

Self-Organizing Distinctive-State Abstraction For Learning Robot Navigation*

Jefferson Provost, Benjamin J. Kuipers, and Risto Miikkulainen
Artificial Intelligence Lab
The University of Texas at Austin
1 University Station C0500
Austin, TX 78712 USA
{jp,kuipers,risto}@cs.utexas.edu

July 19, 2005

Abstract

A major challenge in reinforcement learning research is to extend methods that have worked well on discrete, short-range, low-dimensional problems to continuous, high-diameter, high-dimensional problems, such as robot navigation using high-resolution sensors. Self-Organizing Distinctive-state Abstraction (SODA) is a new, generic method by which a robot in a continuous world can better learn to navigate by learning a set of high-level features and building temporally-extended actions to carry it between distinctive states based on those features. A SODA agent first uses a self-organizing feature map to develop a set of high-level perceptual features while exploring the environment with primitive, local actions. The agent then builds a set of high-level actions composed of generic trajectory-following and hill-climbing control laws that carry it between the states at local maxima of feature activations. In an experiment on a simulated robot navigation task, the SODA agent learns to perform a task requiring 300 small-scale, local actions using as few as 9 new, temporally-extended actions, significantly improving learning time over navigating with the local actions.

1 Introduction

Modern robots are endowed with rich, high-dimensional sensory systems, providing measurements of a continuous environment. In addition, many important real-world robotic tasks have *high diameter*, that is, their solutions require a large number of primitive actions by the robot, for example, navigating to distant locations using primitive motor control commands. Reinforcement learning (RL) methods show promise for automatic learning of robot behavior, but extending these methods to high-dimensional, continuous, high-diameter problems remains a major challenge. Thus, the success of RL on real-world tasks still depends on human analysis of the robot, environment, and task to provide a useful set of perceptual features and an appropriate decomposition of the task into subtasks. A major goal of AI research, however, is to create autonomous learning agents, and the “hard part” of learning ultimately needs to be performed by the agent, rather than the human engineer.

Self-Organizing Distinctive-state Abstraction (SODA) is a new method for automatic discovery of high-level perceptual features and large-scale actions for learning to navigate in continuous environments. This paper presents a first implementation of the algorithm in which the features are learned and the high-level action policies are specified by generic control laws based on those features. The control laws encode no specific information about the robot’s sensorimotor system, environment, or task, yet the new actions significantly speed learning on a high-diameter navi-

*This paper appears as University of Texas Artificial Intelligence Lab Technical Report AI-TR-05-319. This work was supported in part by the National Science Foundation (grant IIS-0413257), by NIMH Human Brain Project (grant IR01-MH66991), and by an IBM Faculty Research Award.

gation task. The implementation is therefore a significant step towards a generic, autonomous learning agent for robot navigation.

Given high-dimensional, continuous-valued sensory input, and continuous motor output, SODA works as follows:

1. Learn a set of high-level perceptual features that define *distinctive states* in the environment by training a Self-organizing Feature Map (Kohonen, 1995) while exploring the environment with primitive actions.
2. Construct a set of high-level actions that carry the robot from one distinctive state to another using generic trajectory-following and hill-climbing control laws.
3. Learn policies for high-diameter tasks through reinforcement learning in the abstracted space of high-level features and actions.

In order for the control-laws to work, SODA relies on the assumption that the world is continuous, i.e. small actions generally induce small changes in sensor values. This assumption is usually valid in robotic navigation.

This paper is organized as follows: The next section describes the foundations of SODA in self-organizing maps and robot navigation, as well as related work in reinforcement learning. The following section describes the SODA method in detail. The remainder of the paper describes a preliminary experiment with the method that shows a large reduction in task diameter using the learned high-level actions, and discusses ongoing and future work on SODA.

2 Background and Related Work

The state abstraction in SODA applies the concept of the winning unit from Self-Organizing Feature Maps, and the action abstraction applies the concept of the distinctive state from the Spatial Semantic Hierarchy (Kuipers, 2000).

2.1 Self-Organizing Feature Maps

SODA constructs perceptual features by using the sensory input as training data for an unsupervised self-organizing feature map (SOM) (Kohonen, 1995) that learns a set of *sensory prototypes* to represent its sensory experience.

A standard SOM consists of a set of units arranged in a lattice. The SOM takes a continuous-valued vector \mathbf{x} as input and returns one of its units as the output. Each unit i has a weight vector \mathbf{w}_i of the same dimension as the input. On the presentation of an input, each weight vector is compared with the input using the Euclidean distance and a *winner* is selected as $\arg \min_i \|\mathbf{x} - \mathbf{w}_i\|$. The winner and its lattice neighbors are then adjusted toward \mathbf{x} .

The SOM has been used previously for learning perceptual features and state representations in general robotics (Martinetz, Ritter, & Schulten, 1990; Duckett & Nehmzow, 2000; Nehmzow & Smithers, 1991; Provost, Beeson, & Kuipers, 2001; Chaput, Kuipers, & Miikkulainen, 2003), as well as for learning state abstraction in RL (Smith, 2002). All these methods use the SOM as a clustering or vector quantization method, ignoring the variation between the inputs that produce the same winner. The SOM, however, can provide both a coarse-grained discretization of its input space, and a set of continuous features (or “activations”) defined in terms of the distance of the input from each unit. These properties divide the continuous state space into a set of neighborhoods defined by their winning unit, each containing a stable fixed point at the local maximum of the winner’s activation. SODA uses this feature to construct actions, as described in the next section.

The implementation in this paper uses a variant on the standard SOM algorithm called the Growing Neural Gas (GNG) (Fritzke, 1995), that begins with a small set of units and inserts new units incrementally to minimize distortion error. The GNG is able to continue learning indefinitely, adapting to changing input distributions. This property makes the GNG especially suitable for robot learning, since a robot experiences its world sequentially, and may experience entirely new regions of the input space after an indeterminate period of exploration. In addition, the GNG is not constrained by the pre-specified topology of the SOM lattice. It learns its own topology in response to experience with the domain.

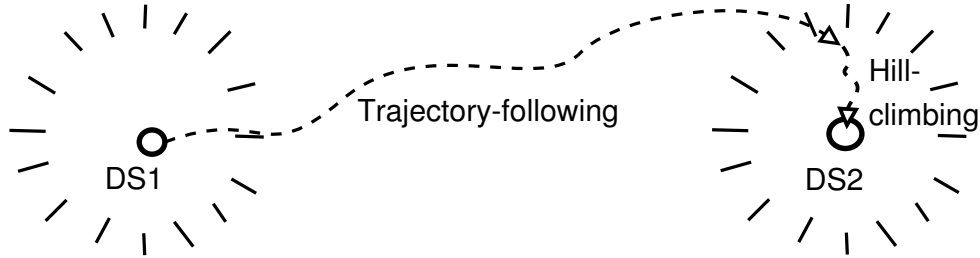


Figure 1: **High-level Actions.** The agent travels from one distinctive state to another using an action with two parts: first a *trajectory-following* (TF) controller drives the robot into the neighborhood of a new sensory prototype, then a *hill-climbing* (HC) controller takes the robot to the local state that best matches that prototype.

2.2 Abstract Actions in the Spatial Semantic Hierarchy

To create high-level actions, the agent uses the abstraction from the *control level* to the *causal level* of the *Spatial Semantic Hierarchy* (SSH), a theory of representation of large-scale space (Kuipers, 2000). At the control level, there are two kinds of control laws: *Trajectory-following* (TF) control laws carry the robot from one distinctive state into the neighborhood of another, while *Hill-climbing* (HC) control laws carry the robot to a distinctive state within the neighborhood, by climbing the gradient of some “distinctiveness measure” on the sensory input (See Figure 1). The causal level defines a set of high-level actions each consisting of a TF/HC control-law pair that carries the robot from one distinctive state to another. The benefit of these actions is twofold: the TF component gives the actions extent, allowing the agent to move through its environment in large steps, while the HC component reduces positional uncertainty by bringing the robot to a fixed state at the end of each action.

2.3 Temporal Abstraction

There has been considerable research into automatically discovering high-level actions for reinforcement learning (Digney, 1998; McGovern & Barto, 2001; Ryan, 2002; Hengst, 2002). These methods all assume that a continuous-to-discrete abstraction already exists, and they search for higher-level temporal abstractions in the (already abstracted) discrete Markov decision process. SODA assumes a continuous state space and discovers a continuous-to-discrete abstraction of both perceptual and action space that results in temporally extended, abstract actions, using the gradients of continuous features to define the high-level actions. The above methods and SODA are potentially complementary. In very large problems it is likely that multiple levels of abstraction will be needed. Eventually we expect to use the methods above to perform additional temporal abstraction on top of the continuous-to-discrete abstraction SODA provides.

3 Learning Method and Representation

The SODA algorithm can be characterized formally as follows. Given

- a robot with a sensory system providing experience as a sequence of N-dimensional, continuous *sensory vectors* $\mathbf{y}_1, \mathbf{y}_2, \dots$, where every $\mathbf{y}_t \in \mathbb{R}^N$,
- a continuous, M-dimensional motor system that accepts from the agent a sequence of *motor vectors* $\mathbf{u}_1, \mathbf{u}_2, \dots$ where every $\mathbf{u}_t \in \mathbb{R}^M$,
- a continuous world, in which small actions induce small changes in sensor values,
- and a scalar *reward signal*, r_1, r_2, \dots , that defines a high-diameter task (that is, maximizing the expected value of r requires a long sequence of motor vectors $\mathbf{u}_t, \dots \mathbf{u}_{t+k}$).

the algorithm performs these steps:

1. Define a set of discrete, local primitive actions \mathcal{A}^0 : First, using methods developed by Pierce & Kuipers (1997), learn an *abstract motor interface*, a basis set of orthogonal motor vectors $\mathcal{U} = \{\mathbf{u}^0, \mathbf{u}^1, \dots \mathbf{u}^{n-1}\}$ spanning the

<p>Trajectory-follow on a_i^0:</p> <p style="padding-left: 2em;">$f_w \leftarrow \arg \max_{f \in \mathcal{F}} f(\mathbf{y})$</p> <p style="padding-left: 2em;">while $f_w = \arg \max_{f \in \mathcal{F}} f(\mathbf{y})$:</p> <p style="padding-left: 4em;">execute action a_i^0</p> <p>$f_{ds} \leftarrow \arg \max_{f \in \mathcal{F}} f(\mathbf{y})$</p> <p>Hill-climb on f_{ds}:</p> <p style="padding-left: 2em;">while $\max_j \Delta^j f_{ds} > 0$:</p> <p style="padding-left: 4em;">$w \leftarrow \arg \max_j \Delta^j f_{ds}$</p> <p style="padding-left: 4em;">execute action a_w^0</p>

Table 1: Trajectory-following/Hill-climbing Pseudo-code. $\Delta^j f_i$ is the gradient of feature f_i with respect to primitive action a_j^0 .

set of motor vectors \mathbf{u}_t possible for the robot. Then define \mathcal{A}^0 to be the set of $2n$ discrete actions derived from \mathcal{U} by setting the motor signal \mathbf{u}_t to the value \mathbf{u}^i or $-\mathbf{u}^i$ for a short time period Δt .

2. Learn a set \mathcal{F} of high-level perceptual features: Exploring the environment with a random sequence of \mathcal{A}^0 actions, train a SOM with the sensor signal \mathbf{y}_t to converge to a set of high-level features of the environment. For each weight vector \mathbf{w}_i in the SOM, there is an *activation function* $f_i \in \mathcal{F}$ such that:

$$f_i(\mathbf{y}) = \frac{\hat{f}_i(\mathbf{y})}{\sum_{j=0}^{|\mathcal{F}|-1} \hat{f}_j(\mathbf{y})} \text{ where } \hat{f}_j(\mathbf{y}) = \frac{1}{\|\mathbf{y} - \mathbf{w}_j\|^z} \quad (1)$$

These equations define an activation for each unit that varies with the inverse of the Euclidean distance of that unit from the input vector. The activation across all units is normalized to sum to unity. The *response exponent* z is a parameter that controls the “width” or “focus” of the response, with higher z allocating more activation to the winner.

3. Define a hill-climbing (HC) control law for each $f_i \in \mathcal{F}$: For each f_i in the context where $\arg \max_{f \in \mathcal{F}} f = f_i$, estimate its gradient with respect to each primitive action a_j^0 in \mathcal{A}^0 , denoted $\Delta^j f_i$. This estimate is currently found by applying each action, sampling the feature value, and reversing the action. The HC control-law selects the action with the highest gradient, terminating when all the gradients are negative (See Table 1). The state of the agent when hill-climbing terminates is defined to be a *distinctive state* (dstate).
4. For each distinctive state defined by $f_i \in \mathcal{F}$, define trajectory-following control laws that take the agent to a state where a different feature $f_j \in \mathcal{F}$ is dominant. Currently, the algorithm proposes one such TF control law for each primitive action $a_k^0 \in \mathcal{A}^0$, consisting of simply repeating a_k^0 until the dominant feature f_i is replaced by another feature f_j (See Table 1).
5. Define a set of higher-level actions \mathcal{A}^1 where each $a_m^1 \in \mathcal{A}^1$ consists of executing one TF control law, and then hill-climbing on the resulting dominant feature f_j . At this point the agent has abstracted its continuous state and action space into a discrete Markov Decision Process (MDP) with one state for each feature in \mathcal{F} , and the large-scale actions in \mathcal{A}^1 .

Once the agent has abstracted its world into an MDP defined in terms of the winning GNG units and the \mathcal{A}^1 actions it, can learn its task using standard model-based or model-free RL methods, or even propositional planners or A*, if the transitions are sufficiently deterministic.

4 Experiment

This section presents an experiment demonstrating SODA significantly reducing the task diameter in a robot navigation task, using the MDP abstraction from step 5 of the algorithm (see above), and model-free reinforcement learning.

In the experiment, a SODA agent is instantiated in a simulated robot using the Stage simulator (Gerkey, Vaughan, & Howard, 2003). The environment is a T-shaped room or maze, shown in Figure 2, measuring 10,000 mm \times 6,000 mm. The robot has a simple drive/turn motor system, and the agent is assumed to have already identified the two dimensional abstract motor interface \mathcal{U} shown in Table 2(a). Stage does not simulate acceleration, so velocity changes

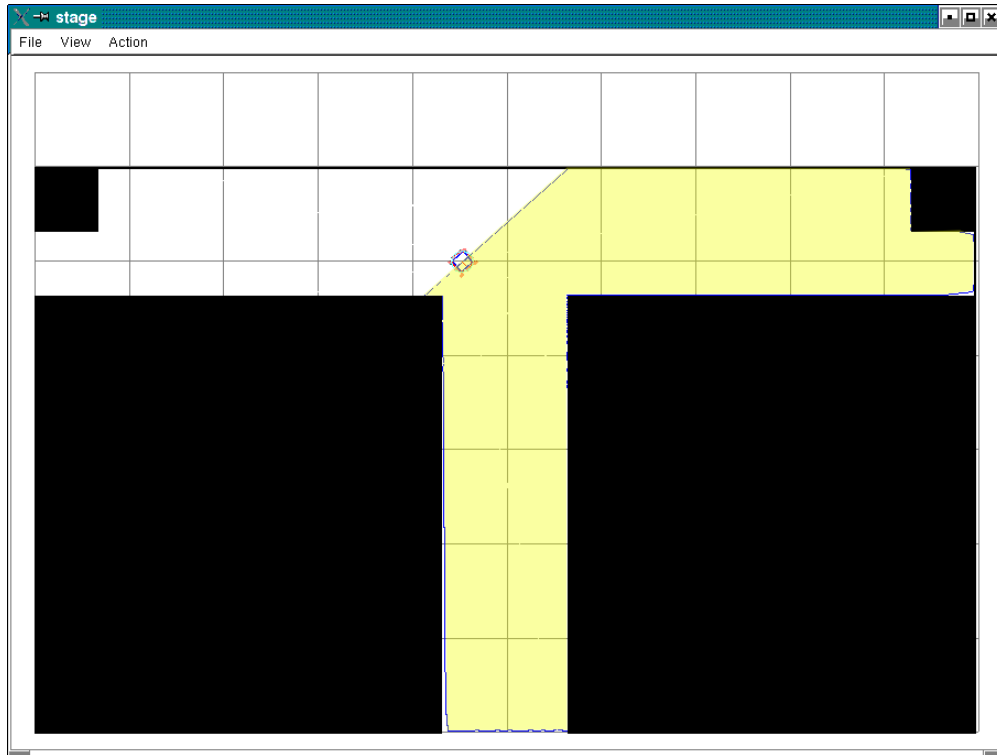


Figure 2: **Simulated Robot Environment.** .

A screen-shot of the Stage robot simulator with the simulated robot and experimental environment. The robot has a drive-and-turn base, and a laser rangefinder. The agent must learn to travel from the left end of the upper hallway to the end of the lower hallway. The shading indicates the area swept out by the laser rangefinder.

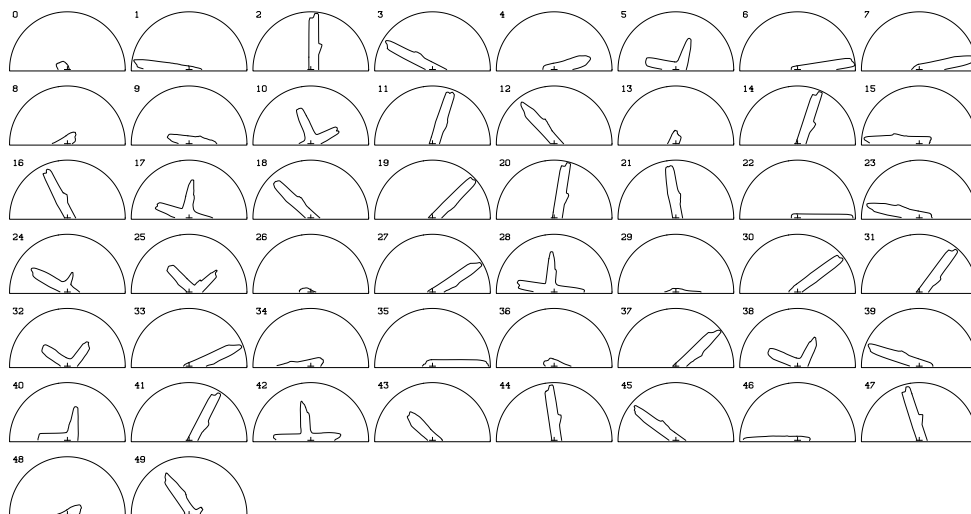


Figure 3: **Example Learned Perceptual Prototypes..** The agent's self-organizing feature map learns a set of perceptual prototypes that are used to define perceptually distinctive states in the environment. This figure shows the set of features learned from one run. Each feature is a prototypical laser rangefinder image plotted radially, with the robot at the origin.

(a) Abstract Motor Interface		
	\mathbf{u}^0	\mathbf{u}^1
drive	250 mm/sec	0 mm/sec
turn	0°/sec	90°/sec

(b) Primitive Actions \mathcal{A}^0			
action	label	\mathbf{u}	step
a_0^0	<i>ahead-short</i>	\mathbf{u}^0	25mm
a_1^0	<i>back-short</i>	$-\mathbf{u}^0$	-25mm
a_2^0	<i>left-short</i>	\mathbf{u}^1	9°
a_3^0	<i>right-short</i>	$-\mathbf{u}^1$	-9°

Table 2: Abstract Motor Interface and Primitive Actions

instantaneously. The agent’s input \mathbf{y} consists of the response from a single simulated SICK LMS laser rangefinder, providing 180 range readings over the forward semi-circle, with a maximum range of 8000 mm and a resolution of 10 mm. The simulator accepts motor commands and provides sensations at 10 Hz (simulator time). The abstract motor interface is quantized at this rate to provide four \mathcal{A}^0 actions consisting of positive and negative steps along each axis of motor control. These actions are described in Table 2(b).

The agent’s task is to drive from the left end of the upper hallway to the bottom of the center hallway. The task terminates when the robot reaches within 500 mm of the goal (a point 500 mm from the end of the lower corridor), or times out after 10,000 simulator steps. The reward on each non-terminal step is -1 unless the robot collides with a wall, in which case it is -5. The reward upon successful termination is 0. This scheme rewards finding the shortest path to the goal while not bumping into walls.

To learn the initial set of perceptual features, the agent trains a Growing Neural Gas network with its sensory input over a 500,000 step random walk through the environment (the equivalent of 50 task learning episodes). The GNG parameters are $\lambda = 2000$, $\alpha = 0.05$, $\beta = 0.0005$, $e_b = 0.05$, $e_n = 0.0006$, $a_{max} = 100$ (Fritzke, 1995). The GNG was configured to grow only if the average cumulative distortion error across all units was greater than 37500. A typical set of learned features is shown in Figure 3. This set comprises a wide variety of prototypical views of the environment. Feature activations are computed according to Equation (1), with response exponent $z = 4$.

The agent has four simple, open-loop trajectory-following control laws, one for each \mathcal{A}^0 action, as described in step 4 of the algorithm. The agent learns its high-level control policy using episodic, tabular SARSA(λ) reinforcement learning (Sutton & Barto, 1998), using the SOM winner as the state. SARSA parameters¹ used are $\lambda = 0.9$, $\alpha = 0.2$, $\gamma = 1.0$. All Q values are initialized optimistically to 0.0, and actions are selected greedily with ties broken randomly. These parameter values for both the GNG and SARSA were found to be reasonable after some manual experimentation, but, in keeping with the spirit of autonomous learning, no exhaustive parameter search was performed. Learning trials consisted of 12 runs of 5000 episodes in each of two experimental conditions, the first with only \mathcal{A}^0 actions, the second with only \mathcal{A}^1 actions. This experiment used 10 different trained SOMs for a total of 240 runs.

4.1 Results

Figure 4 demonstrates that the agents learn to perform the task much faster using the high-level actions than without them. Figure 5 shows a typical task solution learned using high-level actions. Figure 6 shows the sequence of features that define the distinctive states in that solution, in the order that they are encountered in the solution. These figures show that the \mathcal{A}^1 actions are highly abstracted compared to the \mathcal{A}^0 actions. In particular the first and eighth actions can be interpreted as “travel down the hall” and comprise large numbers of primitive actions, and even the five smaller actions used to turn in the intersection are considerably larger than single \mathcal{A}^0 actions.

¹SARSA parameters λ and α have different meanings than the GNG parameters with the same names.

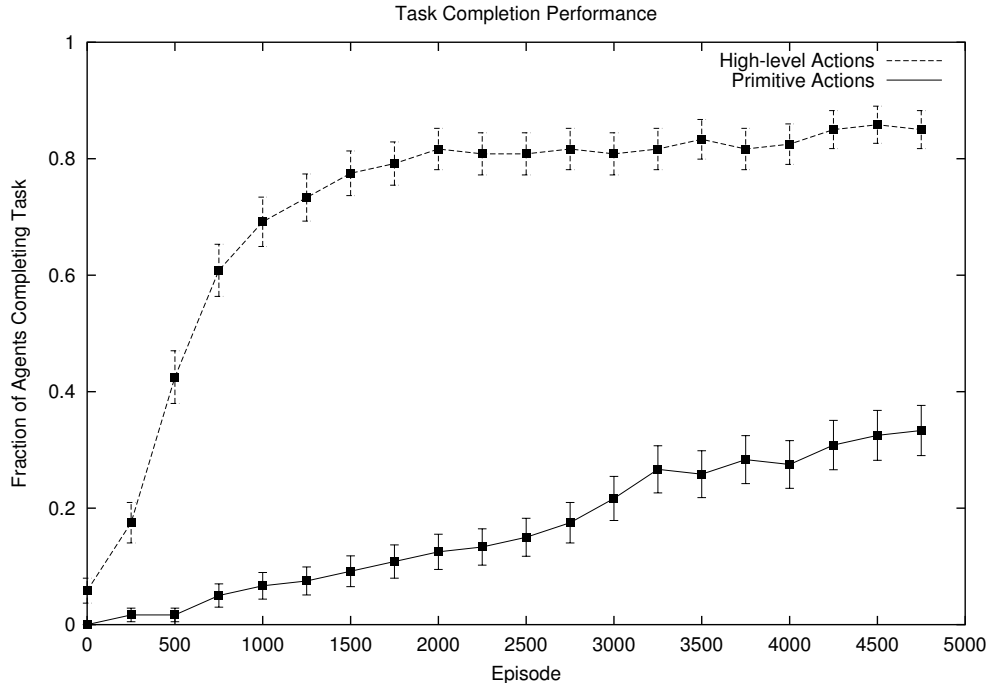


Figure 4: **Learning Performance** Comparison of the fraction of agents completing the task within each 10,000 time-step episode using primitive (\mathcal{A}^0) actions vs. using high-level (\mathcal{A}^1) actions. Each curve is an average of 12 runs using each of 10 different learned feature sets. Error bars indicate +/- one standard error. Agents using the \mathcal{A}^1 actions learn the task much faster.

5 Discussion and Future Work

The results of these experiments demonstrate that SODA can dramatically improve reinforcement learning over using the same coarse-coded state representation and short-range, \mathcal{A}^0 actions. Ongoing work on SODA includes learning the TF and HC policies, disambiguating aliased states, and evaluating the method on larger, more realistic navigation problems.

The improved task learning in this experiment follows from two sources: reduced positional uncertainty and, reduced task diameter. First, state abstraction methods that partition a continuous state space alias the environment, creating uncertainty about the outcome of actions. For example, if two states in the same partition differ in orientation by a few degrees, moving forward may lead to drastically different states. Hill-climbing to the local feature maximum greatly reduces this uncertainty and thus makes actions more reliable, making it easier to learn the task.

Second, using the short-range actions, the task has a large diameter: the agent must make at least 300 actions to get to the goal. Since a critical choice that must be learned – turning right at the intersection – is approximately halfway through the action sequence, it takes many trials and much exploration to back up the reward to the critical decision point and discover the correct choice. In contrast, traveling between distinctive states using \mathcal{A}^1 actions, the agent needs only nine actions to arrive at the goal. This decrease in task diameter makes it much easier to propagate the reward back to the critical choice point, and thereby discover critical decisions in the task.

When using a coarse-grained perceptual representation like a SOM, there is sure to be state aliasing. For example, in the task in the experiment above, the agent passes through two different states represented by feature 44 *en route* to the goal (the first and eighth states in Figure 6). In this case, the optimal action is the same in both cases, but in general, the optimal actions could be different and the agent would need to disambiguate aliased states. If, the robot’s perceptual system may be rich enough to disambiguate all aliased states (Kuipers & Beeson, 2002), it may be appropriate to replace tabular SARSA(λ) with an algorithm with higher resolution value function representation, for example, a locally-weighted regression method, such as *Hedger* (Smart & Kaelbling, 2000). In some cases, however,

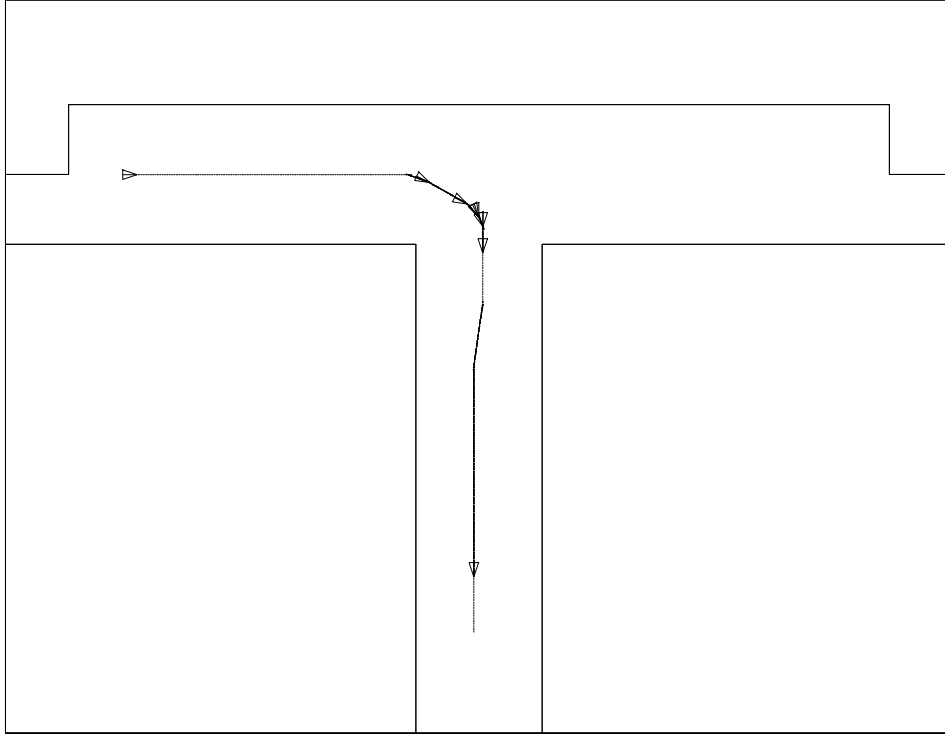


Figure 5: **Navigation using Learned Abstraction.** An example episode after the agent has learned the task using the \mathcal{A}^1 actions. The triangles indicate the state of the robot at the start of each \mathcal{A}^1 action. The sequence of winning features corresponding to these states is [44, 10, 25, 32, 38, 5, 40, 44, 13] (See Figure 6). The narrow line indicates the sequence of \mathcal{A}^0 actions used by the \mathcal{A}^1 actions. Note that in two cases the \mathcal{A}^1 actions carry the robot essentially abstract the concept, “drive down the hall to the next decision point.” In the south corridor, the path turns slightly to the west at the transition from trajectory-following to hill-climbing. Navigating to the goal requires only 9 \mathcal{A}^1 actions, instead of hundreds of \mathcal{A}^0 actions, in other words, task diameter is vastly reduced.

Such disambiguation is not possible. Handling these cases will require adding a new step to the SODA algorithm:

6. Use the exploratory history and abductive methods from the SSH (Kuipers, 2000; Remolina & Kuipers, 2004) to disambiguate perceptually aliased distinctive states associated with each feature in \mathcal{F} , thus further abstracting the world into a larger, but more deterministic, partially-observable MDP (POMDP).

The agent can then learn the value function over the hidden states by standard temporal-difference methods. Implementing and evaluating this extension is a most interesting area of future work.

Another important area of future work is replacing the hard-coded trajectory-following and hill-climbing controllers with learned controllers. The current hill-climbing implementation samples each feature gradient using primitive actions. This process is expensive, and may be impractical with highly noisy or irreversible actions. Likewise, trajectory-following as implemented in these experiments, is simply an open-loop repetition, and may be less reliable outside of simulation. Both HC and TF controllers should be relatively straightforward to learn with reinforcement learning. Each HC controller must act to maximize a particular feature value in a local area; its reward function is straightforward to compute from the feature activation. Similarly, each TF controller must act to keep a feature’s value as high as possible while making progress along some control axis. Although this low-level action learning will lengthen the agent’s overall learning time early on, when an agent must perform many different tasks in the same environment, this cost can be amortized over all asks, and SODA will still provide an improvement over learning each new task from scratch.

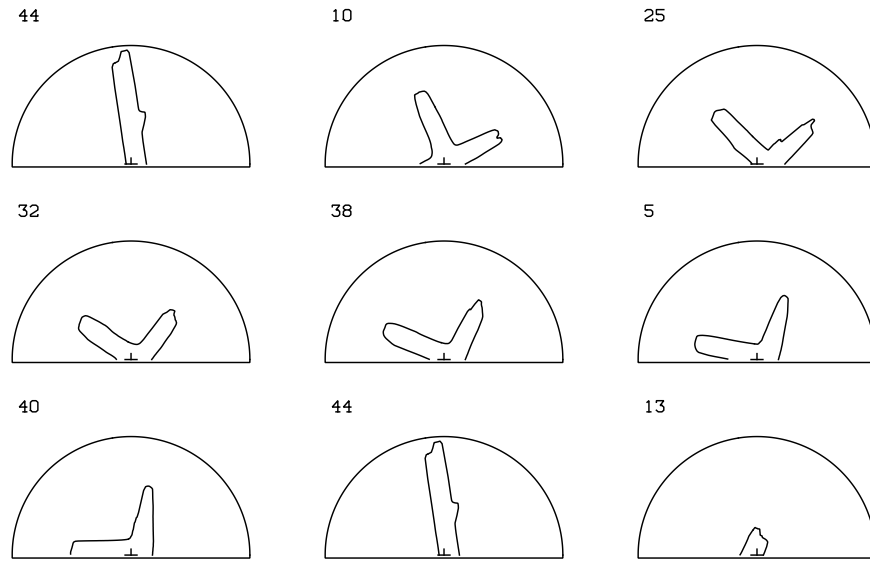


Figure 6: **Features for Distinctive States** The perceptual features for the distinctive states used in the solution shown in Figure 5, in the order they were traversed in the solution. (Read left-to-right, top-to-bottom.)

6 Conclusion

Self-Organizing Distinctive-state Abstraction (SODA) is a method by which a robot in a continuous world builds complementary perceptual and temporal abstractions by first self-organizing a feature map into a set of higher-level perceptual features, and then using those features to build a set of high-level actions that carry it between perceptually distinctive states in the environment. Experiments on a simulated robot navigation task showed that, starting with a robot with uninterpreted sensors and effectors, SODA was able to learn sets of high-level perceptual features, distinctive states, and actions that greatly abstracted and simplified the robot’s model of its environment and its own sensorimotor access to it. Applying reinforcement learning to this abstracted model, the robot was able to learn much more quickly to navigate to its goal.

References

- Chaput, H. H.; Kuipers, B.; and Miikkulainen, R. 2003. Constructivist learning: A neural implementation of the schema mechanism. In *Proceedings of the Workshop on Self-Organizing Maps (WSOM03)*.
- Digney, B. 1998. Learning hierarchical control structure for multiple tasks and changing environments. In *Proceedings of the Fifth Conference on the Simulation of Adaptive Behavior: SAB 98*.
- Duckett, T., and Nehmzow, U. 2000. Performance comparison of landmark recognition systems for navigating mobile robots. In *Proc. 17th National Conf. on Artificial Intelligence (AAAI-2000)*, 826–831. AAAI Press/The MIT Press.
- Fritzke, B. 1995. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*.
- Gerkey, B.; Vaughan, R. T.; and Howard, A. 2003. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics*, 317–323.
- Hengst, B. 2002. Discovering hierarchy in reinforcement learning with hexq. In Sammut, C., and Hoffmann, A., eds., *Machine Learning: Proceedings of the 19th Annual Conference*, 243–250.

- Kohonen, T. 1995. *Self-Organizing Maps*. Berlin: Springer.
- Kuipers, B., and Beeson, P. 2002. Bootstrap learning for place recognition. In *Proc. 18th National Conf. on Artificial Intelligence (AAAI-2002)*, 174–180. AAAI/MIT Press.
- Kuipers, B. 2000. The Spatial Semantic Hierarchy. *Artificial Intelligence* 119:191–233.
- Martinetz, T. M.; Ritter, H.; and Schulten, K. J. 1990. Three-dimensional neural net for learning visuomotor coordination of a robot arm. *IEEE Transactions on Neural Networks* 1:131–136.
- McGovern, A., and Barto, A. G. 2001. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Machine Learning: Proceedings of the 18th Annual Conference*, 361–368.
- Nehmzow, U., and Smithers, T. 1991. Mapbuilding using self-organizing networks in really useful robots. In *Proceedings SAB '91*.
- Pierce, D. M., and Kuipers, B. J. 1997. Map learning with uninterpreted sensors and effectors. *Artificial Intelligence* 92:169–227.
- Provost, J.; Beeson, P.; and Kuipers, B. J. 2001. Toward learning the causal layer of the spatial semantic hierarchy using SOMs. AAAI Spring Symposium Workshop on Learning Grounded Representations.
- Remolina, E., and Kuipers, B. 2004. Towards a general theory of topological maps. *Artificial Intelligence* 152:47–104.
- Ryan, M. R. K. 2002. Using abstract models of behaviours to automatically generate reinforcement learning hierarchies. In *Proceedings of The 19th International Conference on Machine Learning*.
- Smart, W. D., and Kaelbling, L. P. 2000. Practical reinforcement learning in continuous spaces. In *Machine Learning: Proceedings of the 17th Annual Conference*. San Francisco: Kaufmann.
- Smith, A. J. 2002. Applications of the self-organizing map to reinforcement learning. *Neural Networks* 15:1107–1124.
- Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.