Real-time Unsupervised Clustering

Gabriel J. Ferrer
Department of Mathematics and Computer Science
Hendrix College
1600 Washington Ave.
Conway, Arkansas 72032

Abstract

In our research program, we are developing machine learning algorithms to enable a mobile robot to build a compact representation of its environment. This requires the processing of each new input to terminate in constant time. Existing machine learning algorithms are either incapable of meeting this constraint or deliver problematic results. In this paper, we describe a new algorithm for real-time unsupervised clustering, Bounded Self-Organizing Clustering. It executes in constant time for each input, and it produces clusterings that are significantly better than those created by the Self-Organizing Map, its closest competitor, on sensor data acquired from a physically embodied mobile robot.

Clustering algorithms are unsupervised learning algorithms that employ a distance metric to organize their training inputs into clusters. The classification of a previously unseen input is determined by the cluster to which it is closest, again based on the distance metric. This enables a large input space to be compressed into a more compact representation. A useful application of this technique is to create models of mobile robot environments based on the robot's sensor inputs.

In both training and classification, the distances between the input and the representative example of each cluster must be calculated. By placing a fixed upper bound on the number of clusters, we limit the total number of calculations required by each training and classification operation. This, in turn, enables us to predict the cycle time of the algorithm, a necessity for the practical deployment of such an algorithm aboard a mobile robot.

In this paper, we introduce a novel clustering algorithm, Bounded Self-Organizing Clusters (BSOC), that is designed to meet these constraints. In the robotics literature, the Kohonen Self-Organizing Map (SOM) has been used for this purpose as well. We argue that the design of the SOM leads to a relatively poor clustering, and that BSOC represents a significant improvement. We demonstrate this experimentally using both sonar and image data captured from a mobile robot.

We begin with some background and describe related work. Next, we describe our physical system hardware, with an eye towards making clear the technical limitations to which any solution must conform. We then describe our learning algorithm, followed by our experiments and results. We then give our conclusions and speculations on future work.

Background

Clustering algorithms have long been an appealing machine learning technique for robotics applications, due to their potential for incremental machine learning in combination as well as their ability to reduce complex continuous sensor input values to a smaller number of discrete values. Several examples of this approach employ the Kohonen Self-Organizing Map (SOM) (Kohonen 2001) to reduce the sensor inputs into a finite number of discrete states to facilitate the Q-learning algorithm.

Each SOM is a neural network consisting of a single two-dimensional layer of nodes arranged in a rectangular grid. The number of nodes and their arrangement is fixed. Each node is an object of the same specifications as a network input, which represents the reference input for that node. A distance metric is defined for the space of possible inputs. Typically, this distance metric will be the sum-of-squared-differences. The output node whose reference input that is closest in distance to an input presented to the network is considered the winning node; its coordinates are the output of the SOM.

To train a SOM, a training input is presented to the network. The winning output node's reference input, along with some of its neighbors, is then combined with the input value. The difference vector between the values is calculated and multiplied by a learning rate. The resulting value is then added to the reference input of the learning node. For each affected SOM node i and input vector element j, the update equation for the weight vector w of the reference input (given a learning rate $\alpha, 0 < \alpha < 1$) is:

$$w_{ij} = w_{ij} + \alpha * (input_{ij} - w_{ij})$$

The α parameter is determined by the neighborhood scheme of the specific SOM implementation. Many implementations (e.g. (Smith 2002)) make multiple passes over the input, slowly reducing α on each pass. (Touzet 1997) trains the winning node with $\alpha = 0.9$, and the four cardinal neighbors are trained with $\alpha = 0.5$. As α remains constant throughout, this formulation of the SOM lends itself well to

real-time implementation aboard a robot. Since the number of nodes and topology are fixed, one can define a SOM output map that guarantees that the robot's cycle time will reliably meet its timing requirements. This will be the variation of the SOM that will serve as a baseline in our experiments for assessing our own algorithm.

The earliest application of the SOM to robotics was that of (Touzet 1997), in which a tiny (6 cm diameter) Khepera mobile robot equipped with eight infrared sensors for object detection learned to move forward while avoiding obstacles. The continuous values from the IR sensors were routed into a 4x4 SOM. The SOM output node was then employed as a state index for Q-learning. Touzet later scaled up this work to a much larger robot (Touzet 2006) employing a ring of 16 sonars. In this later work, Q-learning was abandoned; instead of using reinforcement rewards, the desired behavior was specified in terms of an acceptable range of sensor values. (Smith 2002) did similar work, with two major differences: a different definition of the SOM learning neighborhood than that of Touzet, and implementation in simulation rather than on a physical robot. Instead of using immediate neighbors, all nodes in the network were affected, with the learning rate combined with a Gaussian smoothing function. (Ferrer 2010) implemented both approaches on a mobile robot and compared the results. He found that both approaches worked respectably well, but that both were surpassed by Q-Learning without the use of clustering.

(Provost, Kuipers, and Miikkulainen 2006) undertook similar work in simulation, but replaced the SOM with the closely related Growing Neural Gas (GNG) algorithm (Fritzke 1995). The GNG is appealing in this context because of its flexible topology. It starts with two nodes. Additional nodes are incorporated into the network as needed to represent inputs that correspond to significant previously-unseen aspects of the input space.

(Ferrer 2014a) also applied the GNG algorithm to robot learning. A robot built a GNG as it drove through its environment. A human then examined each GNG node, specifying an appropriate action based on a visual inspection of the node's reference input. This work was reasonably successful for behavior specification on a physical robot. However, the manual specification of an action for each node was found to be laborious. (Ferrer 2014b) further advanced the concept of employing a GNG for learning an environment. A human pilots a robot through an environment, learning a GNG representation of the environment and associating actions from the human with the GNG nodes. When running autonomously, the robot selects its action based on which GNG node is the winner for the current input image.

Because the GNG adapts the number of clusters based on the inputs it receives, it is not possible to guarantee that the robot will complete each round of processing within a specified target time. While one appealing aspect of the GNG algorithm is the elimination of underutilized nodes, the specifics of this elimination process are too unpredictable to enable us to make guarantees about timing.

The problem of learning from a real-time input stream is also of interest outside the robotics community. (Hapfelmeier et al. 2012) mention application domains such as real-time scientific data processing, finance, web applications, and telecommunications.

Figure 1: Robot

The most prominent clustering algorithm in the literature is arguably k-means (Forgey 1965) (MacQueen 1967). For our application domain, we did not consider this approach at all, as it requires multiple passes through the input data. As each of these multiple passes requires visiting every input sample, our target constraint of constant-time execution cannot be satisfied. Also related to our approach is bottom-up agglomeration as described, for example, by (Koivistoinen, Ruuska, and Elomaa 2006). We do not seek to perform a full bottom-up agglomeration; instead, we use agglomeration as a strategy to help us execute in constant time while retaining good relationships with the inputs.

Hardware

Our robot is a Lego Mindstorms EV3 equipped with a Logitech C270 webcam and three sonar sensors. The EV3 has a 300 MHz ARM9 CPU and 64 MB of RAM. Its OS is a version of Linux. We implemented our software in Java using LeJOS 0.9.0. The webcam is connected to the EV3 via its USB 1.1 port. As the webcam is designed to be used with a USB 2.0 port or better, it is limited to grabbing 160x120 YUYV images at about 14 frames per second. We configured the robot as shown in Figure 1.

Bounded Self-Organizing Clusters

Basic Version

We introduce a novel clustering algorithm called *Bounded Self-Organizing Clusters* (BSOC) (see Figure 2). Each BSOC is a complete undirected weighted graph. Each node in the graph contains the reference input for a cluster (comparable to the nodes in a Self-Organizing Map (Kohonen 2001)). Each edge weight denotes the distance between the reference inputs of the clusters.

Each BSOC is given a maximum number of clusters it can represent. When training, each training input is added to the

set of clusters as a reference input. If adding the training input causes 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 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.

```
setup(max-nodes)
  edges = new red-black-tree
  nodes = map of node-indexes to inputs

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

train(input)
  insert(input)
  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))

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

merged(n1, n2)
  img = (n1.image + n2.image) / 2
  return img
```

Figure 2: Pseudocode for Basic BSOC Training

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, $2M$ edge removals (each requiring $2 \log M$ comparisons in a red-black tree), and two hash table removals. Consequently, given that M can be fixed as a constant, we can say that each call to `train` or `lookup` takes constant time.

Weighted-Input Variation

A potential drawback of the above algorithm is that clusters that are the result of repeated averages are given the same importance in the merging stage as clusters that are the result of single inputs. We find it intuitive that clusters that are the result of repeated merges would be closer in distance to a larger number of inputs from the input space. The following variation is an attempt to codify this intuition.

In addition to a reference input, each node in the graph contains a counter, initially set to one. Each counter denotes the number of input samples that contributed to the current node. These counters serve two major roles. First, they will be used as weights when merging nodes. Second, each edge weight (i.e., the distance between the reference inputs of the connected nodes) will be multiplied by the larger of the two counters for the connected nodes.

We expect these roles to achieve two things. First, by employing the larger counter as an edge weight multiplier, nodes that are derived from larger numbers of input samples will be less likely to be selected for merging. Second, by weighting the merge based on the counters, when they are selected they will preserve more of the broad influence of the inputs from which they were generated.

The revised BSOC algorithm is given in Figure 3. Note that `setup` and `lookup` are unchanged (and hence omitted). The additional arithmetic calculations have no asymptotic impact on performance.

```
train(input)
  insert(input, 1)
  if number of nodes exceeds max-nodes
     edge = edges.remove(smallest edge)
     (n1, n2) = endpoints of edge
     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 3: Pseudocode for Weighted BSOC Training

Experiments and Evaluation

Robot Environment

We gathered two different types of sensor data from our robot: triples of distance values from the sonars, and images from the webcam. To gather the sonar data set and the first

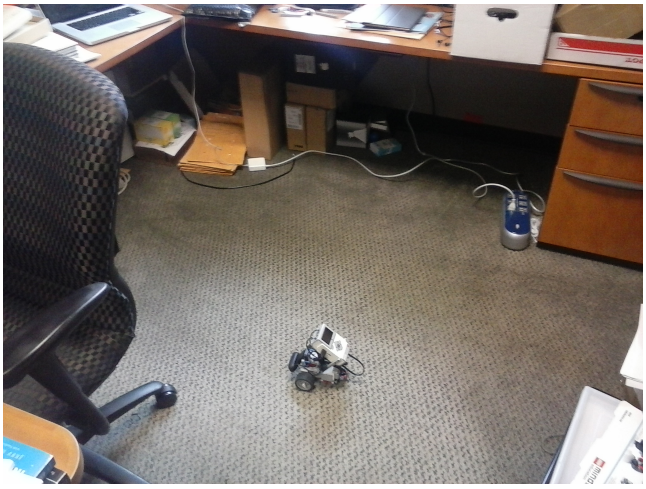

Figure 4: Office, Carpeted Floor

Figure 5: Office, Tile Floor

video data set, we navigated the robot through the cluttered office environment depicted in Figure 4.

For the second video data set, we navigated the robot through a different office environment. While the first environment has a carpeted floor, the second environment has a tile floor of alternating colors, as we can see in Figure 5.

Each video sequence is 700 frames. Each frame is a 160x120 image in YUYV format. As YUYV uses 2 bytes per pixel, the total size of each frame is 38400 bytes. The sonar data set was created from a sequence of 1304 sonar triples. The sonars are configured as follows: One points about 45 degrees to the left of the robot, one points forward, and one points about 45 degrees to the right. The effective range of each sonar is from 0 to 2.5 meters. In all cases, the robot visited many different areas of the office, so as to ensure that each sequence was reasonably representative of the environment.

For both types of data, distance values between samples are calculated as sum-of-squared differences. For computing the distance between two triples of sonar distance values, the differences between the distance values of each corresponding sonar position are computed. For the YUYV images, the differences between each corresponding byte are computed.

When creating clusters, the merging process requires averaging the inputs. In both cases, the averaging is analogous to the computation of the distance. For the sonar triples, the weighted mean values are computed for each sonar position. For the images, the weighted mean values are computed for each byte. Since the byte values are 8-bit unsigned integers, the result of each division is truncated.

	Sonar (m^2)		
Nodes	SOM	BSOC1	BSOC2
16	1.812×10^3	2.310×10^1	2.451×10^1
32	1.523×10^3	4.978	2.998
64	7.361×10^1	9.120×10^{-1}	5.473×10^{-1}

Figure 6: Sonar: SSD

Experiments

In this section, the basic version of BSOC is denoted `BSOC1`, and the BSOC variation weighted by input sample counts is `BSOC2`.

We constructed our experiments to test the following hypotheses:

1. Given the same number of clusters, BSOC1 will provide closer matches to its source inputs than the SOM as formulated by (Touzet 1997), and BSOC2 will provide closer matches than BSOC1. We define "closer matches" to be the sum-of-squared-differences (SSD) between each input and the reference input for the closest matching cluster.
2. Given the same number of clusters, BSOC2 will exhibit more consistent performance than the SOM or BSOC1 regardless of the ordering of the input data. We measure "consistent performance" as the ratio of the standard deviation to the mean. Smaller values indicate more consistent performance.

We tested each of our input sequences with clustering algorithms configured with 16, 32, and 64 nodes. For testing the SOMs, we used 4x4, 4x8, and 8x8 grids. The SSD values for the sonar experiments are in units of meters squared. For the video experiments, the units are pixel intensity values squared. For presentation purposes, all entries have been rounded to four significant figures.

Results

Hypothesis 1 As we can see from Figures 6 through 11, Hypothesis 1 was confirmed, with some qualifications. BSOC1 consistently outperformed the SOM in the sonar experiments by multiple orders of magnitude, a compelling improvement. In the first video experiments, it outperformed the SOM by factors ranging from slightly over one to slightly over two, a much less compelling improvement, but still consistently superior. The second video was more favorable, performing 3-4 times better.

Sonar Ratio		
Nodes	BSOC1	BSOC2
16	7.844×10^{1}	7.393×10^{1}
32	3.059×10^{2}	5.080×10^{2}
64	8.071×10^{1}	1.345×10^{2}

Figure 7: Sonar: SOM SSD / BSOC SSD

Video 1			
Nodes	SOM	BSOC1	BSOC2
16	5.465×10^{18}	4.749×10^{18}	1.754×10^{18}
32	4.257×10^{18}	2.524×10^{18}	1.113×10^{18}
64	2.703×10^{18}	1.125×10^{18}	6.765×10^{17}

Figure 8: Video 1: SSD

BSOC2 outperformed both of the other algorithms very consistently, with the exception of the 16-node sonar experiment, in which its performance was very close to BSOC1. In the other two sonar experiments, it outperformed BSOC1 by about 65%.

In the video experiments, BSOC2 consistently outperformed both the SOM and BSOC1. In the first video, it was about 3-4 times better than the SOM, while with the second video it was 8-16 times better.

All of these clustering algorithms show significant improvement in performance as additional nodes are added. This demonstrates that those engineering a robotic system incorporating a clustering algorithm do have a clearly defined trade-off between cycle time and clustering accuracy. This improvement was less consistent with the second video stream, as the 32-node solution was more favorable to BSOC than the 64-node solution.

Hypothesis 2 We ran a separate set of experiments to test Hypothesis 2, as all of the previous experiments trained the learner by showing the inputs in the order that they were acquired by the robot. While we consider that to be the expected deployment situation of these algorithms, the sensitivity to ordering is also worthwhile to measure. To this end, we trained the 64-node version of each algorithm on six different randomly generated permutations of the input samples. We employ here the same performance metric as our earlier experiments, namely, the sum-of-squared-differences between each input and its closest matching cluster. Figure 12 shows the mean, standard deviation, and ratio for the sonar experiments, and Figures 13 and 14 show the same values for the video experiments.

In all three cases, BSOC2 had a much tighter standard deviation than the other algorithms, showing a more consistent performance regardless of move ordering. In spite of this clear result, in all cases the performance on the non-permuted input is much worse. Figures 15, 16, and 17 highlight this phenomenon. For all of these algorithms, processing the inputs in the order they were acquired creates a highly idiosyncratic statistical outlier.

Video 1 Ratio		
Nodes	SOM/BSOC1	SOM/BSOC2
16	1.151	3.116
32	1.687	3.825
64	2.403	3.996

Figure 9: Video 1: SOM SSD / BSOC SSD

Video 2			
Nodes	SOM	BSOC1	BSOC2
16	1.404×10^{19}	3.739×10^{18}	1.199×10^{18}
32	5.390×10^{19}	1.806×10^{18}	6.745×10^{17}
64	4.554×10^{18}	1.058×10^{18}	2.838×10^{17}

Figure 10: Video 2: SSD

Video 2 Ratio		
Nodes	SOM/BSOC1	SOM/BSOC2
16	3.75	1.17×10^{1}
32	2.98	7.99
64	4.30	1.60×10^{1}

Figure 11: Video 2: SOM SSD / BSOC SSD

Potential Applications

We are in the process of developing several different robot learning systems that would employ BSOC as a component. Our first goal is to replicate the learning scheme described by (Touzet 2006). He employed two different SOMs to represent the robot's environment as sensed by a ring of sonars. One SOM was used to represent the environment itself, while the second SOM was used for modeling the result of action selection. The robot's goals are specified in terms of sensor values. The robot then attempts to move itself to a SOM state that is compatible with those values. We have found that while the underlying concept is compelling, there is not sufficient detail to represent the role of the second SOM. We believe that replacing the first SOM with a BSOC will result in improved modeling accuracy, while replacing the second SOM with a Markov chain should prove

Permuted sonar			
Algorithm	$\bar{x}$	σ	$\frac{\sigma}{\bar{x}}$
SOM	4.585×10^{1}	1.008×10^{1}	2.200×10^{-1}
BSOC1	6.695×10^{-1}	2.043×10^{-1}	3.051×10^{-1}
BSOC2	2.477×10^{-1}	2.717×10^{-2}	1.097×10^{-1}

Figure 12: Sonar, with permuted inputs

Permuted video 1			
Algorithm	$\bar{x}$	σ	$\frac{\sigma}{\bar{x}}$
SOM	1.27×10^{18}	1.455×10^{17}	1.15×10^{-1}
BSOC1	9.82×10^{17}	1.990×10^{17}	2.03×10^{-1}
BSOC2	2.84×10^{17}	6.421×10^{15}	2.26×10^{-2}

Figure 13: Video 1, with permuted inputs

Permuted video 2			
Algorithm	$\bar{x}$	σ	$\frac{\sigma}{\bar{x}}$
SOM	1.13×10^{18}	1.455×10^{17}	1.29×10^{-1}
BSOC1	3.74×10^{17}	4.835×10^{16}	1.29×10^{-1}
BSOC2	1.41×10^{17}	8.661×10^{15}	6.12×10^{-2}

Figure 14: Video 2, with permuted inputs

Sonar: Comparisons of SSD			
Algorithm	Original	Permuted $\bar{x}$	$\frac{Original}{Permuted}$
SOM	7.361×10^{1}	4.585×10^{1}	1.606
BSOC1	9.120×10^{-1}	6.695×10^{-1}	1.362
BSOC2	5.473×10^{-1}	2.477×10^{-1}	2.209

Figure 15: Sonar: Comparisons of SSD

Video 1: Comparisons of SSD			
Algorithm	Original	Permuted $\bar{x}$	$\frac{Original}{Permuted}$
SOM	2.702×10^{18}	1.27×10^{18}	2.12
BSOC1	1.125×10^{18}	9.82×10^{17}	1.14
BSOC2	6.765×10^{17}	2.84×10^{17}	2.38

Figure 16: Video 1: Comparisons of SSD

Video 2: Comparisons of SSD			
Algorithm	Original	Permuted $\bar{x}$	$\frac{Original}{Permuted}$
SOM	4.553×10^{18}	1.13×10^{18}	4.02
BSOC1	1.058×10^{18}	3.74×10^{17}	2.83
BSOC2	2.838×10^{17}	1.41×10^{17}	2.01

Figure 17: Video 2: Comparisons of SSD

effective. We plan to employ Dijkstra's Algorithm to find paths in the Markov chain through the BSOC states that it discovers.

Once the above system proves workable with sonars, we plan to employ video as the primary input. We believe this will enable us to do two things. First, we hope to be able to define robot behaviors in terms of video input, potentially opening up a wider array of possibilities than those given by sonar. Second, we would like to investigate the possibility of using the nodes for small-scale navigation of an environment.

We are also interested in using BSOC as a component to a supervised-learning system. In this way, BSOC would enable a scheme along the lines of k-nearest-neighbor (Saunders, Nehaniv, and Dautenhahn 2006) to be implemented in constant time.

We speculate that this algorithm could be useful for informing human decision-makers in situations requiring rapid responses where sensors are widely distributed and submitting information. Examples of such scenarios could include environmental assessment in the wake of a disaster or any number of military or law-enforcement scenarios[1].

Conclusion

For the purpose of modeling a robot environment in constant time using a sequence of sensor inputs, we have shown that Bounded Self-Organizing Clusters (BSOC) is a compelling alternative to the Kohonen Self-Organizing Map, the existing approach, for both sonar and video inputs. The variation of BSOC that weights the results based on the input sources is a particularly clear winner in our experiments; we consider this to be the definitive variation of the algorithm.

We believe that improved clustering has great potential to improve robot learning as a component of several possible larger systems. We would also be interested in finding applications for this algorithm in other domains beyond robotics that would benefit from real-time clustering.

Significant additional work remains in the development of this algorithm. First, many robot vision systems employ preprocessed features rather than raw pixel data, with the goal of improving recognition robustness across many different robot poses. We plan to investigate using ORB (Rublee et al. 2011) for this purpose. Second, many practical learning situations involve supervised learning. We believe that BSOC could be used as a component of a real-time supervised learning system designed along the lines of k-nearest-neighbor. Third, a system that utilizes BSOC clusters and maintains references to those clusters will need to preserve information as clusters merge. We currently have a prototype system in place to address this employing the Observer design pattern, but have not yet conducted an assessment of its utility.

Anyone who would like to experiment with our code and data sets is welcome to do so. It is archived at `https://github.com/gjf2a/maics2016`. The code is in the public domain.

[1]We would like to thank one of our anonymous reviewers for suggesting these possibilities.

Acknowledgements

We would like to thank the three anonymous reviewers for many helpful comments on our initial submission. We would also like to thank Mark Goadrich for proofreading this paper and giving many helpful suggestions, most specifically the idea to assess performance based on random permutations of the inputs.

References

Ferrer, G. J. 2010. Encoding robotic sensor states for Q-Learning using the self-organizing map. *Journal of Computing Sciences in Colleges* 25(5).

Ferrer, G. J. 2014a. Creating visual reactive robot behaviors using growing neural gas. In *Proceedings of the 25th Modern Artificial Intelligence and Cognitive Science Conference*, 39–44.

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

Forgey, E. 1965. Cluster analysis of multivariate data: Efficiency vs. interpretability of classification. *Biometrics* 21(768).

Fritzke, B. 1995. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*, 625–632. MIT Press.

Hapfelmeier, A.; Mertes, C.; Schmidt1, J.; and Kramer, S. 2012. Towards real-time machine learning. In *Proceedings of the ECML PKDD 2012 Workshop on Instant Interactive Data Mining*.

Kohonen, T. 2001. *Self-Organizing Maps*. Springer, 3rd edition.

Koivistoinen, H.; Ruuska, M.; and Elomaa, T. 2006. A Voronoi diagram approach to autonomous clustering. In *Proceedings of the 9th International Conference on Discovery Science*, 149–160.

MacQueen, J. 1967. Some methods for classification and analysis of multivariate observations. In *Proc. of the Fifth Berkeley Symposium on Math. Stat. and Prob.*, 281–296.

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*.

Saunders, J.; Nehaniv, C. L.; and Dautenhahn, K. 2006. Teaching robots by moulding behavior and scaffolding the environment. In *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-robot Interaction*, HRI '06, 118–125. New York, NY, USA: ACM.

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

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.