Case-Based Learning by Observation in Robotics Using a Dynamic Case Representation

Michael W. Floyd and **Mehmet Vefa Bicakci** and **Babak Esfandiari**
Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, Ontario, Canada

Abstract

Robots are becoming increasingly common in home, industrial and medical environments. Their end users may know what they want the robots to do but lack the required technical skills to program them. We present a case-based reasoning approach for training a control module that controls a multi-purpose robotic platform. The control module learns by observing an expert performing a task and does not require any human intervention to program or modify the control module. To avoid requiring the control module to be modified when the robot it controls is repurposed, smart sensors and effectors register with the control module allowing it to dynamically modify the case structure it uses and how those cases are compared. This allows the hardware configuration to be modified, or completely changed, without having to change the control module. We present a case study demonstrating how a robot can be trained using learning by observation and later repurposed with new sensors and then retrained.

1 Introduction

Robots are becoming increasingly prevalent in both industrial and home environments (Thrun 1998). These robots act as the physical embodiments of controlling agents and allow the agents to interact with the world. However, the tasks these controlling agents are required to perform may change over time. It is not reasonable to replace the physical robot every time its task changes, due to both the potentially large monetary costs and the need to redesign the controlling agent to allow it to utilize the new hardware. Additionally, the variety of tasks the robot is required to perform (e.g. for a robot that assists the elderly) or the specific properties of the environment it will be deployed in (e.g. for a search and rescue robot) may not be known in advance. Instead, a more economical solution might be to have a modular robot that can dynamically have new *sensors* (used to sense the environment) and *effectors* (used to perform actions) added as required. While this helps alleviate some of the financial costs of repurposing a robot, it still requires modifying the controlling agent. Newly added sensors or effectors will influence both the sensory features the agent uses to reason and the possible actions it can perform.

The ability to dynamically modify the sensors and effectors available to an agent is vitally important for agents that *learn by observation* if those agents are meant to be deployed in a variety of domains. These agents train themselves to perform a specific task by watching an expert perform the task (Figure 1). As the expert interacts with the environment, by performing actions (A) in response to environment states (S), the observing agent records the action-state pairs (C). The agent can later use those observations to train itself to replicate the expert's behaviour. A primary motivation for learning by observation is that it allows an agent to learn a behaviour even if the expert does not have the necessary technical skills or time to program the agent. Since the expert may not be able to program the agent's behaviour, it is not reasonable to assume the expert has the technical skills necessary to reprogram the agent if the robot is repurposed.

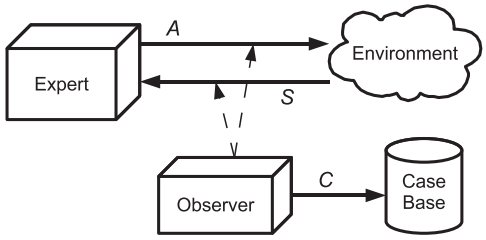

Figure 1: An observing agent watching an expert interact with the environment

We want our learning agent to be independent of the case representation, so that we can repurpose the robot with different hardware or re-train it for different tasks without having to modify any of the agent's code. Instead, sensors and effectors are able to dynamically register themselves with the agent at run-time. The agent can then modify how it reasons, based on the available sensory features, and how it selects actions to perform, based on the available effectors. This also helps make the agent more resilient to hardware failure since it can similarly modify its reasoning and action selection when components are removed. This work is largely motivated by how animal brains deal with newly added sensors. It has been shown that frogs can incorporate sensory information from surgically implanted extra eyes

(Constantine-Paton and Law 1978) and that humans are able to adapt when existing sensors are substituted for new ones (Bach-y-Rita and Kercel 2003). These results seem to indicate that animal brains are not hard-coded for a fixed set of sensors but can use and learn from any sensors that are wired to the brain.

This paper will examine how the need to redesign an agent when a robot is repurposed can be avoided by allowing new sensors and effectors to dynamically register with a case-based learning by observation agent at run time. Section 2 will describe the learning by observation agent and how existing systems deal with changes to the sensors and effectors. The process by which new hardware registers with the learning by observation agent and how the agent makes use of the new hardware is presented in Section 3. A case study showing how a learning by observation agent can handle when the robot it controls is repurposed is shown in Section 4. Related work is discussed in Section 5 followed by conclusions and areas of future work in Section 6.

2 Learning by Observation Agent

Learning by observation agents that can learn a variety of behaviours in a variety of domains (Floyd and Esfandiari 2011; Ontañón and Ram 2011) achieve this by clearly separating how the agent reasons from how it interacts with the environment. By decoupling the agent's reasoning algorithms from its environmental interfaces, this design attempts to avoid biasing the agent to any specific tasks, environments or experts. Our work uses a similar agent design where the learning by observation agent is decomposed into three modules (Figure 2): *Reasoning*, *Perception* and *Motor Control*.

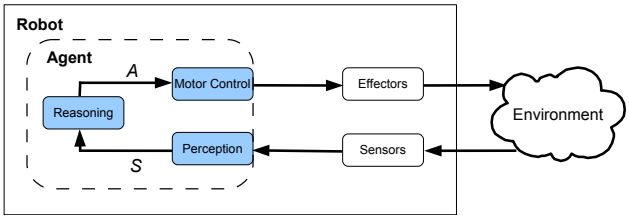

Figure 2: Learning by observation agent design

When the agent is placed in the environment, in place of the expert, it will receive sensory information (from the sensors) and in turn perform an action. Ideally, this action should be the same action the expert would have performed if presented with the same sensory information. The Perception module is responsible for reading the sensor values and converting them into a form that can be understood by the Reasoning module. This sensory information $\mathcal{S}$ contains the values v_i read from each of the N sensors accessible to the agent:

$$\mathcal{S} =< v_1, v_2, \ldots, v_N >$$

This sensory information is used by the Reasoning module to determine what action $\mathcal{A}$ to perform. The Reasoning module uses case-based reasoning with a case base that is automatically acquired by observing the expert. Each case C_j is composed of the environment state, represented by the available sensory information, and the resulting action performed by the expert:

$$C_j =< \mathcal{S}_j, \mathcal{A}_j >$$

When the Reasoning module receives sensory information from the Perception module, it searches the case base for the most similar case and reuses the associated action. This action is then sent to the Motor Control module which causes the appropriate effector to physically perform the action.

The Reasoning module does not contain any prior knowledge about the set of sensors and effectors that will be used so an identical Reasoning module can be used for a variety of hardware configurations. However, both the Perception and Motor Control modules must be modified if the sensors or effectors are changed. In the Perception module these modification require defining the number of sensors that are available and an appropriate similarity metric for comparing each sensory feature. Similarly, in the Motor Control module the set of possible actions must be defined.

Since the Perception and Motor Control modules are hard-coded, this result in a static set of sensors and effectors. Even minor hardware changes require someone with the necessary technical skills to modify the agent. A change in the sensors (from Figure 2) would require changing the Perception module and changing the effectors would require modifying the Motor Control module. The time required to perform these changes is likely significantly less than if the Reasoning module also had to be modified but is still not ideal.

3 Sensor and Actuator Registration

We propose passing the burden of adding new sensors and effectors onto the hardware itself instead of requiring a human expert to modify the agent. We extend the agent design, shown in Figure 2, so that a change in the sensors or effectors no longer requires a redesign of the Perception or Motor Control modules. To achieve this, each hardware component becomes a smart-component that contains the necessary information about itself. Each component has information $\mathcal{I}_k$ that contains its name n_k, value type t_k, and similarity function f_k.

$$\mathcal{I}_k =< n_k, t_k, f_k >$$

The component name is a unique identifier that distinguishes between the various components. The value type defines what kind of value a sensor produces or an effector accepts and is selected from a set of predetermined types. These value types can be simple (boolean, integer, continuous, etc.) or more complex (matrix, vector, etc.) and are used to perform simple error checking on data sent to or received from the component. The component information also includes a similarity function that is used to calculate the similarity (sim) between two values, A and B, that are of the same value type:

$$sim = f_k(A, B), 0 \leq sim \leq 1$$

When a component is connected to the system it registers by providing this information to the agent. The Perception module maintains a list of all registered sensors and the Motor Control module keeps a list of registered effectors. This means that the sensory information received, or observed, by the agent is no longer composed of a constant number of predetermined sensory values. Instead, the number of sensory values received by the agent is dependant on the number of registered sensors and can therefore change over time. In Figure 3, there is initially only one sensor ($Sensor\ 1$) registered with the Perception module so only data from that sensor is sent to the Reasoning module. Later, when an additional sensor registers ($Sensor\ 2$) the Reasoning module will get sensor data from both sensors. This requires having a case definition that is not static but can be modified as new sensors and effectors register with the system.

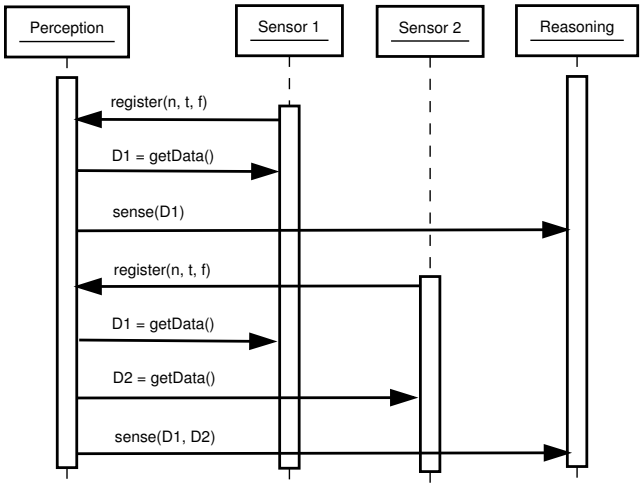

Figure 3: Sensors registering with the learning by observation agent

To allow for a dynamic case definition, the sensory information created by the Perception module is initially an empty set. As new sensors register with the agent, the sensory information will contain values from those new sensors. If there are currently R_t registered sensors at time t, the sensory information will contain data from each of those sensors:

$$\mathcal{S} = \{D_1, D_2, \ldots, D_{R_t}\}$$

Where the data D_j from each sensor contains both the sensor value v_j and the sensor information $\mathcal{I}_j$:

$$D_j =< v_j, \mathcal{I}_j >$$

This means that if a new sensor registers at time t then the cases observed at time $t-1$ will be structurally different from those observed at time $t+1$. This can result in a case base that is not homogeneous but contains cases with different structures. During case retrieval, the sensory information that is used as a query when searching the case base may also have a different structure than some, or all, of the cases. One possible way to avoid having differently structured cases would be to create a new case base every time there was a hardware modification. However, this would result in losing any information stored in the previous case base. For example, a robot used for search and rescue might learn a base set of behaviours initially. Before being deployed in an actual operation, it might need to be outfitted with extra hardware, related to specific environment conditions, and learn several additional behaviours. If the new hardware does not directly influence the previously learnt behaviours then it would be wasteful to empty the case base and would require having all the behaviours observed again.

To facilitate calculating similarity between two sensory information instances, which may be structurally different, we use the similarity approach described in Algorithm 1. The algorithm takes two sensory information instances, S_1 and S_2, one of which is usually the query used during case retrieval and the other is the sensory information component of a case from the case base. For each data item $D1$ in the query sensory information (line 2), a data item $D2$ from the same sensor is found in the second sensory information (line 3). The $findMatch(\ldots)$ function returns a data item with the same component name if one exists in the second sensory information or $NULL$ if it does not. If there was a matching data item, the sensor value of each data item is extracted (lines 5 and 6) along with the associated similarity metric (line 7). That similarity metric is used to calculate the similarity between the two sensor values and add it to a running total (line 8). If the two sensory information instances had at least one common sensor, the average similarity of the sensor values is returned (line 11).

Algorithm 1: Similarity Between Structurally Different Instances

Input: query sensory information (S_1), second sensory information (S_2)
Output: similarity between the two (sim)

Function: *similarity*(S_1, S_2) ***returns*** sim

```
1  comparisons = 0; totalSim = 0
2  foreach D1 ∈ S1 do
3      D2 = findMatch(D1, S2)
4      if D2 ≠ NULL then
5          V1 = getValue(D1)
6          V2 = getValue(D2)
7          simFunct ← getSimilarity(D1)
8          totalSim = totalSim + simFunct(V1, V2)
9          comparisons = comparisons + 1
10 if comparisons == 0 then return 0
11 else return totalSim/comparisons
```

It should be noted that the algorithm, as presented, does not penalize when sensors only exist in one sensory information instance and not the other. This is done to treat the missing sensor values as unknown values rather than values that do not exist. The values do exist in the environment but the agent does not have the necessary visibility to view them

due to a lack of sensory hardware.

4 Case Study

We use a case study of a learning by observation agent adapting when new sensors and effectors are added to demonstrate the applicability of our technique. Initially, the agent is not aware of what type of robot it will be connected to and therefore has no registered sensors or effectors. At this point, the agent is unable to perform any observation or learning because it has no way to interact with the environment. Later, a robot (Figure 4) is connected to the agent. The robot that is connected is the commercially available iRobot Create (iRobot Corporation 2011).

Figure 4: The iRobot Create robot

When the robot is connected to the agent, each of the robot's sensors and effectors register with the agent. There are six sensors, all of which produce binary values, that register with the agent: an infrared detector, left and right bumper sensors (used to detect when the robot runs into something), and three wheel sensors (used to determine if the wheels have pressure against them). The two effectors that register are the drive system and the buzzer. The drive system can be controlled by sending it one of five possible directional values: forward, reverse, left, right and stop. The buzzer can be used to make a beeping sound.

The agent observes and learns a simple obstacle tracking behaviour. A power charging station is placed in the robot's environment and produces an infrared signal. If the infrared signal can be detected, using the infrared sensor, the robot will drive forward. However, if the infrared signal can not be detected the robot turns in a clockwise direction until it can detect the signal. When the robot reaches the power station, as indicated by either of its bumper sensors, the robot will stop.

A human expert was used to demonstrate the behaviour and provided one demonstration of the described behaviour. This resulted in 14 cases being observed. After observing the expert, the agent was able to accurately reproduce the behaviour. To examine the performance, the learning by observation agent observed the expert demonstrate an additional 100 cases. These cases were used as testing cases and each case had its sensor information given as input to the agent. The action performed by the agent was then compared to the action portion of the case to see if they matched. Our results showed the agent was able to select the proper action 100% of the time. The ability of the agent to learn this behaviour is what we would expect given the small problem space of the problem (only 2^6 states). While the behaviour learnt in this case study was simple, it did show that the agent was able to learn without any predetermined knowledge about the task or what robotic hardware it would be controlling. Our goal was not to learn difficult tasks but instead to show the adaptive nature of our learning by observation agent.

The second part of our case study examines a hypothetical hardware upgrade to the robot. The robot is able to detect when it bumps into objects, because of its bumper sensors, but is not able to see when it is approaching a potential obstacle. Initially, the agent observes and learns the behaviour as it did in the first case study. To upgrade the robot we added a sonar sensor that provides a continuous value that indicates the distance to the nearest obstacle. When that sensor is connected, it will register with the agent and all further sensor readings will contain the sonar value. In addition to the hardware upgrade, the required behaviour of the robot was also modified slightly. While the majority of the behaviour is the same, the robot now stops if it is about to make contact with an object (as indicated by the sonar sensor).

The expert, when demonstrating this behaviour, did not demonstrate the entire behaviour but only the new aspects of it. The case base that was generated in the first case study was kept and an additional 6 cases, related to stopping when the sonar sensor indicates an obstacle was close, were added. Even with a change in the structure of cases and differently structured cases in the same case base, the agent was able to learn this behaviour as well. Although, in our example, relearning the entire behaviour might not have taken a significant amount of time, being able to keep and use older cases would be a significant benefit when learning more complex behaviours. More importantly, being able to keep older cases allows an agent to perform life-long learning even when its hardware changes over time.

For the final part of our case study, we examined teaching the agent a completely new behaviour. Instead of adding new cases to an existing case base, the learning by observation agent created an entirely new case base. The new case base was generated while observing the human expert demonstrate an obstacle avoidance behaviour. The robot was controlled to drive forward until it detected, using its sonar sensor, that there was a obstacle approximately 30cm away from it. When an obstacle was detected, it would turn to the left and continue its previous behaviour of driving forward. As with the previous two parts of the case study, the learning by observation agent was able to accurately learn this behaviour. This shows that the agent is able to learn two different types of tasks, driving to a fixed location and obstacle avoidance, without any prior knowledge of the tasks or what robotic hardware it would have available to it.

5 Related Work

The majority of learning by observation systems are designed to operate in a single domain. These include simulated domains like Tetris (Romdhane and Lamontagne 2008), chess (Flinter and Keane 1995), poker (Rubin and Watson 2010), Space Invaders (Fagan and Cunningham 2003), first-person shooter games (Thurau and Bauckhage 2003), domination games (Gillespie et al. 2010), and virtual agent control (Dinerstein et al. 2008). Others are physical domains including robotic arm control (Atkeson and Schaal 1997), robotic soccer (Grollman and Jenkins 2007) and unmanned helicopter control (Coates, Abbeel, and Ng 2008). All of these approaches have prior knowledge of what sensors and effectors will be available and use this knowledge to guide their training. Since all observations have a homogeneous form, the approaches that use case-based reasoning (Fagan and Cunningham 2003; Flinter and Keane 1995; Gillespie et al. 2010; Romdhane and Lamontagne 2008; Rubin and Watson 2010) use a static case structure whereas the approaches that use other learning methods (Atkeson and Schaal 1997; Coates, Abbeel, and Ng 2008; Dinerstein et al. 2008; Grollman and Jenkins 2007; Thurau and Bauckhage 2003) train using fixed-length feature vectors. Our own approach differs by using a dynamic case representation that does not require defining the structure of observations in advance.

The two learning by observation systems that have been shown to work in multiple domains and learn from multiple experts, both of which use case-based reasoning, are the works of Floyd and Esfandiari (2011), and Ontañón and Ram (2011). While both of these approaches are designed to learn in multiple domain, they still require the agents to be modified and told of the available sensors before being deployed in a new environment.

Case-based reasoning has been used for a variety of robotic control systems (Fox 2001; Ros et al. 2006; Supic and Ribaric 2001). Unlike our own work, these systems are designed to control a specific robot and are therefore not able to adapt when the sensors or effectors are changed. Also, each of these works use case bases that are expertly crafted whereas all cases in our system are learnt by observing an expert.

There have been several case-based reasoning systems that use unstructured or weakly-structured cases (Bergmann, Kolodner, and Plaza 2005). These types of cases are particularly common in textual case-based reasoning where the case problem or solution contain unformatted text (Racine and Yang 1997; Recio-García et al. 2005). Similarly, there has also been work on object-oriented case representations (Bergmann and Stahl 1998). These systems use a set of predefined similarity metrics when comparing cases. In our approach, the similarity metrics used to compare cases do not need to be defined in advance and different cases in a case base can have different methods for similarity calculation. This allows for data to be added to a case even if that type of data was not anticipated when the case-based reasoning system was originally designed.

6 Conclusions

In this paper we have described an approach for creating case-based learning by observation agents that does not require reprogramming the agent when the available sensors or effectors are changed. This approach, which was motivated by how animal brains respond to changing sensors, does not use a fixed case representation but instead dynamically modifies the structure of cases as senors and effectors are added. When a new piece of hardware is added to a robot, the hardware registers with the case-based reasoning agent and provides any necessary information about itself. This is particularly beneficial in domains, like search and rescue, where the necessary hardware configuration of the robot can not be anticipated in advance.

We presented a case study where an agent is not aware of the robot it will control or what task it will learn. Our results showed that, when the robot was connected to the agent, the agent was able to modify the case structure it used for reasoning. The agent was then able to observe a simple behaviour, demonstrated by a human, and learn to perform that behaviour. As a second part of our study, an additional sensor was added and the agent was also able to adapt and utilize the new sensor. Both old cases, from the initial robot configuration, and new cases, generated by observing the repurposed robot, were able to be stored in a single case base and used by the agent even though they were structurally different. For the final part of our case study, the agent was trained to perform an entirely new behaviour using its newly added hardware.

While our approach removes the need to reprogram the agent itself, it does not completely remove the programming requirement. Each sensor and effector needs to be programmed with information about itself so it can register with the case-based reasoning agent. If a single sensor is used in multiple robot configurations, the sensor only needs to be programmed once whereas traditional learning by observation systems would requiring modifying the agent for every configuration. Even if each sensor or effector is only used once, the effort required to program each sensor would likely be less than modifying the agent itself. An example of this would be adding a sensor and later removing it. In our design, the registration and deregistration would handle modifying the agent whereas traditional learning by observation systems would require modifying the agent twice (once for addition and once for deletion). While this paper has looked exclusively at robotic domains, our techniques could also be applied to simulated environments (although repurposing a simulated agent might not have many practical applications).

Our future work will look further at the idea of life-long learning and how an agent can update legacy cases as its hardware configuration changes. More specifically, we wish to examine how cases that contain features from sensors that are no longer connected to the agent can continue to be used by learning if there are any similar sensors that could be used instead. Also, we will look at how adding and removing hardware impacts the learning rate of more complex behaviours.

References

Atkeson, C. G., and Schaal, S. 1997. Robot learning from demonstration. In *Fourteenth International Conference on Machine Learning*, 12–20.

Bach-y-Rita, P., and Kercel, S. W. 2003. Sensory substitution and the human-machine interface. *Trends in Cognitive Sciences* 7(12):541–546.

Bergmann, R., and Stahl, A. 1998. Similarity measures for object-oriented case representations. In *4th European Workshop on Case-Based Reasoning*, 25–36.

Bergmann, R.; Kolodner, J. L.; and Plaza, E. 2005. Representation in case-based reasoning. *Knowledge Engineering Review* 20(3):209–213.

Coates, A.; Abbeel, P.; and Ng, A. Y. 2008. Learning for control from multiple demonstrations. In *25th International Conference on Machine Learning*, 144–151.

Constantine-Paton, M., and Law, M. I. 1978. Eye-specific termination bands in tecta of three-eyed frogs. *Science* 202(4368):639–641.

Dinerstein, J.; Egbert, P. K.; Ventura, D.; and Goodrich, M. 2008. Demonstration-based behavior programming for embodied virtual agents. *Computational Intelligence* 24(4):235–256.

Fagan, M., and Cunningham, P. 2003. Case-based plan recognition in computer games. In *5th International Conference on Case-Based Reasoning*, 161–170.

Flinter, S., and Keane, M. T. 1995. On the automatic generation of cases libraries by chunking chess games. In *1st International Conference on Case-Based Reasoning*, 421–430.

Floyd, M. W., and Esfandiari, B. 2011. A case-based reasoning framework for developing agents using learning by observation. In *23rd IEEE International Conference on Tools with Artificial Intelligence*, 531–538.

Fox, S. E. 2001. Behavior retrieval for robot control in a unified cbr hybrid planner. In *14th International Florida Artificial Intelligence Research Society Conference*, 98–102.

Gillespie, K.; Karneeb, J.; Lee-Urban, S.; and Muñoz-Avila, H. 2010. Imitating inscrutable enemies: Learning from stochastic policy observation, retrieval and reuse. In *18th International Conference on Case-Based Reasoning*, 126–140.

Grollman, D. H., and Jenkins, O. C. 2007. Learning robot soccer skills from demonstration. In *IEEE International Conference on Development and Learning*, 276–281.

iRobot Corporation. 2011. http://www.irobot.com.

Ontañón, S., and Ram, A. 2011. Case-based reasoning and user-generated AI for real-time strategy games. In Gonzáles-Calero, P. A., and Gomez-Martín, M. A., eds., *Artificial Intelligence for Computer Games*. 103–124.

Racine, K., and Yang, Q. 1997. Maintaining unstructured case base. In *2nd International Conference on Case-Based Reasoning*, 553–564.

Recio-García, J. A.; Díaz-Agudo, B.; Gómez-Martín, M. A.; and Wiratunga, N. 2005. Extending jcolibri for textual cbr. In *6th International Conference on Case-Based Reasoning*, 421–435.

Romdhane, H., and Lamontagne, L. 2008. Forgetting reinforced cases. In *9th European Conference on Case-Based Reasoning*, 474–486.

Ros, R.; Veloso, M. M.; López de Mántaras, R.; Sierra, C.; and Arcos, J. L. 2006. Retrieving and reusing game plays for robot soccer. In *8th European Conference on Case-Based Reasoning*, 47–61.

Rubin, J., and Watson, I. 2010. Similarity-based retrieval and solution re-use policies in the game of Texas Hold'em. In *18th International Conference on Case-Based Reasoning*, 465–479.

Supic, H., and Ribaric, S. 2001. Adaptation by applying behavior routines and motion strategies in autonomous navigation. In *4th International Conference on Case-Based Reasoning*, 517–530.

Thrun, S. 1998. When robots meet people. *IEEE Intelligent Systems and their Applications* 13(3):27–29.

Thurau, C., and Bauckhage, C. 2003. Combining self organizing maps and multilayer perceptrons to learn bot-behavior for a commercial game. In *Proceedings of the GAME-ON Conference*.