Real-Time Imitation Learning of Visual Behavior by a Mobile Robot

Gabriel J. Ferrer, Eric Huynh
Department of Mathematics and Computer Science
Hendrix College
1600 Washington Avenue
Conway, AR 72032
{ferrer — huynhem}@hendrix.edu

Abstract

We describe a system for a human to train a webcam-equipped mobile robot by remote piloting. This robot learns to imitate the actions selected by the human. To make robotic imitation learning practical, each iteration of training (or classification) must complete quickly enough for the robot to act in a timely manner. Furthermore, it must be possible to interleave episodes of training and autonomous execution, so that the human pilot may correct behavioral problems with the robot as they are observed. To address these issues, we introduce a learning algorithm that executes in constant time and space relative to the number of labeled training samples. This system has enabled successful imitation learning of an obstacle avoidance behavior on a physical mobile robot.

Overview

We are developing a system to enable real-time imitation learning of visual behavior by a mobile robot. By "visual behavior", we mean that the robot has a defined action to perform in response to each input image. By "imitation learning", we mean that our goal is for the robot to learn to imitate the actions specified by a human pilot in a given situation. By "real-time", we require the learning algorithm to update itself in a constant amount of time for each labeled training input. We also require the algorithm to produce an action in a constant-bounded amount of time for each unlabeled input.

We have two additional goals for our system. First, the human pilot ought to have the opportunity to supply additional training when the robot makes a poor action selection. Second, we would like to have available a single hyperparameter that, when it increases in value, increases learning quality while decreasing the frame rate (i.e., images processed per second), and conversely, when it decreases in value, decreases learning quality while increasing the frame rate.

Our current implementation (inspired by clustering approaches such as kNN (Peterson 2009) and the self-organizing map (Kohonen and Honkela 2007)) has all of these capabilities, and we have deployed it on a physical mobile robot. We tested the complete imitation learning system with 16 volunteers on a visual obstacle-avoidance task, 12 of whom agreed that they were able to effectively teach the robot to imitate their performance on the task. We further support this finding with quantitative observations that show how closely the learned behavior matched the behavior modeled by the teacher.

Related Work

Robot imitation learning has been extensively studied; (Argall et al. 2009) give a comprehensive overview of this topic. In this section, we discuss some specific examples that operate in real-time.

(Pomerleau 1991) programmed a self-driving car that used a backpropagated neural network to learn to steer by imitating a human driver. Backpropagation requires significant adaptation to be suitable for use as a real-time learning algorithm. Their solution was to maintain a rotating set of 200 training images. As each new labeled image arrives, 14 variations are created, and the 15 new images replace 15 old images with similar training labels. The network is then trained via backpropagation 50 times using this set.

While this yields a successful real-time imitation learning system, its interrelated hyperparameters are difficult to configure. The quality of the learned result depends upon the number of training images maintained, the number of trainings on each image, the number of hidden nodes, and the learning rate. The first three items also affect the running time. Because of the possibility of overtraining the network, increasing the number of hidden nodes or the number of training images might decrease the quality of the learned behavior while also decreasing the frame rate. Due to this problem in addition to the sheer complexity of four independently varying and interacting hyperparameters, it can be difficult to optimize their values to get the desired trade-off between learning quality and frame rate.

(Sullivan and Luke 2012) describes robots that are controlled using a hierarchical finite-state automaton. Each state defines a behavior, performed as long as the agent is in that state. Transitions are controlled by functions that map the current state and a preprocessed feature vector to a new state. These functions are learned from human demonstration, using a decision-tree approach. They classify imitation-learning systems into those that learn *plans* and those that learn *policies*. As they put it, "...the plan literature builds sparse machines describing occasional changes in behavior, whereas ... policy methods learn fine-grained changes in action, such as might be found in trajectory planning or con-

trol. ... Our work lies in the plan method category."

In contrast, the focus of our present work is to learn a policy rather than a plan. We wish for the robot to learn a response for each new image frame that it receives, several times per second. An additional contrast is that both the goals and the algorithm used by (Sullivan and Luke 2012) requires significant feature preprocessing to create a symbolic input representation. This latter task is part of the work of the machine learning system in our formulation.

A representative example of a policy-based system is that of (Coates, Abbeel, and Ng 2008), in which a human expert pilots a remote-controlled helicopter multiple times through a target trajectory. Their work shows how to learn a complex trajectory given relatively simple sensor inputs. Our work in this paper shows how to learn a relatively simple trajectory given complex sensor inputs. The differences in the underlying data structures employed reflects this.

(Ontanon, Montana, and Gonzalez 2014) classify imitation learning tasks into multiple levels depending upon the complexity of the behavior to be learned. The task our robot learned in this study is Level 2 - Reactive Behavior, because the action selected depends only upon the current observation. Their analysis justifies our decision to use a supervised learning algorithm, given that supervised learning algorithms are sufficient to learn any behavior at Level 2.

(Horswill 1994) created a robot that was carefully engineered to base its behaviors on specific visual cues from its target environment. The present work aims to enable a human to engineer a robot's behaviors for a specific environment via imitation learning rather than by programming.

Applications of clustering in robotics include a combination of the Self-Organizing Map with Q-Learning (Touzet 1997) (Smith 2002), a SOM system for behavior specification (Touzet 2006), and systems based on Growing Neural Gas (GNG) (Provost, Kuipers, and Miikkulainen 2006) (Ferrer 2014). This work is motivated in part by the observation that, because the number of clusters varies, a GNG-based system can violate real-time constraints.

Our algorithm can be viewed as an adaptive variant of kNN (Peterson 2009). Because kNN examines every training sample to classify each unlabeled input, it is not possible to bound the time it needs by a constant. A key goal of ours is to preserve the classification accuracy of kNN while introducing a constant time bound to enable real-time processing.

Unsupervised Clustering

To implement real-time imitation learning, we have built a supervised learning algorithm on top of the unsupervised learner Bounded Self-Organizing Clusters (BSOC) (Ferrer 2016) (see Figure 1). A BSOC is a complete undirected weighted graph. Each node is a reference input for a cluster (comparable to the nodes in a Self-Organizing Map (Kohonen and Honkela 2007)). In our case, these reference inputs are webcam images. Each edge weight denotes the distance between the reference inputs of the clusters. The total number of nodes and edges in the graph is fixed at initialization. This is the key to enabling real-time learning; the total time necessary for training or retrieving depends on the number of nodes in the graph. By specifying that number of nodes as a constant, the processing time can be given a specific real-time bound.

The BSOC nodes are stored in a fixed-size array. Each training input is added to the array as a reference input. If adding the training input would cause the number of nodes to exceed the maximum, the two nodes in the graph with the shortest edge are merged by computing a weighted average of their numerical representations. The training input is then added to the newly available location in the array.

Each node also contains a counter that denotes the number of inputs that contributed to the current node. These counters are used as weights when merging nodes. In addition, each edge weight (i.e., the distance between the reference inputs of the connected nodes) is multiplied by the larger of the two counters for the connected nodes.

The edges are stored in a red-black tree. They are primarily ordered by their distances, and secondarily ordered by the indices of their node endpoints. This enables us to quickly find the smallest edge when needed in the `train` method. When two nodes are merged (and removed from the main graph), their edges are also removed from the red-black tree.

Let M be the maximum number of nodes for a BSOC. Each call to `lookup` or `insert` makes no more than M calls to the distance function. Each call to `insert` performs up to $M \log M^2 = 2M \log M$ numerical comparisons when inserting an edge connecting the input to each of up to M existing nodes. Each call to `train` makes up to two calls to `insert`, along with a comparison and $2M$ edge removals (each requiring $2 \log M$ comparisons in a red-black tree). Given that M can be fixed as a constant, we can say that each call to `train` or `lookup` takes constant time.

Building a Supervised Learner

The robot can perform one action at a time from a fixed number of options: FORWARD, LEFT, and RIGHT. Each node in the BSOC has an associated histogram. After each call to `train()`, a call to `lookup()` is performed to determine the node in the BSOC that most closely corresponds to the training input. The move associated with the training input has its count increased by 1 in the histogram for that node. When the BSOC algorithm merges nodes, the corresponding histogram counts of the source nodes are added together to provide a histogram for the newly merged node.

When an unlabeled image is supplied, a call to `lookup()` again yields the node with the reference image that is closest to that input. The move with the highest count is the action the robot will select for the unlabeled image.

Hardware and Software

We deployed this system on a Lego Mindstorms EV3 mobile robot using a webcam via its USB 1.1 port, by which it receives 14 images per second at a resolution of 160x120. We subsampled the images by a factor of 4, so the image input size for our algorithm is 40x30. The implementation is in Java, using the LeJOS library. The human pilot controls the robot using an Android mobile device.

```
setup(max-nodes)
  edges = new red-black-tree
  nodes = array of inputs

lookup(input)
  for each node
    find distance from input to node
  return (node-index, distance) of
     closest node

train(input)
  insert(input, 1)
  if number of nodes exceeds max-nodes
     edge = edges.remove(smallest edge)
     (n1, n2) = endpoints of edge
     Remove n1 and n2 and their edges
     insert(merged(n1,n2),
            n1.count + n2.count)

insert(image, count)
  add (image, count) to nodes
  for each existing node n
     d = distance from image to n
     Create and insert a new edge:
     - First vertex is image
     - Second vertex is n
     - Weight is d * max(count(image),
                         count(n))

merged(n1, n2)
  w1 = n1.count / (n1.count + n2.count)
  w2 = n2.count / (n1.count + n2.count)
  img = w1 * n1.image + w2 * n2.image
  return img
```

Figure 1: Pseudocode for BSOC

When driving forward, the robot moves at a rate of 16 cm/s. It turns by rotating one wheel backward and keeping the other wheel stationary, at $\frac{1}{4}$ the velocity employed when driving forward. In this configuration, the robot turns at a rate of about 18.5 degrees per second, requiring about 19.5 seconds to make a complete rotation.

Experimental Evaluation

Overview

We conducted three rounds of experiments to assess the ability of our algorithm and system to learn to imitate a modeled behavior. In the first round, we recorded a series of 700-frame movies from the robot in different environments. We then performed off-line training to benchmark our performance against an implementation of k-nearest-neighbors.

In the second round, the robot attempted to train itself to avoid obstacles. We programmed a simple obstacle-avoidance controller that used a sonar and bump sensors to determine its actions. We then assessed the degree to which our imitation learner learned that same behavior with images as its input. The goal of this round of experiments was to assess the impact of varying the number of BSOC nodes on the cycle time and the quality of learning that took place.

In our third round of experiments, 16 human subjects trained the robot in an obstacle-avoidance behavior using a remote control. We then assessed how well the robot imitated the behavior that the human demonstrated. The goal of this round of experiments was to assess the degree to which the robot could imitate a human behavior.

All 16 participants were undergraduate students at our institution. Three of the students had completed two or more computer science courses. One of these students had prior experience with robotics, as did another student who had not taken any computer science courses. Two additional students were currently enrolled in the introductory computer science course. The remaining 10 students had no prior experience with computer science or robotics.

We used two different environments for our experiments. The first environment was created deliberately to be very simple. We created a boxed area using four white foam walls. Each of the four walls is 1.15 meters long, for a total area of 1.3225 m^2. The floor is carpeted with a pattern. We employed this environment for all of the experiments in our first round (robot self-training). It was also used for the first training run with each human subject.

Our second environment was a furniture-free office with area approximately 8.7 m^2. This office has a different carpeting pattern than the first area, and had several obstacles of varying appearance, including a large trash can, a backpack, and a violin case. This was intended to be a more challenging task and a more "natural" environment for an indoor robot.

Configurations

We calculated our metrics for each of the following configurations:

- Is the robot *Learning*, or is it *Applying* its learned behavior?
- Which action is the robot performing?
- Is the robot imitating a human trainer (and, if so, which human?) or imitating the sonar controller?
- In which of two test environments did the experiment take place?
- How long was the initial training period?
- How many BSOC nodes were used for learning?
- For the human experiments, did the human have a chance to retrain the robot after observing its behavior?

Metrics

On each cycle, the robot logged the following information:

- Current action being performed
- Whether the robot is moving autonomously or following commands
- Current sonar value
- Time elapsed since previous log

From this data, we devised the following metrics:

- Mean cycle time
- Behavior Profile sum-of-squared-differences

A Behavior Profile for a given configuration is calculated as follows. On each cycle, the robot is currently executing one of three moves (FORWARD, LEFT, or RIGHT). Across all executions of a given move, we calculate the probability that the sonar reading is below a value c. We express this as $P(d < c|move)$. We then calculate this value for every value of c from 0 to 4 meters, with an increment of 0.02. This collection of values is the Behavior Profile for a given configuration.

The rationale for this metric is as follows. We need a concise way of characterizing how the human and the robot make decisions in a specific setting. As c increases, $P(d < c|move)$ increases monotonically. Moves that happen more often at lower values will see $P(d < c|move)$ increase more quickly in comparison to those that happen more often at higher values. As many different situations correspond to the sonar reading being below a given threshold value, this metric provides a concise way to characterize the overall pattern of behavior that results in a particular move.

An example of a behavior profile for one experiment is given in Figure 2. The c values are along the x axis, and the $P(d < c|move)$ values are along the y axis. The top graph is the profile for LEFT, while the one below it is the profile for FORWARD. $P(d < c|LEFT)$ increases much more quickly but plateaus at a lower level than $P(d < c|FORWARD)$. The dashed lines in each graph represent the human teacher's action selections, while the other two lines represent the robot's action selections.

The relative similarity between lines shows a similar behavior. To concisely assess this similarity, we compute the sum-of-squared-differences (SSD) between the $P(d < c|move)$ values of the corresponding lines. Close imitation is indicated by a low SSD value.

Results: kNN Benchmark

We recorded five pairs of videos of 700 frames each from our robot's camera while piloting it remotely. Each pair was recorded in a different room. For each pair, we trained 3nn, 9nn, BSOC-32, BSOC-64, and BSOC-128 twice; once using the first video as the training set and the second video as the testing set, and a second time with the video roles reversed. Figure 3 shows the mean number of correct classifications (out of 700) for each algorithm configuration. Increasing k and the BSOC nodes both improve performance slightly. BSOC performs slightly worse than kNN, but their overall performance is closely comparable.

Results: Self-Trained Robot

In this experiment, we programmed an obstacle-avoidance controller that selected moves according to the algorithm given in Figure 4. (We added a two-second minimum to its turns to keep it from getting stuck.) We ran six experiments per number of BSOC nodes, for a total of 36 experiments. In half of the experiments, the training time was 1.5 minutes. In the other half, it was three minutes. In both cases, the robot then ran autonomously for one minute afterward.

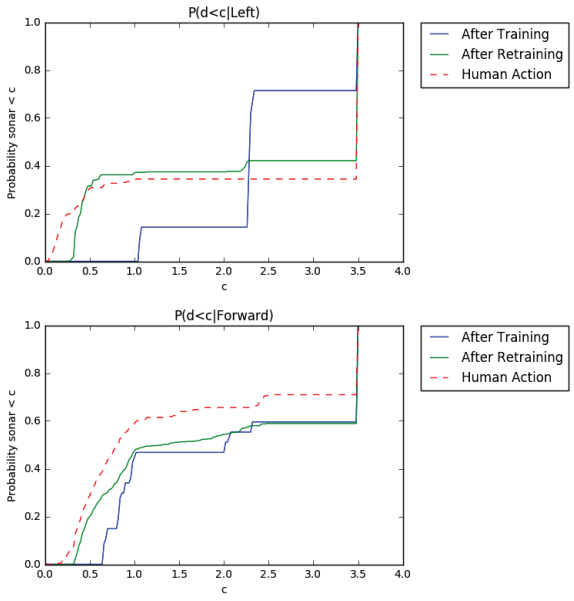

Figure 2: Behavior Profile Example

kNN vs BSOC		
Algorithm	$\bar{x}$ Correct	σ
3-nn	609.5	46.0537126
9-nn	624.4	36.08693208
BSOC-32	601.1	51.96676288
BSOC-64	614.5	40.46740801
BSOC-128	616.1	36.61951514

Figure 3: kNN vs BSOC

Figure 5 shows the relationship between the number of BSOC nodes and the cycle time. Cycle times for the Learning phase (when the system is being trained) and the Applying phase (when the learned behavior is applied) are provided separately. As expected, the cycle-time penalty for increasing the number of nodes is clear. The minuscule standard deviations demonstrate that this implemented and deployed learning algorithm can meet real-time deadlines with reasonably high precision.

Figure 6 shows the relationship between the number of BSOC nodes and how closely the learned behavior imitates the demonstration. There is a clear trend that imitation improves as the number of nodes increases, but this is qualified by the observation that the standard deviations are large.

Interestingly, imitation becomes worse at the largest numbers of nodes. We attribute this to the decrease in cycle time. When an image is acquired by the robot, it is sent to the BSOC for classification. When the classification is complete, the robot executes the corresponding action. A new sonar reading is then obtained. Both the action and the sonar reading are then logged. With 64 nodes, the cycle time be-

```
while not stopped
  if sonar < 20 cm or bumped
    Turn left for two seconds
  else Drive forward
```

Figure 4: Pseudocode for Simple Controller

BSOC Nodes vs. Cycle Time				
Nodes	$\bar{x}$ Cycles/s (Learning)	σ	$\bar{x}$ Cycles/s (Applying)	σ
2	14.90	0.04	12.16	0.02
4	14.52	0.48	9.61	0.04
8	13.24	0.35	6.59	0.04
16	11.26	0.15	4.03	0.03
32	7.13	0.08	2.33	0.03
64	4.0	0.12	1.57	0.01

Figure 5: BSOC Nodes vs. Cycle Time

came slow enough that the robot had often transitioned to a situation that would have warranted a different action. From this observation, we concluded that about 2 cycles/second was our minimum usable cycle time.

The training time had a small but arguably insignificant effect on the robot's performance, as we can see in Figure 8. While the longer training time yielded an average 50% improvement, the standard deviations were so large that they cannot be claimed to be significantly different.

Results: Robot Imitation of Human Behavior

In light of the above results, we chose to use 32 BSOC nodes for the humans to train their robots. We achieved a mean cycle time of 5.05 cycles/second for learning ($\sigma = 1.13$) and 7.69 cycles/second for applying the learned behavior ($\sigma = 0.25$). (We improved on the cycle time from our previous experiments by not reading the bump sensors.)

Each training run had three phases. In the first phase, the human trained the robot for a specified period of time. In the second phase, the robot was allowed to run autonomously, but the human could intervene and retrain it at any time, for any reason, for any duration. In the third phase, the robot ran autonomously without any further human intervention.

For half of our participants, the first phase lasted 1.5 minutes and the second phase lasted 5.5 minutes. For the others, the first phase lasted 3 minutes and the second phase lasted 4 minutes. The final phase always lasted one minute.

Each of the 16 participants performed three training runs, referred to as Trials 1, 2, and 3. In Trial 1, the human used the same box environment as the self-training experiments. In Trials 2 and 3, the human used our second environment, the cluttered office. In Trials 1 and 2, the robot could go FORWARD or LEFT. In Trial 3, the robot could also go RIGHT.

Figure 7 shows how well the robot learned to imitate the human's behavior. If "Retrained" is "No", it gives the robot's performance after the initial training phase was complete, but prior to the first retraining intervention from the human.

BSOC Nodes vs. Quality of Learning		
Nodes	$\bar{x}$ SSD	σ
2	31.74	17.28
4	11.62	10.36
8	4.40	7.68
16	2.87	4.34
32	3.48	5.84
64	12.20	10.57

Figure 6: BSOC Nodes vs. Quality of Learning

Quality of Learned Human Imitation				
Trial	Retrained?	$\bar{x}$ SSD	σ	max
1	No	9.40	12.95	44.90
1	Yes	1.77	1.77	6.96
2	No	14.24	12.15	46.76
2	Yes	1.72	1.93	7.50
3	No	20.47	16.42	65.56
3	Yes	10.84	15.21	86.79

Figure 7: Quality of Learned Human Imitation

A "Yes" indicates the robot's performance after the final retraining intervention had occurred.

In Trials 1 and 2, where the only movement options are FORWARD and LEFT, the robot exhibited very good imitation of the human behavior after the retraining period was completed. Both the SSD and its standard deviation are very low, indicating both that the behavior is similar and that this result was consistent across all of our experiments. Furthermore, the maximum values for these cases are lower than the mean values without retraining, further reinforcing the observation that good imitation learning was achieved. We consider this solid evidence that our system enables successful imitation learning, provided that retraining is possible.

In Trial 3, where the human was able to instruct the robot to go RIGHT in addition to FORWARD and LEFT, the results were much worse. Even the retraining period was insufficient to bring about reliably good imitation of the human behavior. This is not too surprising given that images of obstacles could be readily associated with instructions to turn in either direction. It does suggest that our use of a histogram with more than two actions does not work and that to accommodate multiple actions, another approach is needed.

Figure 8 shows the impact of the initial training time on the quality of the learned behavior. While the larger time yielded a small benefit on average, the standard deviations were too high to conclude anything definitive.

Results: Survey of Participants

In addition to the quantitative data described above, we also asked each participant to respond to the following statement: "The robot demonstrated that it learned the behavior I demonstrated." The answer used a 7-point Likert scale, with 1 indicating "Strongly Disagree" and 7 indicating "Strongly Agree." Each rating from 1 to 4 was awarded once; there

Training Time vs. Quality of Learning			
Teacher	Time (s)	$\bar{x}$ SSD	σ
Robot	90	12.41	15.47
Robot	180	8.21	11.31
Human	90	11.79	14.98
Human	180	9.38	12.91

Figure 8: Training Time vs. Quality of Learning

were five ratings of 5, two of 6, and five of 7, allowing us to conclude that 75% of participants at least slightly agreed that the robot learned the behavior that they demonstrated. Overall, the subjective impressions of the human participants are consonant with the quantitative results outlined above.

Conclusion and Future Work

In this work, we employed Bounded Self-Organizing Clusters as the basis for a supervised learning algorithm, applied to the task of real-time imitation learning for robot behaviors. We demonstrated both theoretically and experimentally that this system implements a real-time learning algorithm. We further demonstrated that if a human trainer is able to retrain a robot as needed, that this algorithm enables the teaching of a visual obstacle-avoidance behavior with two actions. Furthermore, the relationship between a single hyperparameter (the number of BSOC nodes), the resulting algorithmic time constraints, and the resulting algorithm quality was discussed theoretically and established experimentally.

Much remains to be done to overcome the limitations of the work described in this paper. First, we would like the robot to learn the targeted behavior across multiple environments. These could overwhelm the limited number of nodes available given a target cycle time. To ameliorate this, we plan to investigate the use of preprocessed image features (e.g. (Calonder et al. 2010) (Rublee et al. 2011)) to index the BSOC node. We could train a BSOC networks for each environment, using an index to select the network to be employed for action selection based on the current image.

Additional work will be required to address the brittleness of the system with respect to the number of actions available. One option we are considering is to insert a meta-layer that determines which of several two-action behaviors is to be employed. Some circumstances might suggest that turning left consistently would be advantageous, whereas other circumstances might indicate consistent right turns. This meta-layer could either be hard-coded or could itself be learned.

We plan to investigate other visual behaviors such as wall-following, object tracking, and object pushing. We are also interested in real-time learning in other domains, such as real-time sentiment analysis from social media text feeds.

Our complete implementation is available at `https://github.com/E-R-C/FLAIRS30-BSOC`.

References

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5).

Calonder, M.; Lepetit, V.; Strecha, C.; and Fua, P. 2010. Brief: Binary robust independent elementary features. In *Proceedings of the 11th European Conference on Computer Vision*, 778–792.

Coates, A.; Abbeel, P.; and Ng, A. Y. 2008. Learning for control from multiple demonstrations. In *Proceedings of the International Conference on Machine Learning*.

Ferrer, G. J. 2014. Towards human-induced vision-guided robot behavior. In *Proceedings of the 2014 AAAI Fall Symposium: Knowledge, Skill, and Behavior Transfer in Autonomous Robots*.

Ferrer, G. J. 2016. Real-time unsupervised clustering. In *Proceedings of the 27th Modern Artificial Intelligence and Cognitive Science Conference*, 47–53.

Horswill, I. 1994. Visual collision avoidance by segmentation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 902–909. IEEE Press.

Kohonen, T., and Honkela, T. 2007. Kohonen network. *Scholarpedia* 2(1):1568. revision #122029.

Ontanon, S.; Montana, J. L.; and Gonzalez, A. J. 2014. A dynamic-bayesian network framework for modeling and evaluating learning from observation. *Expert Systems with Applications* 41:52125226.

Peterson, L. E. 2009. K-nearest neighbor. *Scholarpedia* 4(2):1883. revision #136646.

Pomerleau , D. 1991. Rapidly adapting artificial neural networks for autonomous navigation. In Lippmann, R.; Moody, J.; and Touretzky, D., eds., *Advances in Neural Information Processing Systems 3*, 429–435. Morgan Kaufmann.

Provost, J.; Kuipers, B. J.; and Miikkulainen, R. 2006. Developing navigation behavior through self-organizing distinctive state abstraction. *Connection Science* 18(2):159–172.

Rublee, E.; Rabaud, V.; Konolige, K.; and Bradski, G. 2011. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of the IEEE International Conference on Computer Vision*.

Smith, A. 2002. Applications of the self-organizing map to reinforcement learning. *Neural Networks* 15:1107–1124.

Sullivan, K., and Luke, S. 2012. Real-time training of team soccer behaviors. In Chen, X.; Stone, P.; Sucar, L.; and van der Zant, T., eds., *RoboCup 2012: Robot Soccer World Cup XVI*. Springer.

Touzet, C. 1997. Neural reinforcement learning for behavior synthesis. *Robotics and Autonomous Systems* 22(3-4).

Touzet, C. 2006. Modeling and simulation of elementary robot behaviors using associative memories. *International Journal of Advanced Robotic Systems* 3(2):165–170.