbleBall`'. Then, the symbolic parameters of the PDDL action (e.g. `pos1`) must be mapped to the continuous parameters of the executable action (e.g. x,y,ϕ,v), which are read from the robot's continuous belief state. Note that the symbolic parameters themselves have no meaning in the belief state. They are just labels used in the PDDL plan. However, causal links define predicates over these labels which *do* have a meaning in the belief state. For instance, `0 -> (robot pos1)` means that `pos1` refers to the position of the robot at time 0, which is the initial state. Therefore, the first parameters of the executable action `approachBall` are set to the initial position of the robot: x=0,y=1,ϕ=0,v=0. The result of this process is a (partially) instantiated sequence of executable actions.

```
(approachball pos1 pos2)
(dribbleball pos2 pos3)
     ⇒
approachBall(x=0,y=1,φ=0,v=0,xg=3,yg=1,φg=[-π,π],vg=[0,0.3])
dribbleBall(x=3,y=1,φ=[-π,π],v=[0,0.3], xg=1,yg=3,φg=2.6,vg=0)
```

Fig. 8. A partially instantiated action sequence.

Note that not all parameters are bound. For instance, although the initial position of the ball binds the x and y values related to `pos2`, it does not constrain the ϕ or v, as the ball's orientation and speed are not stored in the belief state. Therefore, the ϕ is free to choose between -π and π. The `atball` predicate further constrains the translational velocity to be in the range [0m/s,0.3m/s]. If several such predicates hold for an action parameter value, the possible ranges and values are composed. The partially unbound values in the executable action sequence are called *free action parameters*. In the next section, we describe how predictive knowledge can be used to optimize the values of these parameters.

IV. APPLYING ACTION MODELS: SUBGOAL REFINEMENT

In the previous section, declarative knowledge is used to generate abstract plans, and procedural knowledge is used to execute them. However, one of the remaining questions presented in the introduction cannot be answered by either kind of knowledge: 'What is the best angle of approach?' This is because the mapping from declarative to procedural knowledge is ambiguous, and free action parameters remain. Subgoal refinement is the process of choosing (refining) the values of the free action parameters at the subgoal, so that they are optimal with respect to the predicted performance. Here, this performance is execution duration, which is predicted with the action model presented in Section II-C.1.

Using redundant degrees of freedom to optimize subordinate criteria has been well studied in the context of arm control, both in humans [21], [26] and robots [23], [18]. Arm poses are said to be redundant if there are many arm configurations that result in an equivalent pose. In the work cited above, all these configurations are called uncontrolled manifold, motion space, or null space, and finding the best configuration is called redundancy resolution or null-space optimization. A difference between typical redundancy resolution techniques and our approach is that the former optimizes actions themselves, whereas we rather optimize transitions *b*etween actions.

[1]VHPOP can be downloaded free of cost at http://www.tempastic.org/vhpop/

In the running example from Figure 1, the free action parameter is the angle of approach. The three graphs in Figure 9 represent the execution duration of the first and second action, as well as their sum for all possible angles of approach. Subgoal refinement determines the free action parameter for which the overall performance is optimal. The most right graph in Figure 9 is essentially the search space for subgoal refinement. In this case, the lowest execution time of 6.5 seconds is achieved for an angle of 50°. Note that greedily optimizing the performance of only the first action (2.5s) leads to a lower overall performance (7.4) seconds.

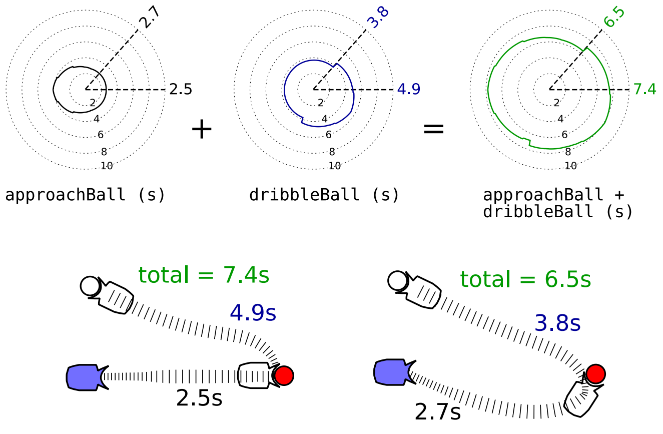


Fig. 9. Selecting the optimal subgoal by finding the optimum of the summation of all action models in the chain.

For this example, these minima can easily be read from the graph, as the search space has only one dimension. When applying subgoal refinement to real problems, search spaces of ten dimensions easily arise, and exhaustive search becomes intractable. Therefore, we use a genetic algorithm to determine the optimum. Each action parameter is encoded by one floating point in the chromosome, and the fitness function is simply the summation of the action models of the actions involved. Optimization time is usually small in comparison to the gain in performance. For the extreme scenario, where several actions with many free action parameters are optimized, our implementation of the genetic algorithm still takes less than 0.5s to get a good result.

A. Empirical Evaluation

In the soccer domain, subgoal refinement was evaluated both on the real robots and in simulation, by executing the scenario in Figure 1 for varying positions of the robot and the goal.

In the service robotics domain, two scenarios are tested. In the first scenario, the goal is to put a cup from one table to another. In each evaluation episode, the topology of the environment in each scenario stays the same, but the initial robot position, the tables, and the cups are randomly displaced. Scenario 2 is a variation of Scenario 1, in which two cups had to be delivered.

In the arm control domain, sequences of reaching movements are performed. Because this particular task does not require abstract planning, we did not use VHPOP. For demonstration purposes, we had the arm draw the first letter of the first name of each author of [25], and chose 4/5 waypoints accordingly. To draw these letters, only two of the six degrees of freedom of the arm are used. The free action parameters are the angular velocities at these waypoints.

Table III lists the results of applying subgoal refinement to the different domains and scenarios. n is the number of episodes tested and $\overline{t_g}$ and $\overline{t_s}$ are the mean execution times of the entire action sequence with the greedy approach (in which only the current action was optimized) and subgoal refinement (which optimizes the current *and* next action). The fifth column lists the mean improvement achieved with subgoal refinement. The p-value of the improvement was computed using a dependent t-test with repeated measures. A significant and substantial improvement occurs in all but one domain.

Scenario	n	$\overline{t_g}$	$\overline{t_s}$	$1 - \overline{t_s/t_g}$	p
Soccer (Simu.)	100	9.8s	9.1s	6.6%	0.00
Soccer (Real)	100	10.6s	9.9s	6.1%	0.00
Kitchen (Sc. 1)	100	46.5s	41.5s	10.0%	0.00
Kitchen (Sc. 2)	100	91.7s	85.4s	6.6%	0.00
Manipulation	4	10.6s	10.0s	5.7%	0.08

TABLE III
SUBGOAL REFINEMENT RESULTS

Figure 12 depicts the results from the manipulation domain. The angular velocities were set to zero (green, dashed) or optimized with subgoal refinement (blue, solid). The axes represent the angles of the two joints. This figure shows well qualitatively that the trajectories ares smoother with subgoal refinement: the arms often draws one long stroke, rather than discernible line segments. Since the manipulation domain was mainly included for visualization purposes, there are only a few episodes.

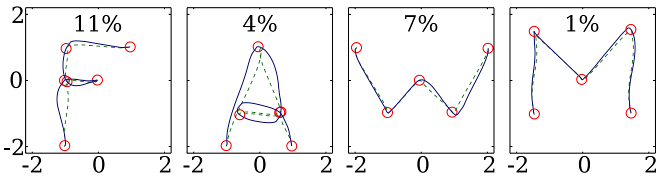


Fig. 12. Drawing letters with (blue, solid) and without (green dashed) subgoal refinement.

Although optimizing execution duration also leads to smoother motion in the manipulation domain, in humans it more likely arises from variability minimization [23]. Our main goal is not to explain or model human motion, but rather to optimize actions sequences.

V. APPLYING ACTION MODELS: SUBGOAL ASSERTION

In the previous section, the soccer robot reused and refined the `approachBall` action so that it performs well in the task context of first approaching and then dribbling the ball. In this section, we show that the `goToPose` action can be reused and refined so that it can be used in the context of approaching the ball. This is possible, because for both humans and robots, approaching a ball is very similar to

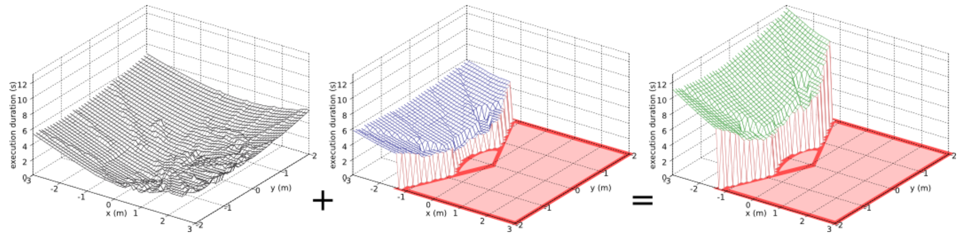


Fig. 10. Search space for subgoal refinement in subgoal assertion with two goToPose actions.

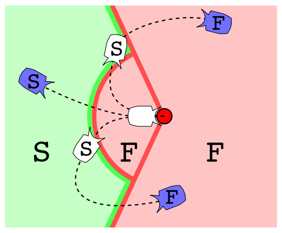


Fig. 11. Example assertions.

navigating without considering the ball. On an abstract level, both involve going from some state to another state on the field, and both should be implemented to execute as efficiently and fast as possible.

However, there are also slight differences between these two tasks. When approaching the ball it is important to not bump into it before achieving the desired state. In Section II-C.2, we demonstrated that this failure occurs approximately half the time. On the other hand, goToPose also *succeeds* at approaching the ball half the time, and can be reused without change in these cases.

The key to reuse is therefore being able to predict when an action will fail, and when it will succeed at a novel task. When it is predicted to succeed, the action is executed as is. If the action will fail, another action should be executed beforehand, such that the robot ends up in a state from which the action *will* succeed. As a novel subgoal is introduced, this approach is therefore called subgoal assertion.

When mapping the declarative action approachBall to the goToPose action(s), a call is first made to the failure prediction action model described in Section II-C.2. If a success is predicted, approachBall is instantiated with one goToPose action. If the goToPose action is predicted to fail, approachBall is instantiated with a sequence of two goToPose actions. Examples of both situations are depicted in Figure 11.

The intermediate subgoal between the two goToPose can be chosen anywhere in the green area (S) in Figure 6, as the execution of the second goToPose is predicted to succeed from this area. The free action parameters of this subgoal are optimized automatically with subgoal refinement. This ensures that the values for the intermediate parameters minimize the predicted execution duration, and that the transition between the two goToPose actions is smooth. Note that the intermediate goal between the actions must lie within the green area in Figure 6 to ensure that the execution of the second goToPose action will succeed. This requirement puts constraints on the values of the intermediate parameters. To ensure that subgoal refinement only considers states in the green area of Figure 6, the action model for the second action is modified so that it returns an INVALID flag for these states. This approach has been chosen as it requires little modification of the optimization module.

Analogously to Figure 9, the predicted execution durations of the two actions for varying x and y, as well as their summation are depicted in Figure 10. Invalid values are rendered in red on the ground plane. Note that due to removal of invalid values, the shape of the functions on the ground plane in the last two graphs corresponds to Figure 6 and 11.

In Figure 11, three instances of the problem are depicted. Since the robot to the left is in the area in which no collision is predicted, it simply executes goToPose, without asserting a subgoal. The model predicts that the other two robots will collide with the ball when executing goToPose, and a subgoal is asserted. The subgoals, determined by subgoal refinement, are depicted as well.

A. Empirical Evaluation

To evaluate automatic subgoal assertion, a hundred random ball approaches are executed in simulation, once with subgoal assertion, and once without. Without assertion, the results are similar to the results reported in Table II. A collision is again correctly predicted approximately half the time: 52% of these hundred episodes. Subgoal assertion is applied in these cases, and is almost always successful: the number of successful executions is raised from 47 to 97%.

VI. Related Work

Refining procedural knowledge instead of learning predictive knowledge. Optimizing action execution can also be done by tailoring the procedural knowledge itself to different task contexts, for instance by implementing novel actions. This is a laborious task, as each task context, and there are usually many, would require their own task-specific action. Many specific actions make action selection programming and planning more complex, and make the system less adaptive, less general and more difficult to maintain.

Reinforcement Learning (RL) is another method that seeks to optimize execution performance with respect to a reward function. Recent attempts to combat the curse of dimensionality in RL have turned to principled ways of exploiting temporal abstraction [1]. The only approach we know of that combines Reinforcement Learning with explicitly represented declarative knowledge to generate action chains is RL-TOPS (*Reinforcement Learning - Teleo Operators*) [20]. Values and action-value pairs for temporally abstracted RL policies can be considered action models. However, our action models are action-specific, not task-specific (rewards are given for finishing tasks, not actions), and have more more informative performance measures. This facilitates the reuse of action models for novel tasks.

Learning predictive knowledge from procedural knowledge. Related work on learning prediction models by observing the outcome of executing actions has enabled robots to predict: gripper poses with Bayesian networks [9], optimal navigation parameters with Dynamic Bayesian Networks [14], cost models for indoor navigation actions with regression [11] and model [4] trees. In these approaches, action models are used to improve the execution of isolated tasks on one type of robot. In our approach, action models are an integral and central part of the computational model, are acquired automatically, represented explicitly, and used as modular resources for different kinds of control problems. The generality of our approach is demonstrated by its use on three different robotic platforms.

Interaction of procedural and predictive knowledge. The Modular Selection And Identification for Control (MOSAIC) architecture [13] integrates action models into a computational model for motor control. This framework uses paired inverse and forward models to model two problems: how to learn inverse models for tasks, and how to select the appropriate inverse model, given a certain task. This architecture has not been designed for robot control.

Interaction of declarative and procedural knowledge. Other recent approaches to using symbolic planning for robots focus on different problems that arise when such plans are executed on real robots. For instance, by using probabilistic planners in the abstract planning domain [5]. aSyMov [7] reasons about geometric preconditions and consequences of actions in a simulated 3-D world. Note that our methods are not incompatible with [5], [7].

Hybrid motion planning approaches such as [6], compute and commit to plans offline, but leave freedom in the plans to react to unforeseen circumstances. Whereas these approaches consider the freedom *between* the subgoals, subgoal refinement considers the freedom *at the subgoal itself*.

In the XFRMLearn framework, human-specified declarative knowledge is combined with robot-learned knowledge [2]. Navigation plans are optimized with respect to execution time by analyzing, transforming and testing plans. XFRMLearn learns by testing plans, whereas we learn the analysis phase.

VII. Conclusion

To reason about their actions, cognitive systems must have models of their actions. In this paper, we present a system that reasons about models at two levels of abstraction. First of all, abstract declarative models, which are manually specified in PDDL, are used to select (chains of) actions. Declarative plans are mapped to actions using an action instantiation algorithm. The resulting action sequences are then optimized with action models, which predict the execution duration and success of an action, given its parameters. Since it is difficult to specify action models manually, the robots apply tree-based induction to learn them from experience, gathered by observing the execution of their own actions. Our empirical evaluation demonstrates that predictive knowledge enables robots to optimize their behavior autonomously in real-time.

We demonstrated how the robots learn accurate action models for actions with up to 8 parameters. For a higher number of dimensions, manually specified feature spaces and tree-based induction might not yield accurate models. Future work aims at using more abstract feature spaces and different learning algorithms. We also aim at learning other performance measures, such as energy consumption, and combining different action models to be able to optimize multi-criteria performance measures.

References

[1] A. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event systems*, 2003.

[2] M. Beetz and T. Belker. XFRMLearn - a system for learning structured reactive navigation plans. In *Proceedings of the 8th International Symposium on Intelligent Robotic Systems*, 2000.

[3] M. Beetz, T. Schmitt, R. Hanek, S. Buck, F. Stulp, D. Schröter, and B. Radig. The AGILO robot soccer team experience-based learning and probabilistic reasoning in autonomous robot control. *Autonomous Robots*, 17(1):55–77, 2004.

[4] T. Belker. *Plan Projection, Execution, and Learning for Mobile Robot Control*. PhD thesis, University of Bonn, 2004.

[5] A. Bouguerra and L. Karlsson. Symbolic probabilistic-conditional plans execution by a mobile robot. In *IJCAI-05 Workshop: Reasoning with Uncertainty in Robotics (RUR-05)*, 2005.

[6] O. Brock and O. Khatib. Elastic Strips: A framework for integrated planning and execution. In *Proceedings International Symposium on Experimental Robotics*, 1999.

[7] S. Cambon, F. Gravot, and R. Alami. A robot task planner that merges symbolic and geometric reasoning. In *ECAI*, 2004.

[8] S. Coradeschi and A. Saffiotti. Perceptual anchoring of symbols for action. In *IJCAI*, 2001.

[9] A. Dearden and Y. Demiris. Learning forward models for robotics. In *IJCAI*, 2005.

[10] M. Fox and D. Long. PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of AI Research*, 20:61–124, 2003.

[11] K. Haigh. *Situation-Dependent Learning for Interleaved Planning and Robot Execution*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1998.

[12] S. Hart, S. Ou, J. Sweeney, and R. Grupen. A framework for learning declarative structure. In *Workshop on Manipulation for Human Environments, Robotics: Science and Systems*, 2006.

[13] M. Haruno, D. Wolpert, and M. Kawato. MOSAIC model for sensorimotor learning and control. *Neural Computation*, 2001.

[14] G. Infantes, F. Ingrand, and M. Ghallab. Learning behaviors models for robot execution control. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI)*, 2006.

[15] R. Jacobs and M. Jordan. Learning piecewise control strategies in a modular neural network. *IEEE Transactions on Systems, Man and Cybernetics*, 23(3):337–345, 1993.

[16] M. Matarić. Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior. *Trends in Cognitive Science*, 2(3):82–87, 1998.

[17] B. Mehta and S. Schaal. Forward Models in Visuomotor Control. *Journal of Neurophysiology*, 2002.

[18] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal. Comparative experiments on task space control with redundancy resolution. In *IEEE International Conference on Intelligent Robots and Systems*, 2005.

[19] R. Quinlan. Learning with continuous classes. In A. Adams and L. Sterling, editors, *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.

[20] M. Ryan and M. Pendrith. RL-TOPs: an architecture for modularity and re-use in reinforcement learning. In *Proc. 15th International Conf. on Machine Learning*, 1998.

[21] S. Schaal and N. Schweighofer. Computational motor control in humans and robots. *Current Opinion in Neurobiology*, 2005.

[22] W. Scoville and B. Milner. Loss of recent memory after bilateral hippocampal lesions. *Journal of Neurology, Neurosurgery and Psychiatry*, 20:11–21, 1957.

[23] G. Simmons and Y. Demiris. Biologically inspired optimal robot arm control with signal-dependent noise. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, 2004.

[24] F. Stulp. *Tailoring Robot Actions to Task Contexts using Action Models*. PhD thesis, Technische Universität München, 2007.

[25] F. Stulp, W. Koska, A. Maldonado, and M. Beetz. Seamless execution of action sequences. In *ICRA*, 2007.

[26] D. Wolpert and Z. Ghahramani. Computational principles of movement neuroscience. *Nature Neuroscience Supplement*, 3, 2000.

[27] H. Younes and R. Simmons. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20:405–430, 2003.