Sensorimotor Abstraction Selection for Efficient, Autonomous Robot Skill Acquisition

George Konidaris, Andrew Barto
Autonomous Learning Laboratory, Department of Computer Science, University of Massachusetts Amherst
Email: {gdk, barto}@cs.umass.edu

***Abstract*—To achieve truly autonomous robot skill acquisition, a robot can use neither a single large general state space (because learning is not feasible), nor a small problem-specific state space (because it is not general). We propose that instead a robot should have a set of sensorimotor abstractions that can be considered small candidate state spaces, and select one that is appropriate for learning a skill when it decides to do so. We introduce an incremental algorithm that selects a state space in which to learn a skill from among a set of potential spaces given a successful sample trajectory. The algorithm returns a policy fitting that trajectory in the new state space so that learning does not have to begin from scratch. We demonstrate that the algorithm selects an appropriate space for a sequence of demonstration skills on a physically realistic simulated mobile robot, and that the resulting initial policies closely match the sample trajectory.**

I. Introduction

One of the most impressive features of human intelligence is skill acquisition—the ability to create new skills, refine them through practice, and apply them in new task contexts. This ability to retain and refine solutions to important subproblems and then employ them later lies at the heart of two important aspects of human intelligence: the ability to perpetually improve at difficult control tasks through practice, and the ability to solve increasingly difficult problems.

The design and coordination of independent specialized skill units (often called behaviors) is fundamental to modern robotics [1]. However, a robot that must act in a complex environment over an extended period of time should do more than just use existing skills: it should learn new skills that increase its capabilities and facilitate later problem solving. Although robots exist that can learn a skill given a reward function and hand-engineered state space, none exist today that display truly autonomous skill acquisition.

Reinforcement learning [2] is a natural fit for the robot skill acquisition problem, and with the recent development of the options framework [3] provides a principled approach to hierarchical learning and planning. However, most useful robots have multiple sensors and actuators from which very many features can be extracted. Real-time learning is extremely difficult in high-dimensional spaces, only becoming feasible when the learning problem is posed in the right space—one in which a small number of relevant (often heavily preprocessed) features are present, and a large number of irrelevant features are not. Because of this, successful robot learning research has made use of carefully designed problem-specific state spaces, which are not appropriate for truly autonomous skill acquisition. This poses an important question: when a robot decides to learn a new skill, *which state space should it use?*

We propose that instead of having either a single very large but general state space, or a single small but problem-specific state space, an autonomous robot should be equipped with a *set* of useful state spaces (in the form of sensorimotor abstractions) from which it can *select* when it decides to learn a new skill. We introduce an incremental algorithm that selects an appropriate state space for a skill given a successful sample trajectory. The algorithm returns a policy fitted to the sample trajectory in the selected space, avoiding the need to start learning from scratch. We use a physically realistic simulated mobile robot to show that the algorithm selects appropriate state spaces for a sequence of skills along a sample trajectory, and that the resulting initial policies replicate the sample trajectory in a form suitable for further learning.

II. Background

A. The Options Framework

The options framework adds methods for learning and planning using temporally extended actions, or *options*, to the standard reinforcement learning framework [2]. An option, o, is a control unit consisting of three components: an *option policy*, π_o, mapping state-action pairs over which the option is defined to execution probabilities; an *initiation set* indicator function, I_o, which is 1 for states where the option can be executed and 0 elsewhere; and a *termination condition*, β_o, giving the probability of the option terminating at each state in which it is defined [3]. The option policy may be defined in its own state space, S_o, which is usually a subset of the space in which the primary problem is posed.

An option is an appropriate model for a robot skill unit because it captures the essential elements required for control. The option framework provides methods for learning and planning using options as temporally extended actions, but it also provides a framework for learning new options. A system for learning new skills as options must include methods for determining when to create an option or expand its initiation set, how to define its termination condition, and how to learn its policy. In this paper we assume an existing mechanism for determining when an option should be created and determining a reward function, R_o, that specifies the goal of the option. The option policy can be learned using existing reinforcement learning methods; for robotics these are most likely to be policy gradient [4], [5] or actor-critic methods [2], in which

the policy is represented explicitly and an approximate value function provides a gradient used to improve it.

B. Setting

To study autonomous skill acquisition—in the sense of the *robot* deciding what skills to acquire, rather than the designer—we must frame the skill acquisition problem in the context of an autonomous robot control architecture. For this we adapt the setting commonly used in hierarchical reinforcement learning to the robot case, as follows:

1) *The robot has an environmental context.* The robot is in the process of solving some task set in an environment that is unknown, but that is drawn from a class of environments whose shared properties are known. For example, we may know that the robot must operate in office buildings, but not the layout of any particular office building it may encounter.
2) *The robot has a task context.* It is in the process of solving some task, is equipped with a set of controllers—e.g., a control basis [6]—sufficient to (possibly inefficiently or inconsistently) solve it, and uses a learning or planning algorithm to sequence them to do so.
3) *The robot has a control context.* While solving the task, the robot decides that something that it has just done should be isolated and learned as a new skill.

Skill acquisition serves two purposes: making a subgoal prominent (by making it reachable using a single decision) in planning or learning, and making skill execution more efficient or consistent over time.

This setting has two important implications not commonly present in the robot learning literature. First, the robot is already capable of sequencing its existing controllers to achieve the skill's goal. In particular, it has a successful trajectory sample that it can use to decide how to learn that skill, and which can be used to initialize the skill policy to avoid learning from scratch [7]. A similar successful trajectory (or set of trajectories) could also be obtained by demonstration [8] or teleoperation. Second, the robot may choose the range of the learned skill. It may even choose to learn *multiple* skills to achieve a given subgoal rather than a single monolithic skill, learning each component skill in an appropriate space, and retaining them for independent use elsewhere.

This setting is unusual for robot learning research. More typically, a robot is given a starting point for learning, a problem-specific state space, a reward function and no other knowledge. The task of learning is to first discover a solution (which often takes a great deal of time), and then to refine it. We argue that very little human learning happens that way: humans first discover an objective using their existing skill repertoire and *then* create a skill to refine the initial solution, using the ability to achieve it as a starting point for learning.

III. Sensorimotor Abstraction Selection

Given a robot, we may in principle attempt to use reinforcement learning techniques to learn a skill directly in its sensor and motor spaces. However, for most interesting robots learning this way is infeasible since both spaces will have very high dimension. Fortunately, most of the skills we would want robots to learn do not (at least initially) require all of the robot's sensor features and actuators. Instead, they can be solved using only a small set of high level features relevant for the task—for example, balancing skills require only measures of velocity, acceleration and horizon, rather than (for example) the entire output of a high-resolution video camera. Following Huber and Grupen [6], we model these *sensorimotor abstractions* as couplings of a set of input features and a set of control variables.

A. Sensorimotor Abstractions as State Spaces

Sensorimotor abstraction i consists of two components. A sensor abstraction program, σ_i, maps robot sensor state (possibly including state history) to some small set of features, s_i. A motor abstraction program, τ_i, maps the full robot actuator space to a motor feature space, m_i, and executes commands in motor command space, a_i (for example, m_i could be a set of joint angles and torques, and a_i could be a set of motor increment and decrement commands). Note that σ_i and τ_i may involve significant high-level processing—for example σ_i may perform high-level visual feature extraction.

When used as a sensorimotor abstraction, σ_i and τ_i can be considered features that define a state-action space (S^i, A^i), where the state space S^i includes the values of all of the variables in s_i and m_i, and A^i is the action space defined using the variables in a_i as features. Given some reward function we can thus learn a policy $\pi : S^i \rightarrow A^i$, using a suitable reinforcement learning algorithm. This is depicted in Figure 1.

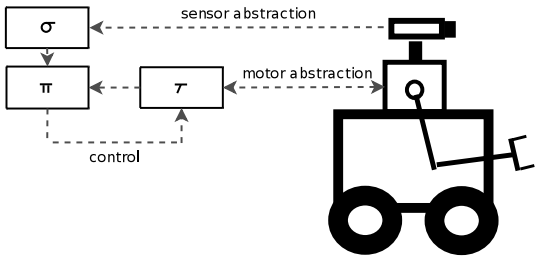


Fig. 1. A control policy using a sensorimotor coupling.

A robot with a given set of sensorimotor abstractions $\{(\sigma_1, \tau_1), \ldots, (\sigma_n, \tau_n)\}$ thus has a corresponding set of n state-action spaces $\{(S^1, A^1), \ldots, (S^n, A^n)\}$ from which it may choose to use to learn a new skill.

This formulation is very general, and covers more than just sets of sensory features combined with low-level control. For example, any parameterized policy can be considered a sensorimotor coupling. Similarly, each combination of assignments to a set of deictic pointers forms a sensorimotor coupling.

B. Selecting a Space

We assume that we are given a sample successful skill trajectory of t steps, represented by a sequence of t state-action pairs and associated rewards (typically energy cost):

$\{(\mathbf{s}_1, a_1, r_1), (\mathbf{s}_2, a_2, r_2), \ldots, (\mathbf{s}_t, a_t, r_t)\}$. Any sample trajectory that achieves the goal of the skill should be sufficient, although the closer to optimal the better. In addition, we can run each sensor and motor abstraction program during the sample trajectory and obtain a sample trajectory for each sensorimotor coupling: $\{(\mathbf{s}_1^i, a_1^i, r_1), (\mathbf{s}_2^i, a_2^i, r_2), \ldots, (\mathbf{s}_t^i, a_t^i, r_t)\}$, where $(\mathbf{s}_m^i, a_m^i, r_m)$ is a state-action-reward tuple obtained from sensorimotor abstraction i describing the mth state-action pair in the trajectory.

We seek the sensorimotor abstraction best able to represent a policy corresponding to the sample trajectory. Since the appropriate sensorimotor abstraction will likely support a good policy only locally—it may be insufficient to approximate a policy for the entire trajectory—we emphasise later samples (which are closer to the goal). This corresponds to finding abstraction i that can represent policy π_i mapping S^i onto A^i with lowest weighted error.

Since we assume a policy gradient style algorithm, we must fit both a value function and the trajectory (and thus policy) directly (a value-function-based algorithm would omit the policy fit). We cover the value-function fit in detail since the policy fit case is simply a weighted least squares fit.

Since we are given reward and assuming a linear function approximation scheme for each sensorimotor abstraction, fitting a value function amounts to performing a linear regression using return as the dependent variable and the basis functions from each S^i as independent variables. We select the abstraction with the least mean squared weighted error over return.

Following Boyan [9] we can accomplish this incrementally. We wish to minimize error term e_i:

$$e_i = \sum_{j=1}^{t} \rho^{(t-j)} [\mathbf{w} \cdot \Phi_i(\mathbf{s}_j) - R_j]^2, \tag{1}$$

where Φ_i denotes the basis functions used by abstraction i, $\mathbf{w}$ denotes the function approximation parameter vector, R_j denotes return (summed discounted rewards) from step j and ρ denotes the weighting factor ($0 < \rho \leq 1$, settings closer to 0 assign more recent returns higher weights). Setting the derivative with respect to $\mathbf{w}$ of Equation 1 to zero implies:

$$\sum_{j=1}^{t} \rho^{(t-j)} \mathbf{w} \Phi_i(\mathbf{s}_j)(\Phi_i^T(\mathbf{s}_j)) = \sum_{j=1}^{t} \rho^{(t-j)} R_j \Phi_i(\mathbf{s}_j), \tag{2}$$

which can be written as:

$$\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}, \tag{3}$$

where

$$\begin{array}{ll} \mathbf{A} = & \sum_{j=1}^{t} \rho^{(t-j)} \Phi_i(\mathbf{s}_j) \Phi_i^T(\mathbf{s}_j), \text{ and} \\ \mathbf{b} = & \sum_{j=1}^{t} \rho^{(t-j)} R_j \Phi_i(\mathbf{s}_j). \end{array} \tag{4}$$

We must also obtain a measure of e_i to use when comparing sensorimotor abstractions. Equation 1 can be expanded to:

$$e_i = \sum_{j=1}^{n} \rho^{(t-j)} [(\mathbf{w} \cdot \Phi_i(\mathbf{s}_j))^2 - 2R_j \mathbf{w} \cdot \Phi_i(\mathbf{s}_j) + R_j^2]. \tag{5}$$

Dropping the R_j^2 term since it is the same for all sensorimotor abstractions, we obtain:

$$\hat{e}_i = \sum_{j=1}^{n} \rho^{(t-j)} [(\mathbf{w} \cdot \Phi_i(\mathbf{s}_j))^2] - 2 \sum_{j=1}^{n} \rho^{(t-j)} [R_j \mathbf{w} \cdot \Phi_i(\mathbf{s}_j)], \tag{6}$$

which can be written as:

$$\hat{e}_i = \mathbf{w}^T \mathbf{A} \mathbf{w} - 2\mathbf{w} \cdot \mathbf{b}, \tag{7}$$

with $\mathbf{A}$ and $\mathbf{b}$ defined as before. This leads to an incremental (least-squares weighted TD(1)) algorithm for obtaining the weights and associated error for a skill value function shown in Figure 2. The algorithm should be run simultaneously for each sensorimotor abstraction during the sample trajectory, and the abstraction with the lowest returned error should be selected for use in learning the skill.

function Sensorimotor Abstraction Fit (i, ρ) :

1) **Initialization:**
   Set $\mathbf{A_0}, \mathbf{b_0}, \mathbf{z_0}, \mathbf{c_0}$ and d_0 to 0

2) **Iteratively handle incoming samples:**
   for each incoming sample $(\mathbf{s}_t, a_t)$:

   $\mathbf{A_t} = \rho \mathbf{A_{t-1}} + \Phi_i(\mathbf{s}_t)\Phi_i^T(\mathbf{s}_t)$
   $\mathbf{b_t} = \rho \mathbf{b_{t-1}} + \rho r_t \mathbf{z_{t-1}} + r_t \Phi_i(\mathbf{s}_t)$
   $\mathbf{z_t} = \rho \mathbf{z_{t-1}} + \Phi_i(\mathbf{s}_t)$

   $\mathbf{c_t} = \rho \mathbf{c_{t-1}} + a_t \Phi_i(\mathbf{s}_t)$
   $d_t = \rho d_{t-1} + a_t^2$

3) **Compute weights and error:** (after n samples)
   $\mathbf{w} = \mathbf{A_n}^{-1}\mathbf{b_n}$
   $\hat{e} = \mathbf{w}^T \mathbf{A_n} \mathbf{w} - 2\mathbf{w} \cdot \mathbf{b_n}$

   $\mathbf{w}_\pi = \mathbf{A_n}^{-1}\mathbf{c_n}$
   $e_\pi = \mathbf{w_\pi}^T \mathbf{A_n} \mathbf{w}_\pi - 2\mathbf{w}_\pi \cdot \mathbf{c_n} + d_n$

4) **Return** $\mathbf{w}$, $\hat{e}$, $\mathbf{w}_\pi$ and e_π.

Fig. 2. An incremental algorithm for obtaining the value function weights ($\mathbf{w}$), policy parameters ($\mathbf{w}_\pi$) and associated errors for a skill, given a sensorimotor abstraction and a successful sample trajectory.

Although the algorithm returns error measures for both a policy and a value function, our experience indicates that the value function error measure is a better metric for state space selection. This is because the sample policy may be easy to reproduce (e.g., a constant velocity for a given period of time) without reference to environmental features, whereas a value function contains more useful information (usually at least a value gradient) that the sensorimotor abstraction must represent.

More than one sample trajectory may be available, or required to produce robust selection. Given m samples, the algorithm can be modified to run steps 1 and 2 (initialization and incoming sample processing) separately for each sample trajectory, and then sum the resulting variables. Steps 3 and 4 (computing and returning the parameter vector and error measure) using the summed variables then performs a fit over all m trajectories simultaneously.

The algorithm uses $O(k_i^2)$ memory, $O(k_i^2)$ time at each step and $O(k_i^3)$ time at selection for every sensorimotor abstraction i using a function approximator with k_i features. Once an abstraction has been selected, its weight vector $\mathbf{w}_\pi$ represents a policy obtained by fitting the sample trajectory, and is thus hopefully a good initial policy from which to begin learning.

If the algorithm is too expensive for real-time execution, a function approximator of the same type but with fewer basis functions could be used for space selection (which reduces k_i) and then upgraded for learning, which reduces the per-sample complexity but may still result in a usable initial policy.

IV. Experimental Platform

A. Simulator

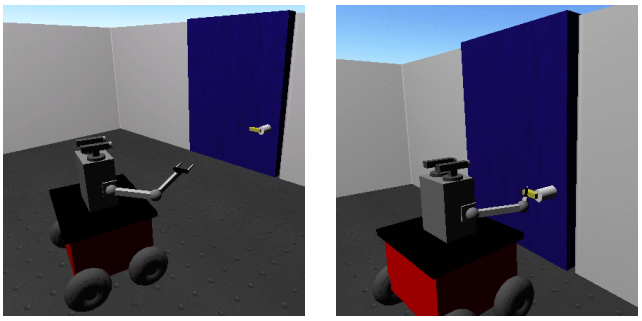


Fig. 3. Spangle, the physically realistic simulated robot used as an experimental platform for state space selection. Spangle's goal is to approach the door it is facing, open it by first turning its handle and then pushing it open, and then roll through it to exit the room.

Spangle, the simulated robot platform used in this section (developed using Gazebo [10]), is shown in Figure 3. Spangle moves using four wheels that can drive the robot's base forward or backward, or rotate in either direction. Manipulation is achieved with an arm with 3 rotational degrees of freedom, a wrist with 1 rotational degree of freedom and a gripper. Spangle has a pair of cameras, a ring of 24 sonars, and gripper touch and beam sensors for sensing its environment. Touch sensors along its arm, wrist and gripper fingers are used for collision detection.

Spangle's task is to approach the door shown in Figure 3, grasp its handle, turn it $\frac{\pi}{4}$ radians (whereupon it stays turned and unlocks the door), push the door open and roll through it to exit the room. A sample trajectory was hand-coded using a sequence of PID controllers for the arm and timed velocity settings for navigation.

B. Sensorimotor Couplings

Table I shows the sensor and motor abstraction programs available to Spangle. For simplicity and ease of exposition these have been kept to a small and relatively simple set.

These were combined into 20 sensorimotor abstractions, with some combinations ruled out as unlikely to ever be useful, and in some cases multiple sensor spaces combined with a single motor space. Each abstraction was assigned approximately 200 Radial Basis Functions, with each variable tiled separately. The fixed size function approximation is loosely based on estimated real time update constraints, so that larger spaces allow fewer function approximation terms per variable, resulting in an implicit form of regularization.

Motor Program	**Controlled Variables**
Arm	Three arm revolute joint motors, revolute wrist joint and gripper.
GraspedArm	Three arm revolute joint motors, with the wrist and gripper running a policy that maintains an existing grasp as the arm moves.
Navigation	Speed and rate of turn.

Sensor Space	**Description**
SonarObject	Angles to the center, left, right and distance to the center of an object detected by sonar.
SonarHole	Angles to the center, left, right and distance to the center of a hole detected by sonar.
Blob	Height, width and (x, y) coordinates of an object identified by color image segmentation. Bindings are labelled *Handle*, *Wrist* and *Door* (for expository purposes only; these are not given to the robot).
BlobDistance	Height, width and (x, y) coordinates for one object (again identified by color image segmentation), and height, width and (x, y) distance (to the first object) for another.
Arm	Joint positions for the arm and wrist.
GraspedArm	Joint positions for the arm.

TABLE I
SPANGLE'S MOTOR PROGRAMS AND SENSOR SPACES.

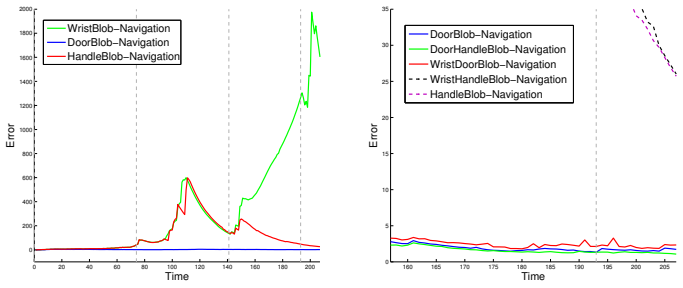
V. Results

We split the sample trajectory into six component skills, and consider the results of applying state space selection to each component skill in turn. For simplicity, we assume a cost function of -1 per timestep and a subgoal completion reward of 1000 for reaching the end of a subskill, and set $\rho = 0.95$.

A. Selected Spaces

In the first segment, Spangle approaches the door, positioning itself to turn the handle. To visualize the change in error as the trajectory is executed, Figure 4 shows two different views of the trajectory, where the state space selection error is computed at every timestep (instead of solely at the end). Figure 4a shows the error for three representative state spaces. The abstraction that uses the visual features of the robot's wrist has a consistently high error, indicating that it is not a useful state space for this trajectory. The abstraction that uses the visual features of the Handle starts out with a high error (because the handle is too far away to be visible in the beginning of the trajectory) but drops quickly as the robot nears the door, indicating that it is only useful near the end of the trajectory. Finally, the state space that uses the visual features of the door has a consistently very low error.

Figure 4b shows the five lowest error state spaces in the last 50 samples of the trajectory. The state space with the lowest error uses the visual features of both the door and the handle. This is because once the robot nears the door, its image fills

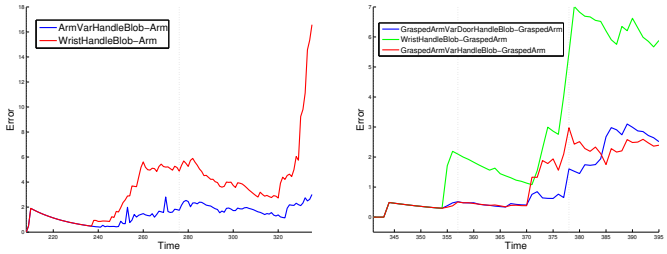


(a) (b)

Fig. 4. Error trend graphs for representative state spaces for (a) the segment of the trajectory where Spangle approaches the door, and (b) the last 50 samples of that segment.

the robot's visual field and thus its centroid and dimensions do not change. Thus, it is best to use the door's visual features to approach the door, and then the handle's visual features when very near to the door. Note that the "WristDoorBlob" abstraction has higher error than the "DoorBlob" abstraction even though "WristDoorBlob" includes every variable in "DoorBlob". This shows that implicit regularization does indeed penalize larger state spaces.

The second segment, where Spangle reaches out and grasps the handle of the door, is best accomplished using the visual features of the handle and joint positions of the wrist, although early in the trajectory the visual features of the handle and wrist are better; this error trend is shown in Figure 5a. This suggests that the visual features of the wrist are sufficient until the wrist is very near the handle, but the actual grasp requires more precise joint positions. Figure 5b shows that the same is true for turning the handle.



(a) (b)

Fig. 5. Error trend graphs for representative state spaces for (a) the segment of the trajectory where Spangle grasps the handle of the door, and (b) the segment where Spangle turns the handle.

In the fourth segment Spangle releases the handle and positions its arm to push the door open. This is best accomplished using the arm position variables and the visual features of the handle. Thereafter, pushing the door open is best accomplished using the visual features of the door and the handle, similarly to the initial approach of the door.

Finally, once the door is open, the best state space to use for rolling through it uses the features of a hole detected using sonar. The error trend graph in Figure 6 shows that although the visual features of the door are preferred initially, once the robot is inside the doorway their error increases while the error associated with using sonar does not. This demonstrates the utility of multiple potential state spaces, since it shows that the appropriate state space for going through a door is different from the one most appropriate for approaching it.

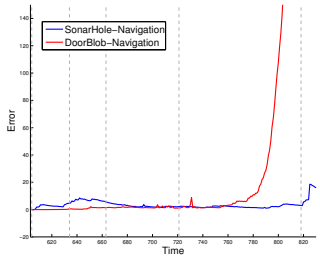


Fig. 6. Error trend graphs for representative state spaces for the segment of the trajectory where Spangle rolls through the open door.

B. Initial Policy Fits

Figures 7 and 8 show example sample and policy fit outputs for the sample trajectory as Spangle approaches the door and grasps the handle. Each individual graph shows the changes in torque applied to an individual motor control variable. The policy fits are a close match to the sample trajectory, and when run in the simulator provide qualitatively similar behavior. They are thus suitable initial policies for a reinforcement learning algorithm, especially one that moves from a training policy to a refined solution (e.g., Rosenstein and Barto [7]).

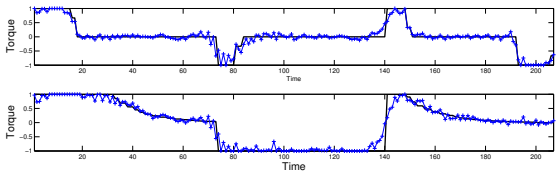


Fig. 7. Sample (thick line) and policy fit outputs while approaching the door.

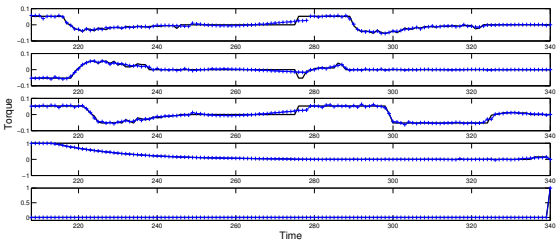


Fig. 8. Sample (thick line) and policy fit outputs while grasping the handle.

C. Summary

These results demonstrate that even an everyday task such as passing through a door can benefit from the use of several different sensorimotor abstractions. The state space selection algorithm selects an appropriate state space for each subtask given a single sample trajectory, and fits a good initial policy suitable for policy refinement using reinforcement learning.

VI. Related Work

Existing reinforcement learning research on state space reduction takes one of two broad approaches. In *state abstraction* (surveyed by Li, Walsh and Littman [11]), a large state space is compressed by performing variable removal or state aggregation while approximately preserving some desirable property. However, without further information about the state space we cannot examine the effects of abstraction on the properties we are interested in—values or policies—without an existing value function. For most useful robots, a value function in such a space is infeasibly difficult to learn and may not even be feasible to represent.

The major alternative approach is to initially assume that *no* states or state variables are relevant, and then introduce *perceptual distinctions* [12] by including them when it becomes evident that they are necessary for learning the skill. This requires a significant amount of data and computation to determine which variable to introduce, and then introduces them one at a time, which may require too much experience to be useful for real time learning.

Rather than starting with all features and removing some, or starting with none and adding one at a time, we start with a fixed number of feature sets and select one prior to learning.

The work most closely related to ours is van Seijen et al. [13], where an agent with multiple representations is augmented with actions to switch between them. This fits neatly into the reinforcement learning framework, but does not appear likely to scale up to large numbers of representations since each new representation adds a new action to every state.

VII. Discussion and Future Work

Sensorimotor abstraction selection shifts the state design problem from one of designing a problem-specific state space to one of designing a set of state spaces sufficient to deal with any skill that a robot might decide to learn. This results in some significant benefits. We can use knowledge of the class of tasks the robot faces to design and constrain those abstractions. We can also include a large number of abstractions, since selecting one is very much faster than learning a skill, and if necessary the selection algorithm can be parallelized since each abstraction can be processed completely independently.

However, space selection presents two disadvantages. First, each state space is likely to support only local (low-range) skills. For example, the skills used to approach a door, insert a key, turn the key, grasp the handle, turn the handle, open the door, and walk through it can all only be learned efficiently using small but distinct state spaces. However, in the context of an autonomous robot, learning a small set of component skills that must be executed in sequence may be more useful than a single monolithic skill, since each component skill can be retained for later use. This requires a method for determining the goal and range of each skill. In future work we aim to show that learning modular component skills using *skill chaining*—creating a sequence of skills where goal of each skill is to allow the skill following it to be executed—is more efficient than learning a single monolithic skill.

The second disadvantage is that the skills an autonomous robot will be able to learn will be restricted by the set of abstractions given to it. Although this is a significant drawback it is still an improvement over entirely task-specific learning. In future work, we intend to add a process that learns sensorimotor abstractions over the lifetime of the robot.

VIII. Conclusion

We have shown that the sensorimotor abstraction selection algorithm presented here selects appropriate state spaces for a sequence of skills along a given sample trajectory.

Choosing an appropriate representation is critical to the successful application of reinforcement learning to real world problems, but to achieve true autonomy that choice must be made by the robot, not its designer. Selecting from a set candidate state spaces rather than relying on a single designed state space shifts the design element out of the robot's control loop, thus removing an obstacle to autonomous learning.

Acknowledgments

We would like to thank Sarah Osentoski, Steve Hart, Ashvin Shah, Sridhar Mahadevan, Roderic Grupen and our reviewers for their useful comments, and Valerie Caro for indispensible technical support.

References

[1] R. Arkin, *Behavior-Based Robotics*. Cambridge, Massachusetts: MIT Press, 1998.

[2] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[3] R. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[4] R. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems 12*, 1999, pp. 1057–1063.

[5] J. Baxter and P. L. Bartlett, "Direct gradient-based reinforcement learning," in *Proceedings of the International Symposium on Circuits and Systems*, 2000, pp. III–271–274.

[6] M. Huber and R. Grupen, "Learning to coordinate controllers - reinforcement learning on a control basis," in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997, pp. 1366–1371.

[7] M. Rosenstein and A. Barto, "Supervised actor-critic reinforcement learning," in *Learning and Approximate Dynamic Programming: Scaling up the Real World*, J. Si, A. Barto, A. Powell, and D. Wunsch, Eds. New York: John Wiley & Sons, Inc., 2004, pp. 359–380.

[8] S. Schaal, "Learning from demonstration," in *Advances in Neural Information Processing Systems 9*, 1997.

[9] J. Boyan, "Least squares temporal difference learning," in *Proceedings of the 16th International Conference on Machine Learning*, 1999, pp. 49–56.

[10] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Proceedings of the International Conference on Intelligent Robots and Systems*, 2004, pp. 2149–2154.

[11] L. Li, T. Walsh, and M. Littman, "Towards a unified theory of state abstraction for MDPs," in *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 2006.

[12] A. McCallum, "Learning to use selective attention and short-term memory in sequential tasks," in *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, 1996.

[13] H. van Seijen, B. Bakker, and L. Kester, "Reinforcement learning with multiple, qualitatively different state representations," in *Proceedings of NIPS 2007 Workshop on Hierarchical Organization of Behavior*, 2007.