Self-Organizing Distinctive State Abstraction Using Options*

Jefferson Provost[1] **Benjamin J. Kuipers**[2] **Risto Miikkulainen**[2]
[1]Center for the Neural Basis of Cognition
and Department of Neuroscience
University of Pittsburgh
Pittsburgh, PA 15260, USA
jp@cnbc.cmu.edu

[2]Artificial Intelligence Lab
The University of Texas at Austin
1 University Station C0500
Austin, TX 78712, USA
{kuipers,risto}@cs.utexas.edu

Abstract

An important problem in developmental robotics is the automatic learning of motor routines or behaviors without human guidance. This paper presents Self-Organizing Distinctive-State Abstraction (SODA), a method by which a robot can learn a set of sensory features and reusable motor routines from raw sensorimotor experience. Experiments show that the learned versions of the motor routines outperform hard-coded alternatives, and that robots using the learned routines can learn tasks much more quickly than when using primitive actions. The features and motor routines are learned autonomously, and reflect only the environment and the agent's sensorimotor capabilities, without external direction.

1. Introduction

An important problem in developmental robotics is the autonomous learning of behaviors without external guidance. Modern robots operate in high-dimensional, continuous sensorimotor spaces defined by rich sensors such as laser rangefinders, and high-resolution motor control that is updated many times per second. Autonomous learning of effective robotic behaviors requires autonomous learning of useful perceptual and motor abstractions for these spaces. This paper presents the Self-Organizing Distinctive-state Abstraction method (SODA) that builds on the concept of the distinctive state from topological mapping literature to define abstractions that can be learned from the agent's own experience, without prior knowledge provided by an external designer.

A robot using SODA develops its abstraction in two stages. First, it learns a set of perceptual prototypes during an initial "motor babbling" phase. At the end of this perceptual learning phase, distinctive states are defined as states that provid a local best match to one of these prototypes. Then, in the second phase the robot learns two types of behaviors that together allow the robot to move between distinctive states: hill-climbing (HC) options, and trajectory-following (TF) options. Once the robot can move reliably between distinctive states, it has developed a new, abstracted state space in which learning long-distance navigation tasks is much easier than in its original raw sensorimotor space.

A preliminary version of SODA was presented by Provost et al. (2006). The current paper grounds SODA rigorously in hierarchical reinforcement learning (Sections 2. and 3.) and provides a detailed evaluation of the approach (Section 4.). Although, the experiments are conducted in the setting of mobile robot navigation, there is nothing in the method that specifically assumes a navigation task, and the basic principles of the method should apply equally well to other configuration spaces, such as the joint-angle space of articulated robots.

2. Background

The sections below explain SODA and the options formalism in detail. Section 2.1 briefly describes SODA. Section 2.2 provides a brief overview of the options formalism for hierarchical RL.

2.1 SODA

SODA's learning method assumes that the learning agent is instantiated in a robot with a continuous sensory input vector $\mathbf{y}$, and a continuous output space spanned by a set of orthogonal basis vectors $\mathcal{U} = \{\mathbf{u}^1, ..., \mathbf{u}^m\}$. A *primitive action*, $\mathbf{a}$, is a vector in this space that is converted to a motor command by multiplying with the motor basis: $\mathbf{u} = [\mathbf{u}^1, ..., \mathbf{u}^m]^T\mathbf{a}$. SODA defines a default set of primitive actions $\mathcal{A}^0$, each consisting of positive and negative unit vectors along each axis of the motor space. For example, in the environment in Section 4., the motor basis consists of one vector to translate the robot and one to rotate, and the primitive actions move the robot forward, backward, left and right. The members of $\mathcal{A}^0$ are referred

*To appear in *7th International Conference on Epigenetic Robotics*, 2007

to as $\mathbf{a}_1, ..., \mathbf{a}_{2m}$.

Learning in SODA operates in two phases. First, the agent explores the environment for a period of time using a random walk over the primitive actions $\mathcal{A}^0$. During this period, it trains a Growing Neural Gas network (GNG) (Fritzke, 1995) with the input vectors $\mathbf{y}$ sampled on the random walk. The GNG is an incremental self-organizing vector-quantizing network that learns a set of prototype vectors to represent the input space. Each prototype $\mathbf{w}_i$ defines a *perceptual neighborhood*, i.e. the region of input space closest to it. The *current* perceptual neighborhood is defined as the neighborhood of the prototype closest to $\mathbf{y}$ at any particular time (called the GNG *winner*). Using the GNG, SODA defines a set of feature activation functions $\mathcal{F}$, where each $f_i(\mathbf{y})$ in $\mathcal{F}$ measures the nearness of $\mathbf{y}$ to $\mathbf{w}_i$.

The set of learned prototypes, along with their activation functions, allow the agent to define a set of distinctive states (Kuipers, 2000) in the environment. The agent can travel between the states using a combination of two types of abstract actions or behaviors, hill-climbing (HC) actions and trajectory-following (TF) actions. HC actions carry the robot up the gradient of the current perceptual feature $f^*(\mathbf{y})$ to a distinctive state from within the state's perceptual neighborhood. TF actions begin at one distinctive state and carry the robot into the neighborhood of another. Using sequences of one TF and one HC action, the agent can move from one distinctive state to another.

In the second phase of learning, the preliminary implementation constructed the TF actions as simple macros that repeated a single action (e.g. move forward) until the agent reached a new perceptual neighborhood. In addition, the HC actions consisted of a hard coded and expensive routine of sampling the feature gradient on each step in order to determine which way to move. This paper redefines both types of actions as options with learned action policies. This improvement creates more reliable, closed-loop TF actions and more efficient HC actions.

2.2 *Behavioral Routines as* Options

SODA defines its TF and HC actions using a formalism from hierarchical reinforcement learning known as *options*. An option is a formal specification of a behavioral routine that can itself be used as an action or operator in a higher-level behavior. Often options are used to define sub-goal behaviors. For example a mobile robot might define an option for reaching the doorway from any point within a room, and a legged robot might have an option for achieving a standing position from sitting or lying down. SODA's hill-climbing options are sub-goal options. Options need not only specify sub-goals, however. Options can also specify behaviors to maintain some current condition or set of conditions for as long as possible, for example a bipedal robot could have an option for remaining standing and balanced. SODA's trajectory-following options are examples of such "condition maintaining" options.

The options formalism (Sutton et al., 1999) is a framework for hierarchical RL in which an agent learns a policy over a set of options $\mathcal{O}$. In a Markov decision process (MDP) with set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, and a reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, each option o_i in $\mathcal{O}$ is a tuple $\langle \mathcal{I}_i, \pi_i, \beta_i \rangle$ where $\mathcal{I}_i \subseteq \mathcal{S}$ is the *initiation set*, that is, the set of states in which o_i may be executed, $\pi_i : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is the action policy of the option, giving the probability of selecting a given action in a given state, and $\beta_i : \mathcal{S} \rightarrow [0, 1]$ is the termination function, giving the probability of the option terminating in a given state. Although there is a wide variety of prior work on automatically discovering options (Şimşek and Barto, 2004; McGovern and Barto, 2001; Hengst, 2002), that work has focused on discrete, atomic or low dimensional domains. This paper develops a method for automatically constructing options in a realistic robotic domain.

When option policies are learned using RL, it is often useful to augment the option with a *pseudo-reward* function R_i, different from the MDP's reward function, to specify the subtask that the option is to accomplish. Further, although, in the original definition of options, the domain of action for each option is the full set of other options, allowing arbitrary hierarchy (primitive actions are defined as one-step options), in cases with very large action sets (such as with continuous actions) it is useful to be able to constrain the actions available to an option to a specific action set $\mathcal{A}_i$. The next section defines $\mathcal{I}_i$,π_i, β_i, R_i, and $\mathcal{A}_i$ for SODA's TF and HC options.

3. Method

This section describes in detail how to implement SODA using options. Section 3.1 describes a small but important change in SODA's perceptual abstraction. Section 3.2 describes the Top-N state representation used by SODA's option controllers. Sections 3.3 – 3.5 formally describe how to define SODA's actions as options.

3.1 *Feature Activation Functions*

SODA defines the feature functions as Gaussian kernels centered on each prototype, with a width defined by the mean distance between adjacent prototypes in the GNG adjacency graph. This feature function provides covers the input space well and increases monotonically as the distance between the input and the respective prototype decreases. These kernels replace the previous design where each feature function returned the normalized inverse distance to the prototype vector, which led to instability and interdependence among feature values.

3.2 *Top-N State Representation*

Trajectory following and hill-climbing are options that operate within an individual perceptual neighborhood. In order to learn policies for them, the learner needs a state representation that provides more resolution than simply

using the winning GNG unit as the state. The options described below are based on a simple new state abstraction derived from the GNG, called the Top-N representation. If $i_1, i_2, ..., i_{|\mathcal{F}|}$ are the indices of the feature functions in $\mathcal{F}$, sorted in decreasing order of the value of $f_i(\mathbf{y})$, then $\mathrm{Top}^n(\mathbf{y}) = \langle i_1, ..., i_n \rangle$. This representation uses the GNG prototypes to create a hierarchical tessellation of the input space starting with the Voronoi tessellation induced by the GNG prototypes. Each Voronoi cell is then subdivided according to the next closest prototype, and those cells by the next, etc. This tuple of integers can be easily hashed into an index into a Q-table for use as a state representation.

3.3 Trajectory Following

Intuitively, TF options are intended to make progress in some direction while maintaining as closely as possible the match with the current prototype. For example, if the prototype is a view looking straight down a corridor, and the progress direction is forward, the TF option is expected to move down the hallway, correcting for deviations to maintain the view looking forward as much as possible.

To this end, SODA defines a new TF option for each combination of primitive actions and prototypes: $\{TF_{ij} | \langle \mathbf{a}_i, f_j \rangle \in \mathcal{A}^0 \times \mathcal{F}\}$. The initiation set of each TF option is the set of states[1] where its prototype is the winner: $\mathcal{I}_{ij}^{\mathrm{TF}} = \{\mathbf{y} | j = \arg\max_k f_k(\mathbf{y})\}$. The option terminates if it leaves its prototype's perceptual neighborhood:

$$\beta_{ij}^{\mathrm{TF}}(\mathbf{y}) = \begin{cases} 0 & \text{if } \mathbf{y} \in \mathcal{I}_{ij}^{\mathrm{TF}} \\ 1 & \text{otherwise.} \end{cases} \quad (1)$$

Each TF option's pseudo-reward function is designed to reward the agent for keeping the current feature value as high as possible for as long as possible, thus:

$$R_{ij}^{\mathrm{TF}}(\mathbf{y}) = \begin{cases} f_i(\mathbf{y}) & \text{if not terminal,} \\ 0 & \text{if terminal.} \end{cases} \quad (2)$$

In order to force the TF options to make progress in some direction (instead of just oscillating in some region of high reward), each option is given a limited action set consisting of a *progress action* selected from $\mathcal{A}^0$, plus a set of corrective actions formed by adding a small component of each orthogonal action in $\mathcal{A}^0$:

$$\mathcal{A}_{ij}^{\mathrm{TF}} = \{\mathbf{a}_i\} \cup \{\mathbf{a}_i + c_{\mathrm{TF}} \mathbf{a}_k | \mathbf{a}_k \in \mathcal{A}^0, \mathbf{a}_k^T \mathbf{a}_i = 0\}. \quad (3)$$

Lastly, the option policy π_{ij}^{TF} is learned using tabular Sarsa(λ), using the $\mathrm{Top}^n(\mathbf{y})$ state representation described in Section 3.2, and the actions $\mathcal{A}_{ij}^{\mathrm{TF}}$.

3.4 Hill-climbing

With the hard-coded macros in the earlier implementation, SODA had only a single hill-climbing controller that worked in all perceptual neighborhoods. In contrast, when hill-climbing is learned, each feature presents a different pseudo-reward function, and thus requires a separate HC option, HC_i, for each f_i in $\mathcal{F}$.

As with the TF options, the initiation set of each HC option is the perceptual neighborhood of that option's corresponding GNG prototype: $\mathcal{I}_i^{\mathrm{HC}} = \{\mathbf{y} | \arg\max_j f_j(\mathbf{y}) = i\}$. Termination, however, is more complicated for hill-climbing. The perceptual input is unlikely to ever match any prototype exactly, so the maximum feature value attainable in any neighborhood will be some value less than 1. Because it is difficult or impossible to know this value in advance, the stopping criterion is not easily expressed as a function of the single-step input. Instead, the options use a k-Markov termination function, in which the option terminates if the average step-to-step change in the feature value over a finite moving window falls below a small fixed threshold:

$$\beta_i^{\mathrm{HC}}(\mathbf{y}_{t-c_w}, ..., \mathbf{y}_t) = \left\lceil c_{stop} - \frac{\sum_{i=0}^{k-1} |\Delta^{t-k} f_i(\mathbf{y})|}{c_w} \right\rceil, \quad (4)$$

where c_w is the window size, c_{stop} is a constant threshold and $\Delta^t f_i(\mathbf{y}) = f_i(\mathbf{y}_t) - f_i(\mathbf{y}_{t-1})$ computes the change in feature value at time t. The task of the HC option is to climb the gradient of its feature as quickly as possible, and terminate at the local maximum of the feature value. On nonterminal steps the pseudo-reward for each HC option is a shaping function (Ng et al., 1999) consisting of a multiple of the one-step change in f_i, minus a small penalty for taking a step; on terminal steps, the reward is simply f_i:

$$R_i^{\mathrm{HC}} = \begin{cases} c_{R1} \Delta f_i(\mathbf{y}) - c_{R2} & \text{if not terminal,} \\ f_i(\mathbf{y}) & \text{if terminal.} \end{cases} \quad (5)$$

Finally, the action set for HC options is just the set of primitive actions: $\mathcal{A}_i^{\mathrm{HC}} = \mathcal{A}^0$.

3.5 TF+HC Actions

As in the previous SODA implementation, the agent's high-level navigation policy chooses from a set $\mathcal{A}^1$ of TF+HC pair options. There is one such option for each TF option. When selected, each $\mathcal{A}^1$ option executes its corresponding TF option, and when trajectory-following terminates, immediately executes whatever HC option is applicable. Each $\mathcal{A}^1$ option's initiation set is identical to that of its corresponding TF option.

4. Experiments and Results

Below are the results of four experiments showing that the new TF and HC options improve SODA's performance in an environment with realistic sensor and motor noise. The first two experiments examine the performance of the new TF and HC options in isolation, comparing them to the hard-coded macros used in the

[1] To simplify the terminology, these descriptions refer to the input vector $\mathbf{y}$ as if it were the state s. Since $\mathbf{y}$ is a function of s, this terminology is sufficient to specify the options.

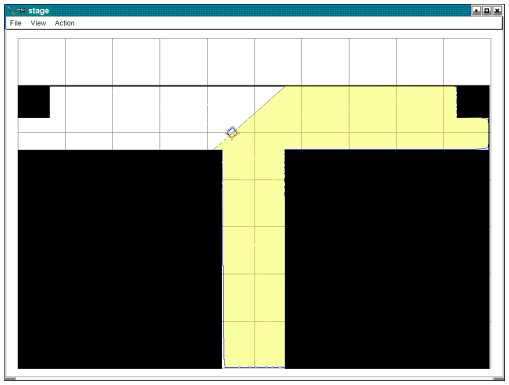


Figure 1: **Simulated Robot Environment**. A screen-shot of the Stage robot simulator with the simulated robot and experimental environment. The robot is in the intersection, facing Southeast, and the shaded region indicates the area swept by its laser-rangefinder. The robot simulates sensor and motor noise by perturbing the input signals from the laser and the output signals to the motors.

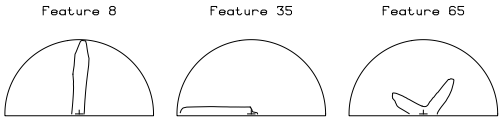


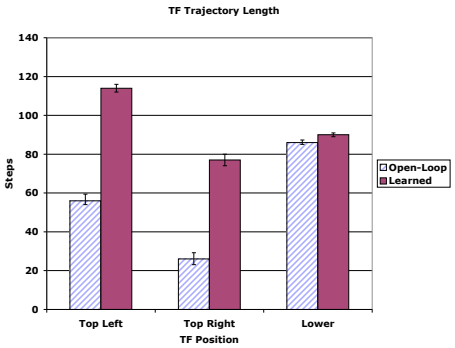
Figure 2: **Example Learned Prototypes**. These plots show three prototypical sensor images learned from the environment shown in Figure 1. Each plot represents a range image from the laser rangefinder, plotted in polar coordinates with the robot at the origin, facing up the Y-axis. The semi-circular boundary indicates the maximum range of the rangefinder (8 meters). Feature 8 represents the view seen when facing down the long upper hallway from either end. Feature 35 represents the view seen when turned 90°to the right from Feature 8. Feature 65 represents a view seen in the middle of the intersection.

previous implementation. The third and fourth experiments show that the new options improve performance on robot navigation tasks as compared to primitive actions and macros.

4.1 Experimental Setup

The simulated robot and environment for these experiments replicates the robot and environment used in the original SODA experiments, plus sensor and motor noise. The simulations were performed in the Stage robot simulator (Gerkey et al., 2003). The environment is a 10 m $\times$ 6 m T-shaped room, called the T-Maze, shown in Figure 1.

The robot was equipped with a laser range finder reading 180 readings at 1° intervals over the forward semicircle around the robot, with a maximum range of 8000 mm. The Stage laser rangefinder model returns ranges with an accuracy of 1 mm, with no noise. Sensor noise was simulated through a two-stage process of alternately adding



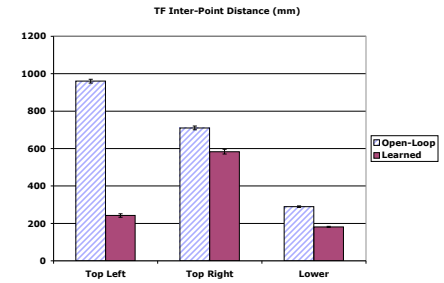


Figure 3: **Trajectory lengths and endpoint clustering**. *Top:* The average trajectory lengths for open loop vs. learned trajectory-following. Learned TF options produce longer trajectories. *Bottom:* Average inter-point distance for the trajectory end points, the learned trajectory endpoints are better clustered (see Figure 4). (Error bars indicate +/- one standard error.)

Gaussian error and rounding, applied individually to each range reading. This model provides a good characterization of the error on a SICK LMS laser rangefinder.

The robot was also equipped with a differential-drive base that takes two continuous control values, linear velocity v and angular velocity ω. The Stage simulator does not model positional error internally, so motor noise was simulated by perturbing the motor commands by

$$\hat{v} = \mathcal{N}(v, k_{vv}v + k_{v\omega}\omega)\ , \hat{\omega} = \mathcal{N}(\omega, k_{\omega v}v + k_{\omega\omega}\omega), \tag{6}$$

where $\hat{v}$ and $\hat{\omega}$ are the noisy motor command and the constants used were $k_{vv} = 0.1$, $k_{v\omega} = 0.1$, $k_{\omega v} = 0.2$, $k_{\omega\omega} = 0.1$. Equation (6) is a simplified motor noise model, inspired by realistic models used in robot localization and mapping (Roy and Thrun, 1999; Beeson et al., 2006). The agent's motor basis set (Section 2.1) $\mathcal{U}$ consisted of two motor basis vectors, $\mathbf{u}^1 =$ [250 mm/sec, 0°/ sec] and $\mathbf{u}^2 =$ [0 mm/sec, 20°/sec]. The robot accepts motor commands from the agent 10 times per second. In all the experiments below, the agent had already explored the environment and trained a prototype set. Examples of three learned prototypes are shown in Figure 2.

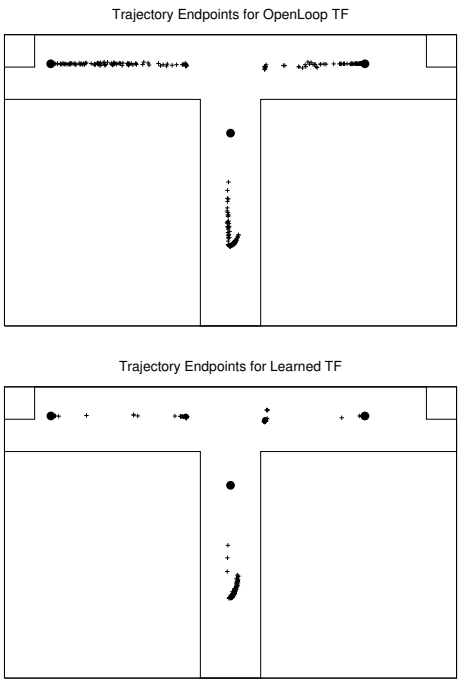


Figure 4: **Trajectory Following Improves with Learning**. The experiment in Section 4.2 trained the "trajectory-follow forward" option starting from each of the three locations marked with circles above and compared the results with open-loop TF from the same locations. *Top:* The endpoints of 100 open-loop TF runs from each location. *Bottom:* The endpoints of the last 100 runs of the learned options. The TF learned using RL are much better clustered, indicating much more reliable travel.

4.2 Trajectory-Following Experiment

The trajectory-following experiments compared the reliability of open-loop TF macros and learned TF options, by testing the agent's ability to learn to follow a trajectory forward down each of the three corridors of the T-Maze. The option's action set (see Equation 3) consisted of the progress action (moving straight forward), $[1, 0]^T$, and two corrective actions (moving forward while turning left or right) $[1, 0.1]^T$ and $[1, -0.1]^T$. The option was trained for 2000 episodes from each starting point, although in each case the behavior converged within 100–400 episodes. The Sarsa(λ) parameters were: $\lambda = 0.9, \alpha = 0.1, \gamma = 1.0$, the option used ϵ-greedy action selection with $\epsilon_0 = 1.0$ and annealing down to $\epsilon_\infty = 0.001$ with a half-life of 400 steps. All Q values were initialized to 0, and the state representation used the top three winners from the GNG.

Figure 4 shows the ending points of the last 100 runs from each starting point, compared with the ending points for 100 runs using the open-loop TF macro. For each of the three starting points, the endpoints are more tightly clustered when using the learned option. This is because there are many fewer episodes where the trajectory terminates part of the way down the hall due to motor noise pushing the robot off of its trajectory and into a new perceptual neighborhood. As a result, the learned option produces longer trajectories more reliably, and the endpoints tend to be near one another.

The top panel of Figure 3 shows the average lengths (in steps) of the last 100 runs of the learned TF option, compared with 100 runs from the open-loop macro. From all three starting points, the learned option produces longer runs; from the two starting points in the top corridor, the average is dramatically longer. The bottom panel shows the average inter-point distance between trajectory endpoints for each corridor; in all cases the trajectory endpoints are significantly closer together.

4.3 Hill-Climbing Experiment

The hill-climbing experiments compared the speed and effectiveness of the learned HC options with the hard-coded HC routine described in Section 2.1. Each experiment tested the agent's ability to hill-climb on the gradient one of three feature from the GNG (shown in Figure 2). The goal of hill-climbing is for the option to increase the feature value as much as possible as quickly as possible.

Each experiment consisted of 2000 episodes in which the robot was placed at a randomly selected pose in the perceptual neighborhood of the given feature and the hill-climbing option (or macro) was initiated from that point. The option's Sarsa(λ) parameters were $\lambda = 0.9$, $\alpha = 0.1$, $\gamma = 0.997$. The agent did not use ϵ-greedy action selection, but all Q-values were initialized optimistically to 1.0 to encourage exploration. The HC option parameters (see Section 3.4) were $c_{stop} = 0.005$, $c_w = 10$, $c_{R1} = 10$, $c_{R2} = 0.001$.

Figure 5 compares the episode length and final feature value achieved for each option. Runs were performed using the top 3, 4, and 5 GNG features as the state representation. There was no significant difference in their performance, so only the $n = 4$ runs are shown. The learned HC options were able to achieve as high or higher activation than the hard-coded HC macro, using many fewer steps per episode.

4.4 Navigation Experiments

The final two experiments examined the effect of the new learned options on the agent's learning performance in large-scale navigation.

The first of these experiments replicated the original SODA experiment, in which the agent must learn to navigate from a point in the upper left corner of the T-Maze environment to a 1-meter-wide area at the bottom of the central corridor. In the previous SODA paper, Provost et al. (2006) noted that SODA abstraction induces perceptual aliasing in the environment, and that such aliasing may make it difficult to learn to navigate. Because the focus of the current work is specifically on learning the TF and HC options, the agents were given more sensory information to help reduce aliasing without resorting

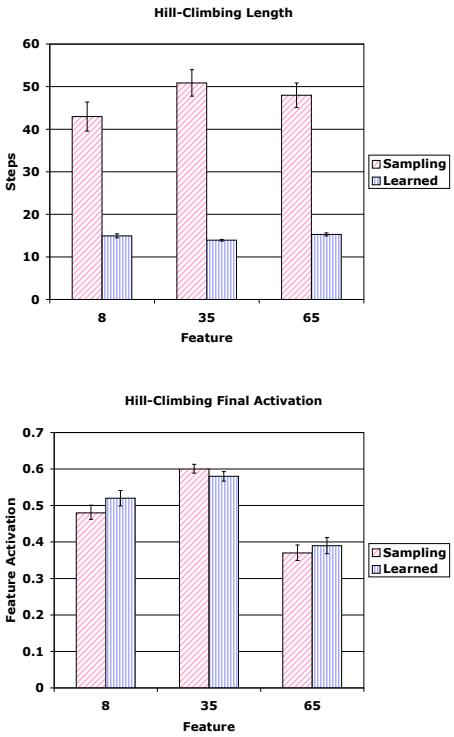


Figure 5: **Effect of learning on hill-climbing.** Using learned options makes hill-climbing achieve the same feature values faster. *Top:* The average lengths of hill-climbing episodes in the neighborhoods of the three different features shown in Figure 2. *Bottom* The average maximum feature value achieved for each prototype per episode. The plots compare the last 100 HC episodes for each feature with 100 hard-coded HC runs. For each feature, the maximum value achieved is comparable, but the number of actions needed is much smaller. (Error bars indicate +/- one standard error.)

to more complicated representations for partially observable environments. The two new sensors are a stall warning to indicate collisions, and an eight-point compass.

The learning agent's state representation is the tuple $\langle$stall,compass,$\langle \text{argmax}_j f_j \in \mathcal{F} \rangle \rangle$. As with the Top-N representation, this tuple is hashed into an index into the Q-table. The reward function for the task gives a reward of 0 for reaching the goal, -1 for taking a step, -6 for stalling. Each episode times out after 10,000 steps. The experiment ran 150 total trials of 1000 episodes: five trials using each of 10 different trained GNG networks using primitive actions, hard-coded macros, and using learned options. The Sarsa(λ) parameters were $\lambda = 0.9$, $\alpha = 0.1, \gamma = 1.0$. All Q values were initialized optimistically to 0. The agent also used ϵ-greedy action selection with $\epsilon_0 = 0.1$ annealing to $\epsilon_\infty = 0.01$ with a half-life of 100,000 (primitive) steps. The experiment began with all options untrained, and option policy learning proceeded concurrently with high-level policy learning. Figure 6 shows the learning curves comparing the performance of the agents using macros to the agents using learned options. Both the agents using macros and the agents using options learn dramatically faster than the agents using primitive actions. In addition, the behavior of the agents using the learned options converges faster than that of the agents using macros, and to a lower asymptote.

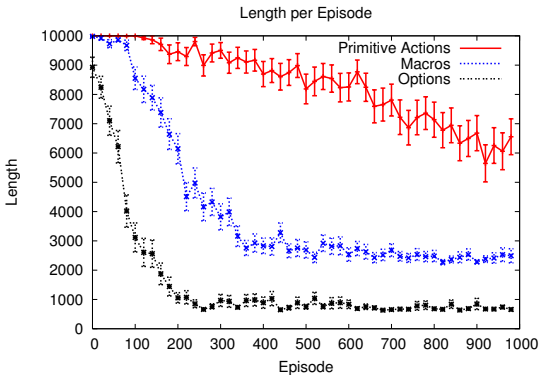


Figure 6: **Navigation Learning with Learned Options**. This learning curve shows the number of steps taken per episode while learning to navigate from the upper left to the bottom of the environment in Figure 1. The upper curve represents SODA agents using open-loop trajectory-following and sample-based hill-climbing, the lower curve represents agents using learned TF and HC options. (Error bars represent +/- one standard error.) The agents using learned options learn the task faster and converge to a shorter solution than those using macros, and vastly outperform the agents using primitive actions.

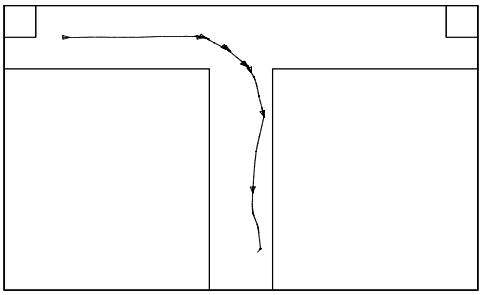


Figure 7: **Navigation using SODA**. This trace shows the path of a robot navigating in the T-maze environment using SODA. The triangles show the starting points of the learned high-level actions. The narrow line shows the path of the robot using low-level motor commands. A path that required hundreds of actions in the lower, pixel-level representation requires only ten in the high-level action-based representation.

The second navigation experiment tested the ability of SODA using options to scale up to larger environments. This experiment was performed in the environment shown in Figure 8, a simulation of the fourth floor of the ACES building at the University of Texas, constructed using simultaneous localization and mapping with a robot similar in configuration to the simulated robot in these experiments. With a length and width of approximately 40 m, this environment is several times the size of the T-maze. Furthermore, the outer hallways are too long and narrow to traverse using open-loop trajectory following

under motor noise. The navigation task in the ACES environment was to navigate from the center-right intersection to the lower-left corner of the environment.

The experimental set-up and agent parameters were identical to those in the T-Maze experiments, with the following exceptions. Because of the larger size of the environment, the agent was allowed to explore the environment with a random walk over trajectory-following options, allowing the agent to cover a larger area than a random walk over primitive actions. In this case, the GNG was trained concurrently with the training of the TF option policies. However, for experimental clarity, the learned TF policies were discarded after the feature sets were learned, then the TF options were trained anew during the navigation experiment. In addition, the forward speed in the abstract motor interface (see Section 4.1) was increased from 250 mm/sec to 500 mm/sec, and the time-out for each episode was increased to 30,000 time-steps. Finally, because the horizon of the robot's rangefinder is much less than the length of a typical corridor, many corridors and intersections are perceptually indistinguishable. To deal with problem, this the agent was given extra state variables consisting of the eight-point compass and bump sensor used in the T-Maze experiment, plus a coarse tiling of the robot's x/y position into 10 m $\times$ 10 m squares. This tiling can be seen as roughly equivalent to having large sections of the building marked with different colors that are visible to the robot.

Figure 9 shows the learning curves for the ACES environment. Each curve averages the performance of 50 agents — 5 runs using each of 10 trained GNGs — for agents using SODA's TF+HC options and agents using primitive actions only. The curves show that the SODA agents are able to learn the navigation task within the alloted time-out period, while the agents using primitive actions are not.

5. Discussion and Related Work

These experiments show that SODA's abstraction allows robotic agents to learn to perform navigation tasks that are difficult or impossible to learn without the abstraction. The abstraction makes it possible to learn to navigate in noisy, high-diameter environments from high-dimensional sensor data with little prior knowledge added by engineers.

SODA's learned TF and HC options each contribute to the power of the abstraction. The TF options allow the agent to traverse long, uniform segments of the environment reliably in a single action, even when perturbed by motor noise. The HC options allow the agent to localize itself at a distinctive state efficiently and autonomously without the need to manually sample the feature gradient to determine which way to go. In this case, each HC option policy can be thought of as a learned model of its respective feature gradient.

This work makes an important contribution to reinforcement learning, by showing how options can be

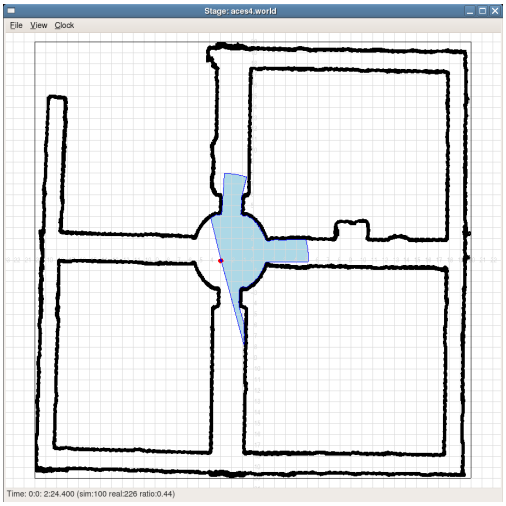


Figure 8: **Larger Robot Environment – ACES**. A screen-shot of the Stage robot simulator with the simulated robot and ACES experimental environment used for the larger navigation experiment. The small circle represents the robot and the shaded region indicates the area swept by its laser rangefinder. This environment is approximately 40 m $\times$ 40 m. The larger size and richer set of perceptual situations allows testing of SODA's ability to scale to realistic environments.

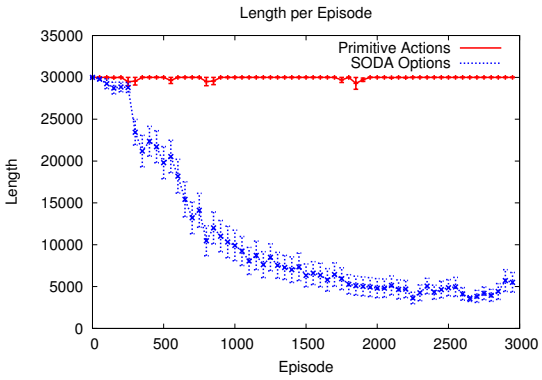


Figure 9: **ACES Navigation Learning Curve** This learning curve shows the number of steps taken per episode while learning to navigate from the center-right intersection to the bottom-left corner of the ACES environment in Figure 8. The upper curve represents agents using primitive actions only, the lower curve represents agents using learned TF and HC options. (Error bars represent +/- one standard error.) SODA agents are able to learn the task while the agents using primitive actions are unable to make reasonable progress on the task in the allotted time.

extended to realistic continuous domains in robotics, through both the formalization of TF and HC macros as options, and through the Top-N state representation. Although there are a variety of other methods for RL in continuous state spaces (Santamaria et al., 1998; Atkeson et al., 1997; Fernández and Borrajo, 2000), these methods each require their own state representations, entailing special data structures and procedures and often including search in large instance databases, and it is not clear how these methods can be scaled to the high-dimensional sensory data for which SODA is designed. The Top-N state representation on the other hand, is simple to compute from SODA's perceptual features and, through hashing, just "plugs in" to existing table-based RL methods. It is interesting to note that, as used here, the Top-N representation does not exploit the hierarchical way in which it divides the state space. It seems likely that this hierarchy could be utilized to generalize better to nearby places in the state space.

Another contribution of SODA as a hierarchical RL method is the introduction of trajectory-following options. Previous work on discovering options focuses on finding sub-goals, i.e., discovering useful states and constructing options that achieve those states as quickly as possible. TF options are essentially the dual of sub-goal options: They try to maintain the current state for as long as possible, while their progress is constrained in some way. TF options are subtasks that do not not fit cleanly into the category of sub-goal discovery, but that are useful to have in the repertoire of an intelligent agent. Generalizing such actions beyond navigation is an interesting direction for future work.

Finally, the agents' use of extra state information to disambiguate percepually aliased states while navigating (Section 4.4) may seem to run counter to the stated goal of removing the need for human-engineered perceptual features. Resolving perceptual aliasing is a well-studied problem for which many methods have been developed (see the survey by Shani (2004)). SODA's philosophy is to focus on learning a perceptual-motor abstraction that is suitable for use with existing methods for disambiguating aliased states. Removing the extra state information and replacing Sarsa(λ) with a learning method that handles hidden state is future work.

6. Conclusion

This paper has shown how SODA can be used to learn a useful perceptual and action abstraction in robot navigation problems with rich, high-dimensional sensors. SODA's trajectory-following and hill-climbing macros are formalized as learnable options. Part of this formalization includes a new state abstraction, the Top-N representation, that is based on SODA's learned set of perceptual prototypes, but provides the higher resolution necessary for local closed-loop control. These new learned options make navigation using SODA more reliable and efficient, reducing learning times and improving learned performance, and allowing robots to learn to navigate in large spaces entirely autonomously, without a pre-engineered model of the environment or the robot's sensorimotor system.

References

Atkeson, C. G., Moore, A. W., and Schaal, S. (1997). Locally weighted learning for control. *Artificial Intelligence Review*, 11(1/5):75–113.

Beeson, P., Murarka, A., and Kuipers, B. (2006). Adapting proposal distributions for accurate, efficient mobile robot localization. In *IEEE International Conference on Robotics and Automation*.

Fernández, F. and Borrajo, D. (2000). VQQL: Applying vector quantization to reinforcement learning. In Veloso, M., Pagello, E., and Kitano, H., (Eds.), *In Lecture Notes in Artificial Intelligence 1856*. Springer.

Fritzke, B. (1995). A growing neural gas network learns topologies. In Tesauro, G., Touretzky, D. S., and Leen, T. K., (Eds.), *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press.

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*, pages 317–323, Coimbra, Portugal.

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*, pages 243–250.

Kuipers, B. (2000). The Spatial Semantic Hierarchy. *Artificial Intelligence*, 119:191–233.

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*, pages 361–368.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: theory and application to reward shaping. In *Proc. 16th International Conf. on Machine Learning*, pages 278–287. Morgan Kaufmann, San Francisco, CA.

Provost, J., Kuipers, B. J., and Miikkulainen, R. (2006). Developing navigation behavior through self-organizing distinctive-state abstraction. *Connection Science*, 18.2. In press.

Roy, N. and Thrun, S. (1999). Online self calibration for mobile robots. In *IEEE International Conference on Robotics and Automation*.

Santamaria, J., Sutton, R., and Ram, A. (1998). Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6(2).

Shani, G. (2004). A survey of model-based and model-free methods for resolving perceptual aliasing. Technical Report 05-02, Department of Computer Science at the Ben-Gurion University in the Negev.

Şimşek, Ö. and Barto, A. G. (2004). Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 751–758. ACM Press.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and SMDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211.