Seamless Execution of Action Sequences

Freek Stulp, Wolfram Koska, Alexis Maldonado and Michael Beetz
Intelligent Autonomous Systems Group, Technische Universität München
Boltzmannstrasse 3, D-85747 Garching bei Munich, Germany
stulp@cs.tum.edu

***Abstract*— One of the most notable and recognizable features of robot motion is the abrupt transitions between actions in action sequences. In contrast, humans and animals perform sequences of actions efficiently, and with seamless transitions between subsequent actions. This smoothness is not a goal in itself, but a side-effect of the evolutionary optimization of other performance measures.**

In this paper, we argue that such jagged motion is an inevitable consequence of the way human designers and planners reason about abstract actions. We then present subgoal refinement, a procedure that optimizes action sequences. Subgoal refinement determines action parameters that are not relevant to why the action was selected, and optimizes these parameters with respect to expected execution performance. This performance is computed using action models, which are learned from observed experience. We integrate subgoal refinement in an existing planning system, and demonstrate how requiring optimal performance causes smooth motion in three robotic domains.

I. Introduction

When it comes to elegant motion, robots do not have a good reputation. Jagged movements are actually so typical of robots that people trying to imitate robots will do so by executing movements with abrupt transitions between them. In contrast, one of the impressive capabilities of animals and humans is their capability to perform sequences of actions efficiently, and with seamless transitions between subsequent actions. In nature, fluency of motion is not a goal in itself, but rather an emergent property of time, energy and accuracy requirements. In this paper, we demonstrate that requiring optimal execution of action sequences also automatically leads to smooth motion in robots.

Let us illustrate the problem with a typical scenario from robotic soccer taken from [12], depicted in Figure 1. The robot's goal is to be in possession of the ball in front of the goal, which can be achieved by the action sequence 1) go to the ball; 2) dribble the ball in front of the goal. If the robot naively executed the first action, it might arrive at the ball with the goal at its back, as depicted in Figure 1a). 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 1b, 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 1b exhibits seamless transitions between actions and has higher performance, achieving the ultimate goal in less time.

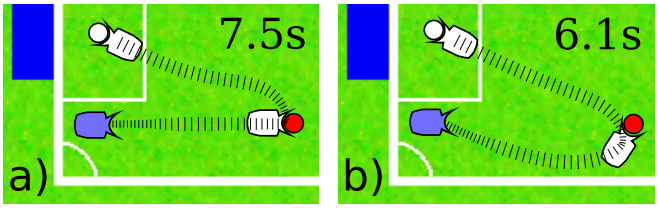

Fig. 1. Alternative executions of the same action sequence. Execution b) is faster and has seamless transitions between the actions.

In this paper, we will argue that jagged motion is a fundamental problem that inevitably arises from the way robot controllers and actions are designed and reasoned about. We then present *subgoal refinement*, which deals with this problem by reasoning about how action execution can be tailored to the task context [12]. To be able to predict the performance of an action before it is actually executed, the robots learn action models from observed experience. Subgoal refinement optimizes action sequences with respect to the predicted performance, with fluent motion as a side-effect.

The contributions of this paper are 1) analyzing why abrupt robotic motion arises, 2) integrating subgoal refinement in an existing planning system, and 3) demonstrating how optimality requirements lead to smooth motion in three robotic domains: service robotics, robotic soccer and manipulation. We believe these contributions help bridge the gap between classical planning and applied robotics.

The rest of this paper will proceed as follows. The next section will have a closer look at why abrupt transitions arise, and informally introduce the solution idea, subgoal refinement. Section III explains how sequences of actions are generated and optimized using learned action models. We evaluate the application of subgoal refinement in three robotic domains in Section V, after which we present related work in Section VI. We conclude with Section VII.

II. Problem analysis and solution idea

In this section we will explain the key issues that cause jagged motion, as well as the rationale behind our solution. These issues are related to the different modules of the system implementation, which we present in Section III. References to the corresponding subsections have been made.

1) Parameterizable actions enable abstract action selection. A successful approach to expressing high-level goals in terms of low-level motor commands is to specify a set of temporally extended parameterizable actions, also known as behaviors, skills or controllers. Complex behaviour is then achieved by generating sequences of parameterized actions, as in Figure 1. Planning, behaviour-based, hierarchical reinforcement learning, and most hand-coded controllers use actions to facilitate controller learning or design.

Because actions are designed with a certain goal in mind, they can be selected based on *what* they do, thereby abstracting away from *how* they do it. For instance, the name of the action `approachBall` alone already gives a clear indication of what it is intended to do, although it is unknown, and for high-level action selection purposes irrelevant, how it actually achieves what its name indicates. In Section III-A, we describe a planning system that automatically selects parameterizable actions based on their abstractions.

2) Action conditions bind some action parameters, but not all. *What* an action can achieve is usually referred to as its *effects* or *post-conditions*, and *when* it can be executed as its *pre-conditions*. These conditions are either only implicitly known to the designer of the action, or explicitly encoded in the controller, as in planning. Programming action selection modules in terms of such modular, declarative components is widely held to be easier than in a direct procedural account.

Conditions are abstractions, as they usually disregard many aspects of the situations before, during, and after action execution. This means that they often abstract away from some of the action's parameters. How to determine which action parameters are bound by the conditions and which are not will be discussed in Section III-B.

The disadvantage of abstracting away from action parameters is that these parameters might influence the performance or robustness of actually executing the action. This became clear in the example in Figure 1. From the point of view of the abstract conditions, being at the ball is sufficient for dribbling it. Although the angle of approach might not be relevant on an abstract level, the example clearly shows that it does influence execution performance. Instead of requiring the user to refine conditions to include these considerations, the robot should do this autonomously.

3) Free action parameters can be optimized with respect to expected performance. Parameters that are not bound by the conditions of an action are called *free action parameters*. For instance, in the running example, the free action parameter is the angle of approach. The position and angle of the robot with respect to the ball are not, as they are bound by the precondition of the `dribbleBall` action. Changing these values could mean the condition no longer holds, and the action cannot be executed correctly. On the other hand, the angle of approach is free, and *any* angle of approach would lead to a successful execution of the action sequence. Instead of choosing just any angle, our system automatically computes the value that optimizes the overall expected performance of the action sequence, as in Figure 1. Because one optimal intermediate goal is chosen from many, this procedure is called *subgoal refinement*. The implementation of this optimization will be discussed in Section III-D.

4) Action models can be learned. Subgoal refinement requires the expected performance of the overall action sequence to be predicted. Therefore, robots acquire models of their actions, which enable them to predict the performance of the action, given a certain parameterization. Before task execution, these models are learned from observed experience. First, each action is executed for a multitude of parameterizations and the performance recorded. A learning algorithm then learns a generalized model that maps an action and its parameterization to expected performance.

The benefit of this approach over analytical methods is that it is based on real experience, so it takes all factors relevant to performance into account. Also, many hand-coded actions are difficult to formalize, or analysis is impossible because the inner workings of the action are unknown. In principle, learning models can also be done on-line, so that action models can adapt to changing environments [5].

III. Implementing Subgoal Refinement

In this section, we describe how subgoal refinement was implemented, and used to optimize action sequences generated by the planning system VHPOP.

A. Generating Sequences of Abstract Actions

Figure 2 depicts the computational model of action sequence generation. It is similar to those of other approaches that merge symbolic planning with robotic action execution [3], [4]. In the abstract planning domain, a planner generates sequences of abstract actions based on an abstract state and a goal. To do so, it uses the pre- and post-conditions of the actions in the action library. The abstract state is acquired by performing an abstraction on the robot's belief state, called anchoring. The sequence of actions is acquired by replacing the abstract actions in the plan with their corresponding executable actions from the library, and replacing abstract conditions with belief state values.

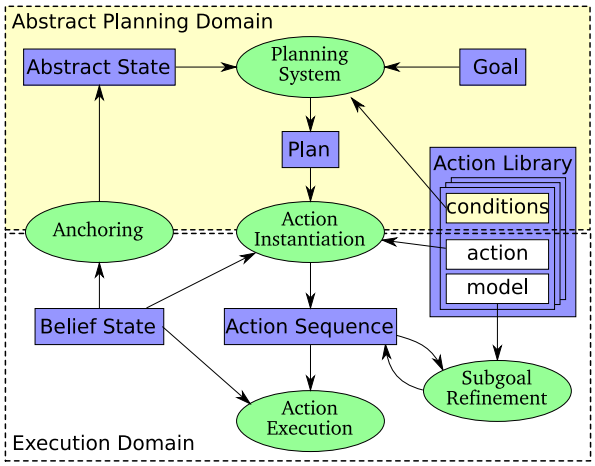

Fig. 2. Computational model of planning and subgoal refinement.

We use the Planning Domain Description Language (PDDL2.1 [6]) to describe abstract actions, abstract states and goals. Converting the continuous variables from the belief state into named symbols (e.g. PDDL symbols) is called anchoring [3]. As we currently do not consider re-planning, anchoring need only take place at the beginning of the planning process. Therefore, a complicated anchoring process with object tracking was omitted.

Because the actual planning process is not the focus of our research, we use VHPOP [15] to generate our abstract plans. This partial order causal link planner was used as is: no adaptations are needed to apply subgoal refinement to the plans it generates. We are aware that VHPOP is far from state-of-the-art with respect to robot planning, and that our assumptions about robot planning might seem naive. We have chosen VHPOP because its high abstraction level allows us to focus on the important aspects of action abstraction, instead of the many intricacies of robot planning. Our methods are certainly not incompatible with more complex robot planning systems [3], [4].

PDDL plans are very abstract, with clear semantics of *what* actions do, even without knowing how the actions are executed. This makes human inspection of the plan feasible. However, it does not specify *how* this plan can or should be executed in the real world. Therefore, once the plan is acquired, the abstract action sequences must be instantiated with the corresponding executable actions. This requires information from the belief state. For instance, the operator `(goto door1 door2)` is converted to an action by extracting the `door1` and `door2` into the robot's coordinate system using the mapping from the anchoring process, obtaining actual coordinates. The `gotoPose` action can thus be instantiated, parameterized, and then executed.

```
Abstract Operator Chain
1: (goto door1 door2)

Executable Action Sequence
1: gotoPose(
     x=2.68, y=2.12, phi=1.78, vtra=0.00,
     sxg=4.17, yg=3.40, phig=0.00, vtrag=0.00
)
```

B. Free action parameters

In the previous section, we saw an example of an instantiated action that was fully parameterized. Since conditions often abstract away from action parameters, the resulting actions are actually often only partially parameterized. Unspecified parameters are usually set to default values, such as `phig` was set to 0.0 in the example. This value is arbitrary, and in principle any value from the range of possible values, defined by the action's postcondition (to be at `door2`), could be used.

To determine which parameters are free, and which ranges can be used for them, our system considers the context in which the actions are executed, their pre- and post-conditions, as well as the values in the belief state. For instance, the maximum velocity of our soccer robots is 2m/s, but the precondition of `dribbleBall` specifies that it should be in the range [0m/s,0.3m/s] because the robot will not be able to control the ball otherwise. So the goal velocity of a `dribbleBall` action will always be in this range, and the starting velocity should be in this range, too. To achieve this, the prior action which affects this parameter has to know the context in which it is executed. In the example below, `approachBall` should be executed to achieve `(at ballpos)` for `dribbleBall`. In this case, the robot's velocity at the end of the `approachBall` action should be slow, as the precondition of `dribbleBall` specifies. In contrast, if achieving `(at ballpos)` for `kickBall`, the velocity should be high. This contextual information is contained in the causal links of the partial order planner.

Below an example of a partially specified action sequence, corresponding to Figure 1 is listed. Note that each action parameter has an ID number, to keep track of equivalent values in different actions, which is important for the optimization process.

```
Abstract Action Chain
1: (approachBall startpose ballpos)
2: (dribbleBall ballpos goalpose)

Executable Action Sequence
1: approachBall(
     1:x=0, 2:y=1, 3:phi=0, 4:vtra=0,
     11:xg=3, 12:yg=1,
     13:phig=[-PI,PI], 14:vtrag=[0,0.3]
 )
2: dribbleBall(
     11:x=3, 12:y=1,
     13:phi=[-PI,PI], 14:vtra=[0,0.3],
     21:xg=1, 22:yg=3, 23:phig=2.6, 24:vtra=0
 )
```

C. Learning Action Models

To optimize free action parameters with respect to the expected performance of all actions in the action sequence, the robot must be able to predict the performance of each action in the sequence. Before task execution, the robot therefore learns *action models* from observed experience. These models predict an action's performance given a specific action parameterization. In this paper, the performance measure is time. The difference with forward models [5] is that action models predict the outcome of temporally extended actions, not single motor commands.

The first step is acquiring training data for each action. This is done by simply executing the action for varying parameterizations, which is also known as *motor babbling*. For instance, navigation actions are executed for random poses in the area where the robot can go. 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 V. Before learning, the data is transformed to a feature space appropriate for the action.

Then, model trees are trained with this data to acquire a general model of the action. Model trees are functions

that map continuous or nominal features to a continuous value. The function is learned from examples, by a piecewise partitioning of the feature space. A linear function is fitted to the data in each partition. This linear function interpolates between data in the partition. Model trees are a generalization of decision trees, in which the nominal values at the leaf nodes are replaced by line segments. For more information on model trees, we refer to [10] and [13]. In our implementation, we use WEKA [14] to learn these model trees, and convert its output to an executable C++ function.

D. Free Action Parameter Optimization

Figure 3 depicts the optimization search space of the running example from Figure 1. The free action parameter is the angle of approach. The three graphs represent the execution duration of the first and second action, as well as their sum. Subgoal refinement determines the free action parameter for which the overall performance is optimal. In this case, the lowest execution time of 6.1 seconds is achieved for an angle of 59°. Note that greedily optimizing the performance of only the first action (2.1s) leads to a lower overall performance (7.5) seconds.

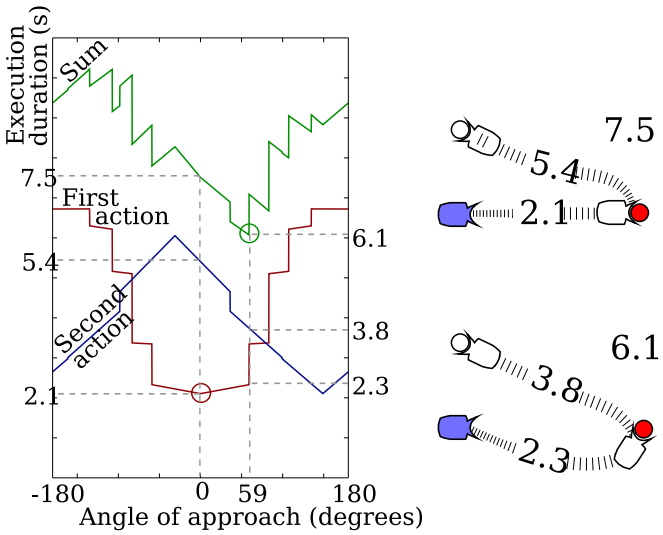

Fig. 3. The predicted performance for varying angles of approach. The scenarios to the right represent the minimum of only the first action, or the sum of both actions.

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.

IV. Application Domains and Scenarios

In this section, we will present the three application domains, and the scenarios and action sequences to which subgoal refinement is applied. The accuracy of the learned action models and the effect of applying subgoal refinement to these action sequences will be presented in Section V. As action model learning and subgoal refinement are independent of specific action implementations, we do not describe them here.

A. Robotic Soccer Domain

In this domain we have used a simulated and real Pioneer I robot of our RoboCup mid-size team the "AGILO RoboCuppers" [2], one of which is depicted in Figure 4a). The optimized action sequence in this domain is the `approachBall` action, followed by a `dribbleBall` action, as in Figure 1. The action models for both these actions were learned by executing them for random locations on the field, as reported in [13]. The free action parameters at the intermediate state are the angle of approach and the translational velocity.

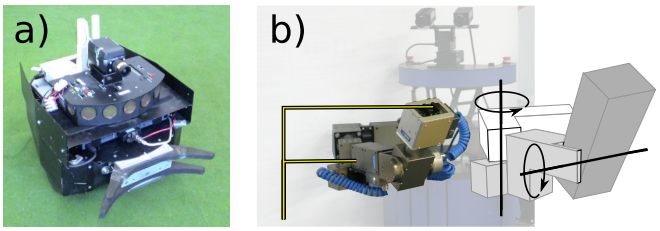

Fig. 4. a) The Pioneer robot used in the robotic soccer domain. b) The PowerCube arm used in the manipulation domain.

B. Manipulation Domain

In this domain, an articulated robot performs reaching movements. Strictly speaking, we do not manipulate any objects yet, but we will continue our research in this direction. The experiments were performed with a Powercube arm from Amtec Robotics with 6 degrees of freedom mounted on a B21 Robot, shown in Figure 4b). Each joint has a brushless servo motor with a Harmonic gear head, and an incremental optical encoder to measure the position. For the experiments, only two joints were used.

Data were gathered by generating random way-points in joint-space and angular velocities at these way-points. In this domain, we have used only one action, that takes the arm from one way-point to the next using a ramp velocity-profile. A PID controller sends power commands to the joints to allows fine control of the action, which is parameterized by initial position, initial velocity, cruise velocity, final position, final velocity, and acceleration.

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, and chose the way-points accordingly. The free action parameters are the angular velocities at these way-points.

C. Service Robotics Domain

The experiments in this domain were carried out in a simulated kitchen environment, depicted in Figure 5. The simulation was implemented with the "Player" and "Gazebo" modules of the Player/Stage project [7]. "Player" provides a network interface to the "Gazebo" 3D robot simulation, which is a realistic simulation of the robot, its sensors and the environment. It also provides a rendered view for the user. We used a B21 robot model, which was modified to have two articulated arms, as shown in Figure 5. It is a model of the a real B21 in Section IV-B.

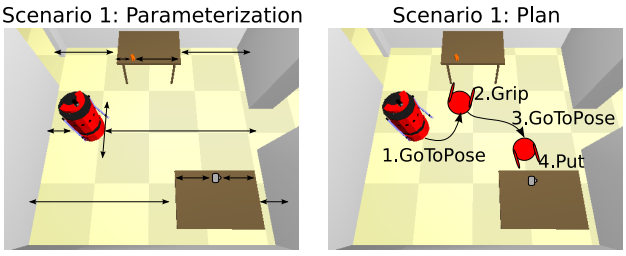

Fig. 5. Scenario 1. In each episode, the objects and the initial robot position are different. Possible positions are indicated by arrows.

The set of actions in this domain are: `goToPose`, `grip`, `put`. Again, action models were learned from data acquired by executing these actions with varying parameterizations.

To evaluate the effect of subgoal refinement, two different scenarios were tested. In the first scenario, the goal is to put a cup from one table to the other, which can be achieved by the action sequence depicted in Figure 5. In each episode in the evaluation, the topology of the environment in each scenario stays the same, but the initial robot position, the tables and the cups are randomly displaced along the arrows in Figure 5. Scenario 2 was a variation of Scenario 1, in which two cups had to be delivered.

These scenarios have many free action parameters. Because pre-conditions usually fix either navigation *or* manipulation motions but never both (they are independent), one of these action parameter sets is always free. Furthermore, the distance the robot must have to the table in order to grab a cup must be between 40 and 80cm (as fixed in the pre-condition of `grip`). This range is another free parameter. As in the soccer domain, the velocity and orientation at waypoints are also not fixed, so free for optimization as well.

V. Empirical Evaluation

In the evaluation, we first investigate the accuracy of the action models. Table I lists the number of episodes (n) executed to gather data for the training set, the data gathering duration (in real time), as well as the model's Mean Absolute Error (MAE) on a separate test set, with $n/3$ episodes. Note that the action models used in subgoal refinement were trained with the data in both training and test set. Although the results are good enough for subgoal refinement, we believe that more accurate models can be learned by using data gathered on-line during robot operation, as more data can be collected. Furthermore, models will only have to generalize over action parameterizations that actually arise during actual robot operation.

Robot	Action	n	Duration	MAE
Pioneer I (Real)	`goToPose`	290	0:31	0.31s
	`dribbleBall`	202	0:26	0.43s
Pioneer I (Simulated)	`goToPose`	750	1:18	0.21s
	`dribbleBall`	750	1:32	0.29s
B21	`goToPose`	2300	5:45	0.49s
	`reach`	2300	1:38	0.09s
PowerCube	`reach`	1100	0:53	0.21s

TABLE I
Action Model Accuracy

Table II 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 II
Subgoal Refinement Results

Figure 6 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. Although the letters in the figure are drawn in two-dimensional plane, the robot actually draws the letters on a virtual sphere. 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. For this reason the overall improvement is not significant (>0.05).

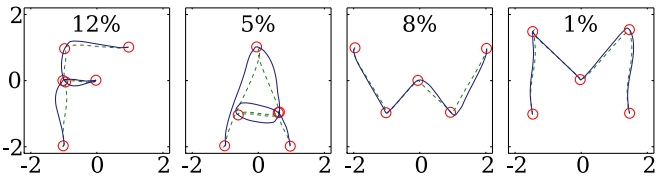

Fig. 6. Drawing letters with (blue, solid) and without (green dashed) subgoal refinement.

VI. Related Work

One way to solve suboptimal execution of action sequences is by implementing or learning many novel actions, one for each action sequence context. This is a laborious task, as each task context, and there are usually many, would require their own task-specific action. The small differences between these actions must also be reflected in their conditions, because the correct action for the context could otherwise not be selected. 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 performance, specified by a reward function. Recent attempts to combat the curse of dimensionality in RL have turned to principled ways of exploiting temporal abstraction [1]. In our view, the benefits of our methods are that they acquire more informative performance measures, facilitate the reuse of action models, and scale better to continuous and complex state spaces [12].

Most similar to our work, from the point of view of smoothness as an emergent property of optimality requirements, is described in [9]. Here a simulated robot maps its environment with range measurements by traversing a set of way-points. Reinforcement learns a policy that minimizes the error in the resulting map. As a side-effect, smooth transitions at way-points arise. This approach has not been tested on real robots.

Although optimizing execution duration leads to smoother motion in the manipulation domain, in humans it more likely arises from variability minimization [11]. In this paper, our main goal is not to explain or model human motion, but rather to demonstrate the effects of optimizing sequences of actions.

Overexpressiveness of actions has been well studied in the context of arm control. In redundant manipulation systems, different arm configurations can achieve the same task. This set of configurations is called motion- or null space. In [8], arm-specific search methods are used to find the configuration with the best fault tolerance in motion space. Our approach is more general, and can be applied to many robotic domains.

In AI action planning [6], actions in plans are often fully parameterized, because there is no difference between an operator's abstraction and its execution. Therefore, the disadvantages and advantages of free action parameters do not arise. In contrast to most planners, our system generates action sequences that have been optimized with respect to very realistic, non-linear, continuous performance models, which are grounded in the real world as they are learned from observed experience. We are not aware of other planning systems that generate abstract plans and simultaneously optimize the actual physical behaviour of robots.

VII. Conclusion

We believe our approach is an important contribution towards bridging the gap between robot action execution on the one hand, and planning systems and deliberative components in general on the other. As robots are becoming more dexterous, and their actions more expressive, abstraction will become more important for keeping action selection tractable. This also means the gap between an action's abstraction and its execution will widen, and more free action parameters will arise. Suboptimal performance and jagged motion is an unavoidable consequence of leaving these free action parameters unconsidered. Subgoal refinement not only contemplates free action parameters, but exploits them by optimizing them with respect to the expected overall performance, thereby turning the curse of free action parameters into a blessing. An interesting and pleasing side-effect is that there are no abrupt transitions, and smooth natural motion arises.

Subgoal refinement is part of a larger research project to use action models to make robots more aware of their actions, also enabling them for instance to coordinate actions [13]. Our future work aims at learning other performance measures, such as energy consumption, and combining them to optimize multi-criteria performance measures.

Acknowledgments

The work in this paper was partially funded by the Deutsche Forschungsgemeinschaft in the SPP-1125 "Cooperating Teams of Mobile Robots in Dynamic Environments".

References

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

[2] 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.

[3] A. Bouguerra and L. Karlsson. Symbolic probabilistic-conditional plans execution by a mobile robot. In *IJCAI05 Workshop: Reasoning with Uncertainty in Robotics (RUR-05)*, 2005.

[4] S. Cambon, F. Gravot, and R. Alami. A robot task planner that merges symbolic and geometric reasoning. In *ECAI*, pages 895–899, 2004.

[5] A. Dearden and Y. Demiris. Learning forward models for robotics. In *IJCAI*, pages 1440–1445, 2005.

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

[7] B. Gerkey, R.T. Vaughan, and A. Howard. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *ICAR*, pages 317–323, 2003.

[8] R. Hooper. *Multicriteria Inverse Kinematics for General Serial Robots*. PhD thesis, University of Texas, 1994.

[9] T. Kollar and N. Roy. Using reinforcement learning to improve exploration trajectories for error minimization. In *ICRA*, 2006.

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

[11] G. Simmons and Y. Demiris. Biologically inspired optimal robot arm control with signal-dependent noise. In *Proc. of IEEE International Conference on Intelligent Robots and Systems*, pages 491–496, 2004.

[12] F. Stulp and M. Beetz. Optimized execution of action chains using learned performance models of abstract actions. In *IJCAI*, 2005.

[13] F. Stulp, M. Isik, and M. Beetz. Implicit coordination in robotic teams using learned prediction models. In *ICRA*, 2006.

[14] I.H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2 edition, 2005.

[15] H.L.S. Younes and Reid G. Simmons. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20:405–430, 2003.