Self-Organizing Perceptual and Temporal Abstraction for Robot Reinforcement Learning

Jefferson Provost, Benjamin J. Kuipers, and Risto Miikkulainen

Artificial Intelligence Lab
The University of Texas at Austin
Austin, TX 78712 USA
{jp|kuipers|risto}@cs.utexas.edu

Abstract

A major current challenge in reinforcement learning research is to extend methods that work well on discrete, short-range, low-dimensional problems to continuous, high-diameter, high-dimensional problems, such as robot navigation using high-resolution sensors. We present a method whereby an robot in a continuous world can, with little prior knowledge of its sensorimotor system, environment, and task, improve task learning by first using a self-organizing feature map to develop a set of higher-level perceptual features while exploring using primitive, local actions. Then using those features, the agent can build a set of high-level actions that carry it between *perceptually distinctive states* in the environment. This method combines a *perceptual abstraction* of the agent's sensory input into useful perceptual features, and a *temporal abstraction* of the agent's motor output into extended, high-level actions, thus reducing both the dimensionality and the diameter of the task. An experiment on a simulated robot navigation task shows that the agent using this method can learn to perform a task requiring 300 small-scale, local actions using as few as 7 temporally-extended, abstract actions, significantly improving learning time.

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 solution requires a large number of primitive actions by the robot, as in, for example, navigating to distant locations using primitive motor control commands. Reinforcement learning (RL) methods that model the world as a Markov decision process (MDP) have shown promise as a method for automatic learning of robot behavior, but extending these methods to high-dimensional, continuous, high-diameter problems still 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. We feel, however, that a major goal of AI research is to create autonomous learning agents, and that one of the requirements of autonomy is that the "hard part" of learning ultimately be performed by the agent, rather than the human engineer. To this end we seek to reduce the amount of prior knowledge needed by the agent.

We claim that a robot in a continuous world can, with little prior knowledge of its sensorimotor system, environment, and task, improve task learning by first using a self-organizing feature map to develop a set of higher level perceptual features while exploring using primitive, local actions. Then using those features the agent can build a set of high-level actions that carry it between *perceptually distinctive states* in the environment. This method combines a *perceptual abstraction* of the agent's sensory input into useful perceptual features, with a *temporal abstraction* of the agent's motor output into extended, high-level actions, thus reducing both the dimensionality and the diameter of the task.

In the remainder of this paper we describe the method, and an experiment that demonstrates the method on a simulated robot navigation task.

Method

Given a robot with high-dimensional, continuous sensations, continuous actions, and a reinforcement signal for a high-diameter task, the agent's learning process consists of the following steps:

Define Primitive Actions – First the agent defines a set of discrete, short-range, local actions to act as the primitive motor operations. These actions can either be defined as a fixed discretization of a learnable abstract motor interface consisting of a set of "principal motor components," as is done in our current experiments, or learned, e.g. using a self-organizing feature map (Kohonen, 1995), in which each feature map unit represents a local motor command, as is done by Smith (2002).

Learn High-level Perceptual Features – Next the agent explores using the primitive actions, and feeds the observed sensations to a self-organizing feature map that learns a set of high-level percepts, in the form of prototypical sensory impressions.

Define High-level Actions – Using these new features, the agent defines *perceptually distinctive states* as points in the robot's state space that are the local best match for some perceptual feature, and creates actions that carry it from one distinctive state to another. The actions are compositions of trajectory-following control laws, that carry the agent into the neighborhood of a new distinctive state, and hill-climbing control laws, that climb the gradient of

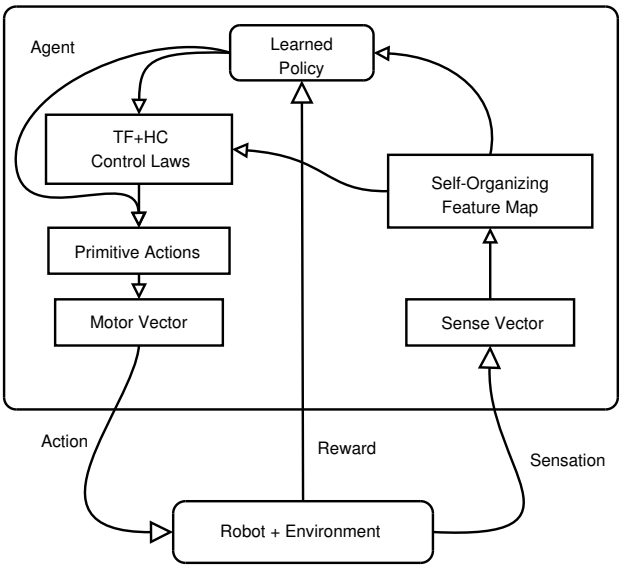

Figure 1: **The Architecture of the Learning Agent.** The learning agent receives a continuous sensory vector from the environment (via the robot), from which a SOM learns a set of perceptual features. These features and the reward signal are fed as input to a reinforcement learning algorithm that learns a policy for generating actions. Each high-level action consists of a *trajectory-following* and *hill-climbing* control law pair that uses primitive actions to carry the robot between *perceptually distinctive states*.

a perceptual feature to a distinctive state.

Learn Tasks – Using the new state and temporal abstraction, the agent attempts to learn its task using a standard RL method such as *SARSA* (Sutton & Barto, 1998), adapted to use temporally-extended actions (Sutton, Precup, & Singh, 1999).

Figure 1 shows the architecture of the learning agent. Below we describe in more detail the two key steps: learning the state abstraction, and using it to construct the temporal abstraction.

Self-Organizing Perceptual Features

Given a set of primitive actions, the agent can begin to act, and receive a stream of sense vectors. The agent constructs features by using the sensory input as training data for a 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 or cells arranged in a lattice.[1] The SOM takes a continuous-valued vector $\mathbf{x}$ as input and returns one of its units as the output. Each unit 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 $\arg\min_i \|\mathbf{x} - \mathbf{w}_i\|$.

In training, the weight vectors in the SOM are initialized to random values. When an input vector $\mathbf{x}_t$ is presented, the winning unit's weight vector is moved toward the input by some fraction of the distance between them, and units in the topological neighborhood of the winning unit in the lattice are moved toward the input by lesser amounts.

Training begins with a large topological neighborhood size. As training proceeds, learning rate and neighborhood are gradually annealed to very small values. As a result, early training orients the map to cover the gross topology of the input space, and as the parameters are annealed, finer grained structure of the input space emerges.

SOMs have several properties that lend themselves well to our feature learning task:

- They are *data- and sensor-general*. Because they operate on any input that can be expressed in the form of a vector, SOMs are not specific to any particular kinds of sensor or environment, making them especially well suited to learning autonomously in a continuous environment.
- A SOM can be trained incrementally, with training vectors presented on-line as they are received during robot exploration. The implementation described below uses a variant on the standard SOM algorithm called the Growing Neural Gas (GNG) (Fritzke, 1995), that begins with a minimal lattice and inserts nodes where the input distribution is densest to minimize distortion error. Unlike the standard SOM, the GNG does not anneal its parameters, and thus is able to continue learning indefinitely, and track changing input distributions. This property makes the GNG especially suitable for robot learning, since a robot experiences its perceptual space sequentially, and may experience entirely new regions of the input space after an indeterminate period of exploration.
- Most importantly, A SOM provides both a coarse-grained discretization of its input space, through its *winner*, the closest prototype to the input, and a set of continuous features, or "activations," defined in terms of the distance of the input from each unit. This divides 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, allowing the agent to construct actions, as described below.

Constructing High-level Actions

To create high-level actions, the agent uses the the abstraction from the *control* and *causal* levels of the *Spatial Semantic Hierarchy* (SSH), a theory of navigation in 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, i.e. the location with maximal SOM-unit response. At the causal level, the agent defines a set of high-level actions each consisting of a TC/HF control-law pair that carries the robot to a new distinctive state. Our current experiments use fixed, hard-coded policies for trajectory following and hill climbing, and are not themselves learned, though in general these behaviors are sequential decision tasks with delayed reward, and could themselves be learned with RL. In that context, the control-laws and actions of the SSH are temporally-extended actions that can be represented as *Options* (Sutton, Precup, & Singh,

[1] The lattice is often, but not necessarily, a 2D rectangular grid.

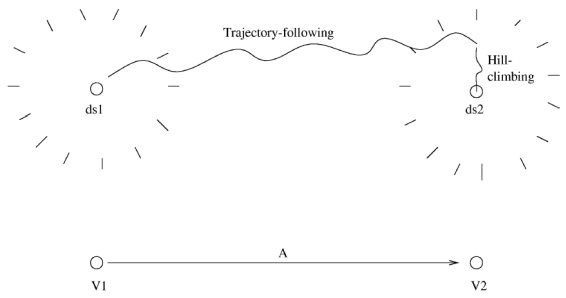

Figure 2: **High-level Actions.** The agent travels from one distinctive state to another using an action with two parts: first a *trajectory-following* controller drives the robot into the neighborhood of a new sensory prototype, then a *hill-climbing* controller to the local state that best matches that prototype.

1999) in a semi-Markov decision process (SMDP).

Related Work

The SOM has been used previously for learning perceptual features or state representations in general robotics, for many of the reasons mentioned above, see for example, Martinetz, Ritter, & Schulten (1990); Duckett & Nehmzow (2000); Nehmzow & Smithers (1991); Nehmzow, Smithers, & Hallam (1991); Provost, Beeson, & Kuipers (2001).

There has also been research into automatically discovering high-level actions for reinforcement learning. Nested Q-Learning (Digney, 1998, 1996) builds a hierarchy of behaviors implemented as learned sub-controllers similar to options (see above). It operates either by proposing every discrete feature value as a subgoal and learning a controller for each, or by keeping track of states that are frequently visited or that have a steep reward gradient and proposing those states as subgoals. The former method requires a relatively small set of discrete features to be tractable. Digney intended the latter version to overcome this problem although it was only tested in a very small, discrete grid-world. The work of McGovern (2002) and McGovern & Barto (2001) is similar in many respects to the work of Digney. It selects states as subgoals using a statistic called "diverse density" to discover states that occur more frequently in successful trials of a behavior than unsuccessful trials, and creates new high-level actions to achieve these states as subgoals. This method differs from Nested Q-Learning in that it is able to use negative evidence (that a state is not on the path to the goal, and thus not likely to be a subgoal). The HEXQ algorithm (Hengst, 2002) attempts to concurrently discover state and temporal abstraction in MDPs with a discrete, factored state vector. It relies on the assumption that some features change more slowly than others, and defines abstract subregions of the state space in which these features remain constant. It also defines abstract actions that take the agent between these regions.

To be used in a continous state space, the temporal abstraction methods above all assume that a continuous-to-discrete abstraction already exists, and they search for higher-level temporal abstractions in the (already abstracted) discrete MDP. The method in this paper 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. The above methods and ours are not, however, mutually exclusive. In very large problems it is likely that multiple-levels of abstraction will be needed. It may be possible to use the methods above to perform additional temporal abstraction on top of the continuous-to-discrete abstraction provided by our method.

action	v	ω	resulting step
ahead	250 mm/sec	0°/sec	25 mm
back	-250 mm/sec	0°/sec	- 25 mm
left	0 mm/sec	90°/sec	9°
right	0 mm/sec	-90°/sec	9°

Table 1: **Primitive Actions.** The learning agent has 4 primitive action, representing positive and negative steps along each axis of motor control.

Preliminary Experiments

In our preliminary experiments, we have tested the system on a small robot navigation task, comparing the ability of the robot to learn the task using the high-level actions against its ability to learn the task using primitive actions. The agent used the same self-organized feature set in both cases.

Robot, Environment, and Task

The experiments were run using the Stage robot simulator (Gerkey, Vaughan, & Howard, 2003). The environment was a simple T-shaped room or maze, shown in Figure 5, measuring $10m \times 6m$. The simulated robot consisted of a simple base with commands to set the linear velocity, v in mm/second and angular velocity ω in degrees/second. Stage does not simulate acceleration, velocity change is instantaneous. The robot's sensation came from a single simulated SICK LMS laser rangefinder, providing 180 range readings over the forward semicircle, with a maximum range of 8 meters. The simulator accepted motor commands and provided sensations every $100ms$ (simulator time).

The agent's task was to drive from the upper left corner to the bottom of the center hallway. The task terminated when the robot reached within approximately $500mm$ of the goal (a point $500mm$ from the end of the lower corridor), or after 2000 simulator steps, whichever came first. The reward on each non-terminal step was 0 unless the robot collided with a wall, in which case it was -5. The terminal reward was graded by the robot's progress toward the goal, according to this formula:

$$r = 0.1(d_0 - d_f) \quad (1)$$

where d_0 and d_f are the initial and final manhattan distances to the goal in mm, respectively.

Features and Actions

The agent had four primitive actions consisting of positive and negative steps along each axis of motor control. The actions are described shown in Table 1

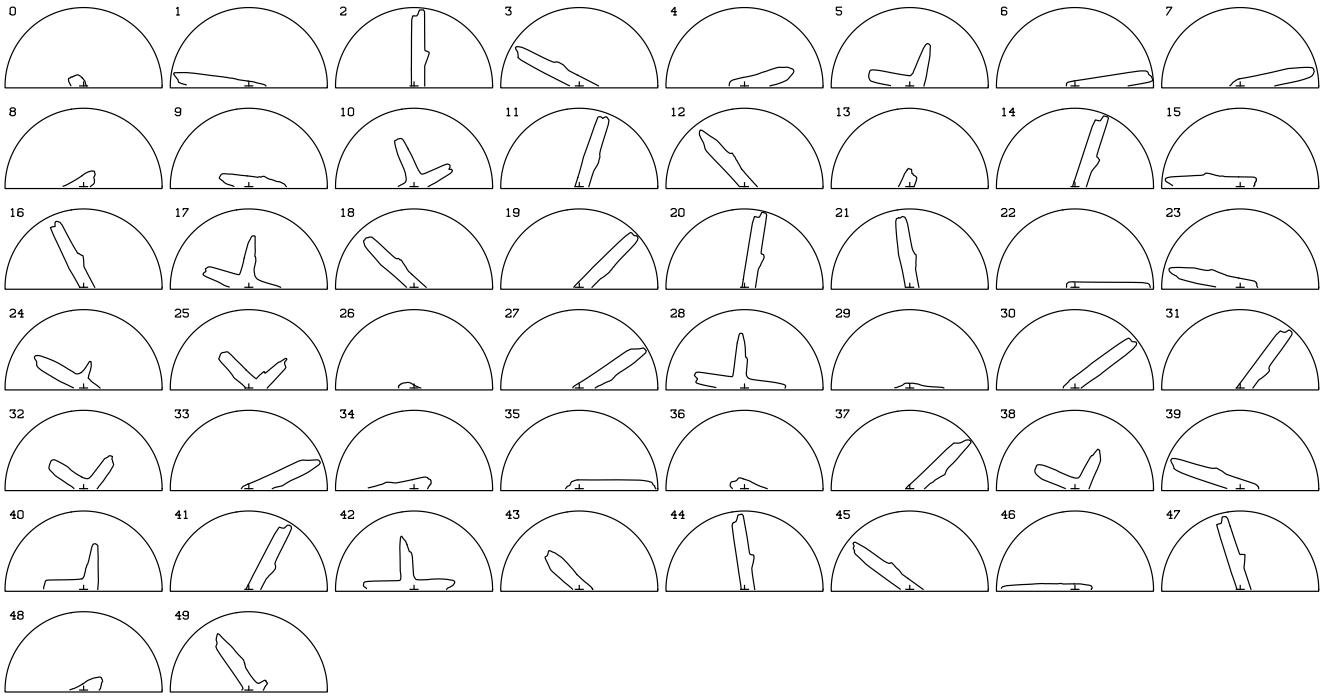

Figure 3: **Learned Perceptual Features.** The agent's self-organizing feature map learns a set a perceptual prototypes that are used to define perceptually distinctive states in the environment. Above is one learned set of features. Each feature is a prototypical laser rangefinder image plotted radially, with the robot at the origin. The prototypes represent a wide variety of different possible views of the environment.

To learn the initial set of perceptual features, the agent trained a Growing Neural Gas network with its sensory input over an extended random walk through the environment. An example set of learned features is shown in Figure 3. The set of features comprises a wide variety of prototypical views of the environment.

The agent had four simple, open-loop trajectory-following control laws, one for each primitive action. Each TF controller simply repeats its primitive action until a new prototype becomes the winner. The agent hillclimbs on the winning feature by sampling the gradient of the feature activation in the direction of each action, and choosing and executing the action for which the gradient was highest. It repeats this process until each action's gradient is negative. The combination of TF and HC controllers gave the agent four high-level actions *tf-ahead-then-hillclimb*, *tf-back-then-hillclimb*, *tf-left-then-hillclimb*, and *tf-right-then-hillclimb*.

Policy Learning

The agent learned its high-level control policy using standard, episodic, tabular SARSA(λ) reinforcement learning, using the SOM winner as the state. Learning curves were generated for 20 runs of 500 episodes in each of two experimental conditions, the first using only primitive actions, the second using only high-level actions. The results of a run for one feature set are shown in Figure 4. Learning performance is significantly better for the agent using the high-level actions. Runs were also done with other learned feature sets. For all feature sets, the primitive actions gave about the same performance, but the amount of performance increase for the high-level actions varied, although, in all cases the agent using high-level actions performed at least as well as the one using primitive actions.

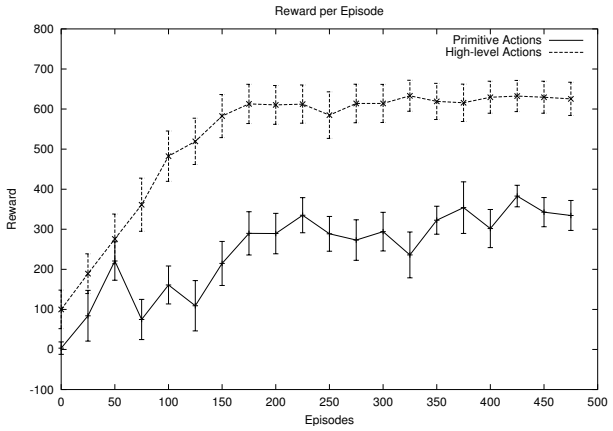

Figure 4: **Learning Performance.** Comparison of the reward earned per episode using primitive actions vs. using high-level actions. Each condition used the feature set shown in Figure 3. Each curve is an average of 20 runs. Error bars indicate +/- one standard error.

Discussion

The results of our experiments suggest that our method can improve reinforcement learning over just using a coarse-coded state representation and small-scale, local actions. This benefit seems to have two main sources: reduced task diameter, and reduced action uncertainty.

First, using high-level actions reduces the diameter of the problem, requiring fewer actions to get near the goal, and making it easier to discover the critical decisions in the task. Using the short-range actions, the agent must make around 300 actions to get to the goal. Since a critical choice – turning right at the intersection – that must be learned is approximately halfway through the action sequence, and the reward comes at the end of the task, it takes many trials and much exploration to back up the reward that far and discover the correct choice. Traveling between distinctive states, the agent needs fewer than 10 actions to arrive near the goal; This makes it much easier to propagate the reward back to the critical choice point.

Second, moving between perceptually distinctive regions and hill-climbing to distinctive states reduces positional uncertainty, and thus reduces the uncertainty in action, making it easier to learn the task. State abstraction methods that partition a continuous state space alias the environment creating uncertainty about the outcome of actions. By definition, all states within a partition of the state space are aliased, but the outcome of an action at two different states in one partition may not be the same. For example, two states in the same partition may differ in orientation by a few degrees, in this case, trajectory-following forward may lead to drastically different states.

A general challenge when automatically generating abstractions is making sure that a solution to the problem still exists in the abstracted search space. In our case, using the high-level actions helped less with some feature sets than with others, though it never hurt. The feature sets that didn't benefit from the high-level actions tended to be smaller suggesting that they may have had insufficient coverage of the sensory space, and the set distinctive states defined by those features did not contain some states necessary to speed learning. One might think that the simple answer to this problem is to configure the GNG to produce a larger feature set, but of course, increasing the number of features likely also increases the task diameter using high-level actions. Exploring this trade-off is a direction for future work.

Future Work

Our continuing work includes expanded evaluation of the method against other other reinforcement learning methods, as well as several improvements to the method, itself.

Expanded evaluation – Although the preliminary experiment shows that our method can reduce the diameter of a navigation task by a couple of orders of magnitude, a detailed comparison with existing reinforcement learning methods for continuous state/action spaces remains to be conducted. We plan to conduct such a comparison.

Learning TF and HC controllers – The trajectory-following and hill-climbing controllers described above are relatively naïve. In principle, both could be learned. Hill-climbing, for example, was done by manually sampling the feature gradients. This is both expensive and likely to be impractical or impossible with noise or irreversible actions. Better would be to learn to predict the gradients from experience using standard supervised learning techniques. Alternatively, the hill-climbing process itself could conveivably be treated as a reinforcement learning problem.

Learning Features and Policy Concurrently – Currently, this method requires an initial exploration phase to learn a set of perceptual features before beginning to learn the action policy. Using the Growing Neural Gas, it may be possible to learn the features and actions concurrently.

Building a Causal Model – In cases where the same agent must perform multiple tasks in the same environment – such as navigation to several different places in the same building – it should be possible to use the learned high-level actions to build a causal model, in the form of the transition function $T(s, a, s')$ that predicts the succeeding state given a starting state and an action. Once the destination states are identified, navigation could proceed by planning, rather than policy learning.

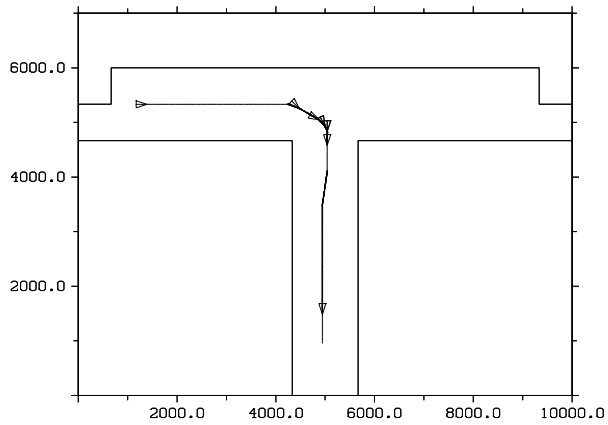

Figure 5: **Navigation using abstraction..** An example episode after the agent has learned the task using the high-level actions. The triangles indicate the state of the robot at the start of each high-level action. The narrow line indicates the sequence of primitive actions used by the high-level actions. Navigating to the goal requires only 7 high-level actions, instead of hundreds of primitive actions.

Conclusion

We have described a method by which an agent in a high-diameter, high-dimensional continuous environment uses a self-organizing feature map to construct a perceptual abstraction that allows it to define a set of perceptually distinctive states in the environment. A set of extended, high-level actions, consisting of combinations of trajectory-following and hill-climbing controllers, allow the agent to move between these states. In an experiment with a simulated robot, this combined perceptual and temporal abstraction allowed

an agent to reduce the effective diameter of a navigation task from approximately 300 small, local actions to 7 high-level actions.

References

Digney, B. 1996. Emergent hierarchical control structures: Learning reactive / hierarchical relationships in reinforcement environments. In *Proceedings of the Fourth Conference on the Simulation of Adaptive Behavior: SAB 98*.

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 *AAAI00*, 826–831. AAAI/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; New York: Springer.

Kuipers, B. J. 2000. The spatial semantic hierarchy. *Artificial Intelligence* 119:191–233.

Martinetz, T. M.; Ritter, H. J.; 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 discogery of subgoals in reinforcement learning using diverse density. In *Machine Learning: Proceedings of the 18th Annual Conference*, 361–368.

McGovern, A. 2002. *Autonomous Discovery of Temporal Abstractions from Interaction with an Environment*. Ph.D. Dissertation, The University of Massachusetts at Amherst.

Nehmzow, U., and Smithers, T. 1991. Mapbuilding using self-organizing networks in really useful robots. In *Proc. SAB '91*.

Nehmzow, U.; Smithers, T.; and Hallam, J. 1991. Location recognition in a mobile robot using self-organizing feature maps. Research Paper 520, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, UK.

Provost, J.; Beeson, P.; and Kuipers, B. J. 2001. Toward learning the causal layer of the spatial semantic hierarchy using SOMs. Submitted to the AAAI Spring Symposium Workshop on Symbol Grounding.

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.

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.