Implementing SODA in Embodied Robots

Ross Greenwood, Emanne Desjardins
Swarthmore College, Swarthmore, PA 19081, USA
E-mail: {rgreenw1, edesjar1}@swarthmore.edu

May 14, 2010

Abstract

Giving robots the ability to learn without the guidance of a researcher is a practical method of developing more adaptable and intelligent systems. Self-Organizing Distinctive-State Abstraction (SODA) is an algorithm that allows a robot to learn routines from from raw sensorimotor inputs through training. The goal of our experiment was to implement SODA on a simulated version of the Khepera II robot in a T-shaped Maze environment. The task we aimed to complete was training our robot to complete a navigation task that consisted of finding a path from the upper left or upper right corner to a light placed at the bottom of the T-maze. We managed to successfully implement the phases dealing with exploration of the environment and training of HC and TF options, but were unsuccessful at training our robot to properly complete the the light-finding task.

1 Introduction

Developmental robotics seeks to simulate in robots the cognitive and learning processes of biological organisms, in order to develop robots that can adapt to a wide variety of problems. Since robots must operate in a noisy and unpredictable world, developmental robotics systems are often first developed and tested in simplified, simulated robot environments. However, since the ultimate goal of robotics research is the design of real-world robots, these methods are useless unless they can be shown to work with embodied robots. To this end, we have attempted to implement a particular developmental robotics algorithm in a physical robot environment. We had a limited degree of success using the physical Khepera robot, since its sensors did not have the necessary richness to learn a significant amount of information about its environment, and as such was unable to complete the light-following task.

1.1 SODA: Growing Neural Gas and Reinforcement Learning

The developmental method we chose to implement is Self-Organizing Distinctive-State Abstraction (SODA). SODA is a developmental robotics system that combines the Growing Neural Gas algorithm with reinforcement learning to train a robot on a given task [2]. Reinforcement learning has been most effectively implemented in agents that operate in a discrete task environment [4]. SODA expands the applicability of reinforcement learning by making use of one of the key concepts of developmental robotics: classification. Intelligent agents use classification to abstract away unnecessary details and simplify their representation of the environment. This type of simplification allows agents to apply discrete-world algorithms such as reinforcement learning to continuous environments with high-dimensional feature space.

The overall architecture of SODA method is a three-phase learning process, as seen in figure 1. The agent first categorizes the world. Afterward, it learns to navigate between categorizations by training and using a reinforcement learning technique known as options. Finally, it uses those options to perform reinforcement learning on the desired task.

The classification mechanism used by SODA is Growing Neural Gas (GNG). Given a probability distribution, which in our case is the sensory-motor feature space of the robot in the environment, the GNG learns

Figure 1: Developmental architecture of SODA

TF: (1,'f')	actions			
States (topN)	Forward	F-Right	F-Left	F-Back
(1,4,5)	0	1.5	0.26	0.37
(1,3,2)	1.0	0.83	0.09	0.73
...				

Table 1: Sample trajectory following SARSA table. There exists one such table for each combination of GNG prototype and primitive action. This table corresponds to prototype 1 and the primitive action 'f'.

to represent the data using a minimal number of nodes or "prototypes". Regions in the sensory-motor space which correspond to roughly similar situations are associated with the same prototype. Thus, the prototypes act as classifications of the world. Given a new input vector, the an agent can classify that input by finding the closest GNG prototype to which it corresponds. Once SODA uses a GNG to create these classifications, the classifications are used as states for the reinforcement learning phases of the developmental process.

In the second phase of the learning process, the robot learns to use hill climbing (HC) to perform gradient descent and approach the closest GNG prototype. It then navigates to another prototype using trajectory following (TF), in which the robot seeks to minimize changes to its sensory inputs, while still making progress in some direction. In the newest version of SODA, which we implemented, these HC and TF options are learned using SARSA. SODA defines a set of primitive actions for the robot. For each pair of primitive action and GNG prototype, it creates a reinforcement learning table. The possible actions that the robot can take when training on that table are the primitive that corresponds to the table, or that primitive action along with another small movement in the direction of some other primitive action. SODA also creates one HC learning table for each GNG prototype. The actions it can choose when training this table are just the primitive actions.

To get the states for the reinforcement learning tables, SODA uses a topN state representation. This means, it classifies every point in the environment based on the N closest GNG prototypes. So within the perceptual neighborhood of a given prototype, the environment can be classified by the N-1 prototypes that are closest to the agent after the closester prototype. Example reinforcement learning tables for trajectory following and hill climbing can be seen in tables 1 and 2.

The third phase of SODA involves using SARSA along with the HC and TF options to train the robot to complete the task. Having developed the ability to travel to GNG prototypes using hill climbing, and then to travel between perceptual neighborhoods using trajectory following, the robot learns to string these actions together using SARSA in order to complete the desired task.

HC: 3	actions			
States (topN)	Forward	Right	Left	Back
(3,5,8)	0.9	0.5	1.28	0.26
(3,6,2)	1.3	0.73	1.09	0.58
...				

Table 2: Sample hill climbing SARSA table. There exists one such table for each GNG prototype. This table corresponds to prototype 3.

1.2 Previous Work Using Soda

The developers of SODA demonstrated that their system can be used to solve a simulated robot navigation task [3]. The version of SODA used in this first experiment was slightly less sophisticated than the one described above. In the original version, the agent navigated from an arbitrary point in feature space to a the nearest prototype using a naive hill climbing algorithm. This initial version implemented hill climbing by trying each primitive action, and then reversing each action to determine which action allowed it to approach the prototype most rapidly. It did a similar "guess and check" technique to learn trajectory following. This version of SODA was implemented by Barlow and Wiedenbeck [1] to solve a navigation task for embodied robots. They found that the original version of SODA did not generalize well to real-world robots. The primary issue was the non-adaptive implementation of hill climbing. This technique works reasonably well in simulation. However, one of the greatest difficulties of embodied robots is that actions cannot always be correctly reversed. Thus, this version of hill climbing does not work well for real robots.

The newer version of SODA uses more adaptive versions of hill-climbing and trajectory following, by employing the hierarchical reinforcement learning system described above [2]. This technique allows the robot to train a different "option" policy for trajectory following and hill climbing at each perceptual prototype. This allows the robot to find different ways to travel to and between perceptual prototypes. Each option is learned using reinforcement learning, which does not require the robot to reverse its actions, and thus this technique is more applicable to real world robots. The authors demonstrate that this new implementation of SODA can effectively be applied to a simulated robot navigation task. However, they do not demonstrate that it can be applied to real world robots. Since this new version of SODA fixes the issue of reversing actions to perform gradient descent, we initially set out to demonstrate that it could be applied to a real world robot navigation task.

2 Experiments and Results

Having initially endeavored to apply SODA to a real world task, we ran into some major difficulties that prevented us from making progress. Due to this, we decided to attempt to implement our system in simulation. Thus, our experiments were performed in two distinct phaes: physical robot testing, and simulated robot testing.

2.1 Robot and Environment for Real-World Experiment

We implemented a version SODA, including a phase that trains hill climbing (HC) and trajectory following (TF) options, as described Provost et al. (2007). The purpose of our experiment was originally to test SODA with Options on a real robot in a physical environment, to see if the success in simulation [2] could be replicated in the real world. We chose to

For our original experiment we, sought to implement a light-seeking robot in a T-shaped environment. We used the Khepera II robot, as was done by Barlow and Wiedenbeck. This robot is equipped with 8 infrared range sensors and 8 light sensors. The topology of the sensors can be seen in figure 2. We placed a bright light source at the base of the T and obscured it with a piece of white paper to reduce the brightness so as to minimize reflection on the back wall. The task for the robot was to navigate from one of the wings at the top of the T to the light at the base of the T.

2.2 Results and Implementation Difficulties

We were unable to successfully train the robot on this task. We have identified a number of causes for this failure to achieve positive results. A primary cause is that the sensors of the Khepera are limited in both range and accuracy. It was difficult for the robot to take measurements that accurately classified the world, because it's sensors were not able to perceive objects that were more than one robot unit away. We tried to correct for this by maing the paths in the world more narrow. However, this correction could only be made up to a certain point without harming the mobility of the robot. The robot's inability to perceive larger

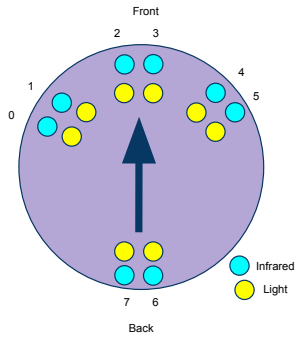


Figure 2: Sensor topology of the Khepera II robot that was used in the real world task

distances cause the robot to perform perceptual aliasing. This is a phenomenon in which vastly different scenarios in the real world are classified as identical by the intelligent agent's internal representation of the world. This failure to distinguish between distinct states of the world made it impossible for the robot to develop a classification of the world that allowed it to navigate to the goal.

Another roadblock was the difficulty in automating the option-learning phase. To effectively train the options, the robot had to maneuver around in the world, and use SARSA to train the options whenever it encountered a new perceptual neighborhood. It was difficult to ensure that each perceptual neighborhood was encountered enough times such that the SARSA tables for that neighborhood were sufficiently trained.

2.3 Robot and Environment for Simulated Experiment

Given the lack of success of our implementation with embodied robots, we opted to attempt to recreate the results of Provost et al. using a simulated robot environment. We recreated the T-Maze setup in simulation and placed a circular light at the bottom of the maze as seen in 3. We also used a simulated robot that had a much more rich set of range sensors. Our simulation robot was equipped with 16 sonar sensors placed at intervals around the robot's body, two light sensors in the front right and front left of the robot. In the simulation world, the sonar is able to detect obstacles from a large distance away. This makes it significantly easier to keep track of the robot's position in the world, particularly if the light source is within range of its sensors. Additionally, this allows the robot to better categorize the world, and avoid the issue of perceptual aliasing that we experienced in the physical robot experiment.

2.4 Results for Simulated Robot Task

This experiment was also unsuccessful. While we avoided many of the issues that we had to deal with in the physical robot experiment, there were still some significant difficulties that we were not able to deal within the time frame of the project. One success was that we were able to get GNG prototypes that correspond to reasonable states in the real-world. Figure 2.4 shows a sample of GNG prototypes from a typical run, plotted as histograms representing the robot's range and light sensors at that point in the environment. While these histograms suggest that the robot was successfully able to categorize the world, there are some issues that we can see. First, for the trial that the histograms in figure 2.4 came from, the GNG created 16 prototypes. Of those 16, all but 3 had light readings of 0. This means that for most of the categories, the robot was not seeing the light. This is potentially problematic, in terms of being able to solve the navigation tasks. We would like our robot to have a series of classifcations in which the intensity of the light gradually increases. If this is the case, it would be able to train HC and TF options to travel between those states, and hopefully navigate towards the light as desired. Our GNG prototypes do not show such a pattern, which suggests that they might not enable us to train our robot on the light finding task.

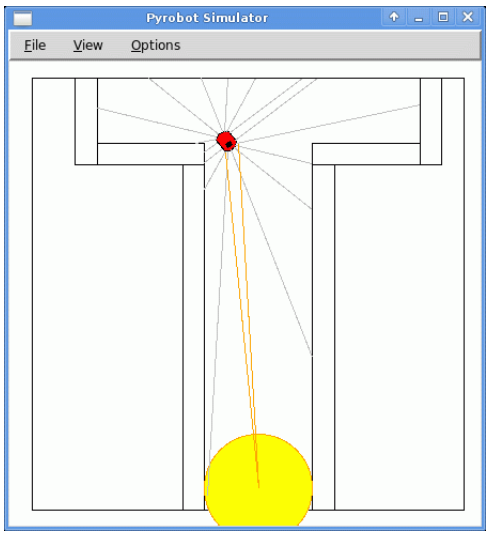


Figure 3: Screenshot of the robot in the simulated environment.

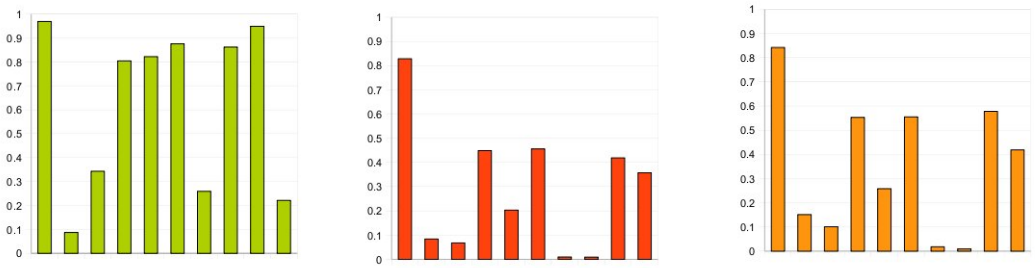
An additional problem that we notice in these histograms is that some of the prototypes require the robot to be very near to the wall. Of the 16 prototypes, 6 had range sensor readings of less than 0.1 robot unit. Further, many of the prototypes had range sensor readings barely above 0.1 robot units. Thus, in order to approach most of the GNG prototypes, the robot had to get very close to the wall. This made it very difficult to train the HC and TF options, because in order to hill climb to the prototype of a perceptual neighborhood, the robot had to ram into the wall. The robot was often unable to get satisfactorily close to the prototype, and was permanently stuck running into the wall. This behavior made it impossible to train the options, and therefore the robot was unable to train on the light-finding task.

3 Discussion

SODA is a highly complex algorithm that combines several different phases in order to be implemented correctly. We managed to get the robot to thoroughly categorize the world and train the HC and TF options. However, we did not manage to train the robot successfully to complete the light-finding task. One difficulty that arose in our implementation was creating a way for the robot to train the action-pairs without colliding with walls or getting trapped in corners. If the robot failed to avoid walls, it would routinely get trapped and be unable to continue training so we would be forced to manually rescue it. However, manually adjusting the position of the robot is impractical as it makes training sessions tedious and defeats the purpose of designing an autonomously-learning robot. Additionally, preventing the robot from colliding with the wall altogether would not have been beneficial since the robot would have not defined a state for this situation, and later would not have been able to associate this state with a negative reward.

One direction for future work would be to improve the ability of our system to categorize the world. This might involve using different sensors. It might also involve changing the way in which the robot explores the world during the GNG training phase. We could modify the code so that our robot does not get too close to the wall while training the GNG. This would prevent the GNG from developing prototypes that require the robot to run into the wall. We might also modify the exploratin phase to ensure that the GNG has more prototypes in which the robot can see the light. Ideally, we would want a progression of GNG prototypes with increasing light values, to allow the robot to learn to navigate between those prototypes to find the light.

Additional work in developing SODA for physical robots would probably require using a different robot with a more powerful set of sensors. SODA relies on the ability of the robot to develop categorizations, and



(a) (b) (c)

Figure 4: Histograms for GNG prototypes for a typical run of the GNG on the simulated robot task. The first 6 columns represent the range sensors, going clockwise from the front left sensor. The Next two columns represent the light sensors. The last two columns represent the translate and rotate motors. The prototype in (a), most of the range sensors are long, but one is short. This suggests that the robot is facing a corner to its left. Additionally, the left light signal has a significantly lower value than the right light sensor. This suggests that the robot's right ligh tsensor is pointed more directly towards the light. This GNG prototype seems like it is a useful classification of the world. However, the prototypes in (b) and (c) seem problematic. First, the states they represent seem almost identical. There is very little variation in the magnitude of each of the sensor readings. This suggests that these prototypes would be less useful in helping our robot classify the environment in order to perform reinforcement learning.

the issue of perceptual aliasing makes it difficult if not impossible for SODA to work. One key improvement would be stronger range sensors that could detect objects at a greater distance. Another useful sensor, which was used in the simulated robot task developed by Provost et al. [[2]], would be a compass. This would allow the robot to better categorize the world, by having some understanding of its orientation in the environment. This would provide yet another tool to prevent perceptual aliasing in the GNG classifications.

If either the simulated or physical robot experiments could be made to work, there are a number of future directions that could be taken. First, we could try to use the same learned HC and TF options that were developed for the light finding task to solve some other task in the same world. Additionally, SODA seems best suited to a world where there are few changes over time. This is because it uses the GNG to classify the world in the beginning, and then relies on those classifications for the remainder of the training process. One possible improvement to SODA would be to allow the GNG to continue updating it's classification of the world throughout the entire learning process. This would enable the robot to update it's internal representation of the environment, to take into account any changes that might occur in the world. One difficulty of course, is that as the GNG changes, the SARSA tables will necessarily have to change, so this type of algorithm might be very challenging to implement. However, if successful, it would have the effect of making SODA more adaptive in dynaimc environments.

References

[1] A. Barlow and B. Wiedenbeck Embodying and Improving SODA In *CS81 Final Projects*, 2008.

[2] J. Provost, B.J. Kuipers, and R. Miikkulainen Self-Organizing Distinctive-State Abstractions Using Options In *Proceedings of the Seventh International Conference on Epigenetic Robotics*, 2007.

[3] J. Provost, B.J. Kuipers, and R. Miikkulainen Developing navigation behavior through self-oranizing distinctive state abstraction In *Connection Science*, 2006.

[4] L.P. Kaelbling, M.L. Littman, and A.W. Moore Reinforcement Learning: A Survey In *Journal of Artificial Intelligence Research*, 1996.