Creating Visual Reactive Robot Behaviors Using Growing Neural Gas

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

Abstract

Creating reactive robot behaviors that rely solely on visual input is tricky due to the well-known problems involved with computer vision. This paper presents a potential solution to this problem. The robot builds representations of its target environment using Growing Neural Gas. The robot programmer then specifies its behavior for each learned node. This approach is shown to have the potential to be effective for simplifying robot behavior programming in an indoor environment with low-cost hardware.

Programming reactive behaviors (Brooks 1986) on a robot equipped with basic sensors such as touch sensors and sonars is reasonably straightforward and well-understood. Despite considerable progress in computer vision techniques, use of camera images to direct reactive behaviors remains tricky. Considerable effort is often invested in computer vision algorithms that are specialized for a particular environment. (Horswill 1994)

Unsupervised learning algorithms have proven to be popular tools for computer vision, especially neural network models such as the Self-Organizing Map (Kohonen 2001) and Growing Neural Gas (Fritzke 1995). These algorithms derive abstractions of subsets of image sets that can be used for later classification of previously unseen images.

This paper describes an application of unsupervised learning, specifically Growing Neural Gas (GNG) to the problem of programming reactive behaviors when using computer vision as the principal sensor for a mobile robot. Behavior programming is a two-stage process. First, the robot builds a GNG network as it explores its environment. Subsequently, the programmer specifies the desired action for each learned GNG cluster. Once this specification is complete, the robot selects its actions based on the GNG cluster that is the closest match to the current input image.

Learning the Environment

Unsupervised Learning

Unsupervised learning algorithms, such as k-means (Forgey 1965) (MacQueen 1967), the self-organizing map (SOM) (Kohonen 2001), and Growing Neural Gas (GNG) (Fritzke 1995), operate by partitioning their training inputs into clusters based on a distance metric. Each cluster is defined as a representative example of the input space. Each representative example is arithmetically derived (in an algorithm-dependent manner) from the training inputs.

Figure 1: Portrait of the Robot

Both k-means and self-organizing maps require a fixed number of clusters to be specified in advance. Growing neural gas adaptively determines the number of clusters based on its inputs. In the interest of allowing the inputs to determine the number of clusters, we selected GNG for this task.

Growing Neural Gas

Growing Neural Gas is an artificial neural network that is trained using an unsupervised learning algorithm. The network is an undirected weighted graph. In this implementation, both the network inputs and the network nodes are 2D grayscale images. When an input is presented to the network, the Euclidean distance between the input and each node is calculated. The node with the shortest distance relative to the input becomes the active node.

Each edge connects two nodes that are active in response to similar inputs. The edge weight reflects the frequency with which the pair has been responsive in tandem; it is reset to zero whenever this occurs. Large weights denote weak connections that are eventually purged. Nodes who lose all

their edges are purged as well.

Training In each iteration of training, a single input is presented to the network. The network is then adjusted as follows:

- Identify the nodes that are the closest and second-closest matches to the input.
- Adjust edge weights.
- Update error and utility values.
- Create a new node.
- Purge edges and nodes.

In the training process, two nodes are identified: the node with the shortest distance to the input, and the node with the second-shortest distance. If an edge is present between them, its weight is reset to zero; otherwise, a new edge is created between them with a weight of zero. All other edges adjoining the winning node are increased by one.

The values of each image pixel p_{xy} for the winning node and each of its neighbors are updated as follows relative to each input pixel q_{xy}:

$$p_{xy} = p_{xy} + \alpha(q_{xy} - p_{xy})$$

The learning rate parameter α is significantly larger for the winning node in comparison to the lower value used for the neighboring nodes.

Error and Utility Each node maintains *error* and *utility* values. The purpose of the error value is to identify nodes that, while they are frequently the best matching node for a variety of inputs, are still not very close matches. These nodes, then, represent parts of the input space that are not adequately covered by the current set of nodes. The purpose of the utility value is to identify nodes that are distinctive relative to their neighbors. Nodes with high utility are frequently much closer matches to many inputs than their neighbors.

On each training iteration, these values are updated as follows:

- For the winning node:
  - Increase the error for the winning node by the Euclidean distance to the input.
  - Subtract the distance to the winning node from the distance to the second-best node. Add this difference to the utility.
- For each node n:
  - Reduce the error and utility values using a decay constant β ($0 < \beta < 1$) as follows:
    - $error_n = error_n - \beta \times error_n$
    - $utility_n = utility_n - \beta \times utility_n$

Creating New Nodes An integer parameter λ controls the creation of new nodes. The frequency of node creation is inversely proportional to lambda; low values imply frequent introduction of new nodes.

Every λ iterations, a new node is created as follows:

- Find the node m with the largest error value in the network.
- Find its neighbor n with the largest error value among all of m's neighbors.
- Create an image by averaging the corresponding pixel values of m and n.
- Create a new node p using:
  - The averaged image
  - The mean of the errors of m and n
  - The maximum of the utility values of m and n
- Break the edge between m and n.
- Add an edge of weight zero between m and p, and another between n and p.
- Divide the error and utility values for each of m and n by two.

Purging Edges and Nodes On each iteration, every edge whose weight exceeds a specified limit is purged. If any node has all its edges purged, that node is purged as well.

In addition, for every node the *utility ratio* is calculated (Fritzke 1997). The largest error of any node, e_{max}, is determined. The utility ratio for each node n with utility u_n is $\frac{e_{max}}{u_n}$. The node with the single largest utility ratio is the *most useless node*. If its ratio exceeds a parameter k, it is purged.

Training and Programming

Our robot is equipped with a controller that allows the programmer to drive it around its environment. As the robot is driven around, the first two images it acquires become the first two nodes of Growing Neural Gas. Each subsequent image acquired is used for one iteration of training the GNG network. Training ends upon a signal from the programmer. The GNG network is then saved for later use.

We slightly modified how the GNG learning algorithm decides to create new nodes. When the programmer changes the robot's command, it is typically in response to a stimulus the programmer perceives. Consequently, whatever the robot is sensing at that time has the potential to be very important. Following this intuition, a new GNG node is created and λ is reset to zero whenever the programmer changes the robot's movement.

When the GNG training process is complete, the programmer runs an application that shows all of the GNG nodes. Each node is annotated with a GUI component that allows the programmer to select an action to correspond to that node. A screenshot of the action selection application is given in Figure 2. When the programmer has selected an action for every node, a controller is generated. When the controller executes, as each image is acquired it is presented to the GNG network. The action specified for the winning node is immediately executed.

Experiments

Configuration and Training

The experimental goal was to train the robot to be familiar with a particular room, such that it could wander the room without hitting anything. The robot is a Lego Mindstorms

Figure 2: Selecting Actions for GNG Nodes

NXT. To enable image processing, a Dell Inspiron Mini Netbook equipped with a webcam was placed atop the robot to control it via a USB cable. The configured robot is shown in Figure 1.

The webcam images were acquired at a size of 640x480 pixels. Each image was averaged to produce a grayscale image upon acquisition, with each pixel value ranging from 0 to 255. Each grayscale image was scaled down to 160x120 prior to being applied as an input to the GNG network. The first two GNG nodes are the first two images acquired. This configuration enabled images to be acquired and processed and learning to occur at about 15-16 frames per second. Given a robot velocity of 17 cm/s, it processes about one frame per centimeter of travel.

The GNG algorithm was parameterized as follows:

- Maximum edge age: 100
- Learning rate (winning node): 0.05
- Learning rate (neighboring nodes): 0.0006
- Maximum utility ratio (k): 4
- Iterations per node creation (λ): 200
- Error decay (β) per iteration: 0.0005

Most of the parameters were taken directly from the experiments described by (Holmstrom 2002). As noted above, every change of command results in the creation of a new node. The λ value of 200, then, was selected to ensure the creation of a new node after, at most, two meters of travel.

The GNG network was trained by driving the robot around the room for several minutes. The trained network had 26 nodes (depicted in Figure 2) by the completion of the run.

Action Selection

Actions were selected by visually examining the reference images for each GNG node and determining an appropriate action for the match. To keep things simple, only three actions were used: `FORWARD`, `BACK_LEFT`, and `BACK_RIGHT`.

For some reference images, the choice of action was clear. In Figure 3, we see an example where going forward is the obvious choice, as there is plenty of clear floor space. Figure 4 is an example of a clear choice of a turn, as the wall is very close.

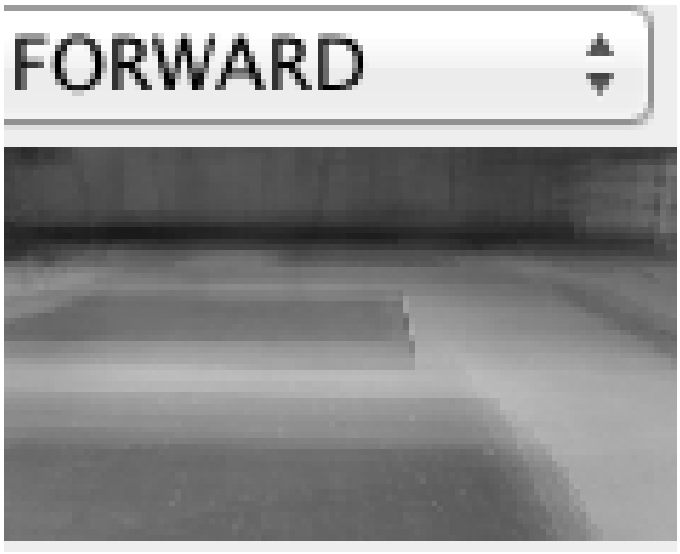

Figure 3: Forward

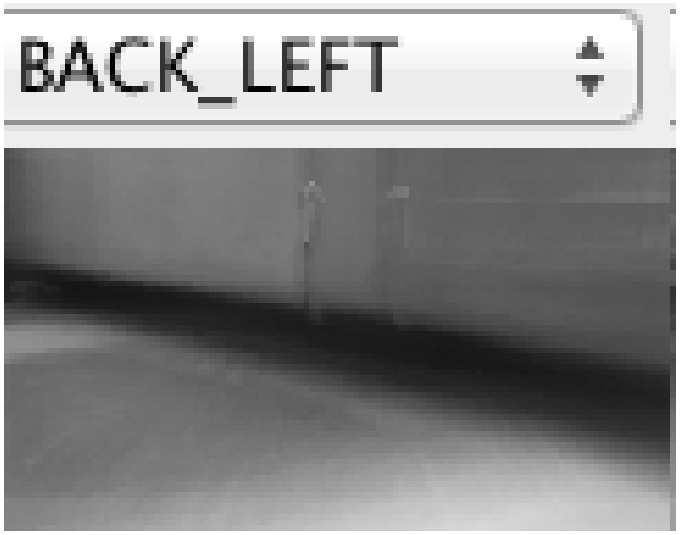

Figure 4: Turn

In other cases, the choice was less clear, and it had to be altered after experimentation. The action for Figure 5 was originally `FORWARD`, but it was changed to a turn after it was

observed that moving forward in that situation led to a collision. The action for Figure 6 was originally `BACK_LEFT`, but it was changed to `FORWARD` when it was observed that this action caused the robot to turn in the middle of the room. Subsequent reflection led to the conclusion that the reference image displays considerably more floor space than was originally thought when the first action was selected. Overall, four out of the 26 actions were changed during the course of experimentation.

Figure 5: Forward, later revised

Figure 6: Turn, later revised

Observed Robot Behavior

For the GNG network shown in Figure 2, three revisions were made to the original action selection, resulting in four total versions. (The actions shown in Figure 2 represent the final version.) The first version produced a robot behavior that countered virtually every forward motion with a backward motion. The result was a robot that rarely hit anything, but also made very little progress. Gradual refinement of the action selection produced a robot (Version 4) that avoided obstacles successfully while maintaining consistent forward motion. The GNG-based controller runs at about 10 frames per second, somewhat slower than the training phrase.

Figure 7 illustrates the effect of refined action selection on forward motion. The vertical axis denotes the percentage of the time that the robot spent moving forward rather than turning. For the first three versions, the data is the average of two runs. (After two runs, the observed behavior was found sufficiently frustrating that actions were altered immediately in light of the observed behavior.) For the considerably more satisfying fourth version, the average of seven runs is given. The "Human" value denotes the percentage of forward motion for a single run of a human piloting the robot, maximizing forward motion while avoiding obstacles.

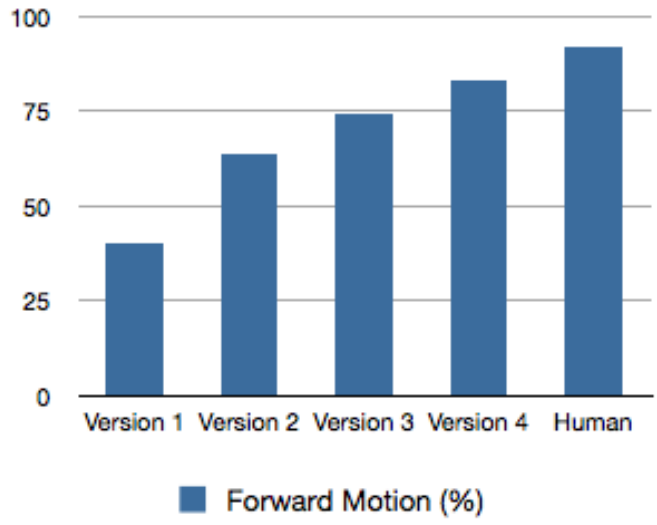

Figure 7: Forward Motion

Figure 8 shows the number of seconds the robot was able to run before striking an obstacle. All runs were for the fourth and final version of action selection depicted in Figure 2. For each run, the robot started moving from the same position in our lab. While it generally did well in avoiding obstacles, there were a couple of locations that consistently caused problems. The long duration for the second run reflects a situation in which the robot was fortunate to avoid the problematic areas for a considerable period of time.

A representative example of a problematic node is node 24, depicted in Figure 9. The lower part depicts open floorspace, while the upper part shows artifacts of several obstacles. Nodes 3, 10, and 16 exhibit similar ambiguities and caused similar problems. Node 18 (in Figure 3) was normally a strong contributor to forward motion; however, it appears that the checkerboard pattern on the floor may have led even this node into ambiguous situations. Node 17 exhibited

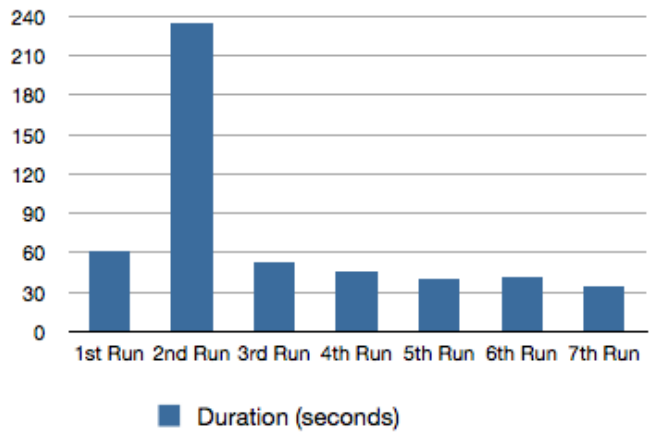

Figure 8: Time until collision

the same issue. The most ironic example is Node 2 (in Figure 6), which had originally been designated as a `BACK_LEFT` node but which was changed to `FORWARD` due to unwanted turns. Node 2 is a combination of floor and wall with a very ambiguous boundary; not all of the turns it triggered turned out to be ”unwanted”.

Figure 9: Node 24

What all of these nodes have in common is that they were otherwise used heavily to trigger productive forward motion. Unfortunately, several locations in our environment proved consistently problematic due to these ambiguities. Giving all of these nodes turn-action designations would have resulted in a significant loss of forward motion. We hypothesize that the underlying problem is that the GNG network needs more nodes to overcome these ambiguities.

Analysis

The results of this preliminary study are very promising. As long as the robot was not exposed to problematic locations, it avoided obstacles perfectly while maintaining significant forward motion. Devising and improving the action configuration proved to be straightforward; simple tweaks produced significant incremental performance improvements. We are considering the following avenues to improve performance.

Merging GNG Networks

Relying upon a single training run to produce a usable GNG interferes with what should naturally be an iterative process of carefully debugging the robot's behavior. By using the initial GNG network as a starting point, additional training runs could be conducted in the physical vicinity of problematic locations in order to help introduce new nodes to overcome ambiguities.

A related approach would be to incorporate the ability to merge GNG networks derived from separate training runs. This would be an alternative means to enable the programmer to “patch” the robot's knowledge of troublesome areas while still retaining the positive aspects of the initially created network.

Implicit Action Selection

An alternative to manual action selection would be a two-phase piloting approach. In the first piloted run, the GNG network would be built. During the second run, popularity of actions selected by the pilot would be tracked for each winning node. In this way, action selection would be implicit rather than explicit. This would be especially useful in allowing this technique to scale to GNG networks with significantly larger numbers of nodes.

Furthermore, this could enable the early identification of ambiguous nodes. Nodes that map onto locations with clear actions would exhibit uniformity in the actions selected on their behalf. Nodes with conflicting action designations could serve as an early signal for trouble, and a focus for efforts to improve the underlying GNG network.

Related Work

Growing Neural Gas has been applied to robotic systems for numerous purposes beyond that proposed in this paper, including localization (Baldassarri et al. 2003) (Yan, Weber, and Wermter 2012), modeling the physical structure of the environment (Kim, Cho, and Kim 2003), (Shamwell et al. 2012), and control via gesture recognition (Yanik et al. 2012).

A representative example of a system that uses artificial neural networks to simplify the programming of reactive behaviors is (Cox and Best 2004). In their system, the programmer specifies motor settings for various combinations of sensor values that a simulated robot encounters. A multilayer perceptron is employed to generalize the motor settings to sensor combinations that had not been previously encountered. The simulated sensors are three infrared sensors and two bump sensors. It is not at all clear how this approach would scale to the much greater complexity of visual input. Our work, by exploiting the inherent intelligibility of the reference images of the GNG clusters, provides the programmer with meaningful abstractions of the visual input for which actions can then be coherently specified.

(Touzet 2006) employs self-organizing maps (Kohonen 2001) to build models of a physical robot's environment and the effect of performing actions in certain states. The desired behavior is then specified in terms of targeted sensor values. For the range sensors they employed, specific windows of target values are specified in order to induce the target behavior. When using computer vision as a sensor, creating such windows of target values is impractical. The GNG cluster reference images provide a usable alternative.

In (Gunderson and Gunderson 2008), an agent architecture is presented that is centered around the issue of reification. Each object to be perceived is represented by a `PerCept` that binds a sensor-derived signature to a symbolic component that can be manipulated by the symbolic reasoning engine. The authors describe how a chair, for example, can be described based on a pattern of readings from multiple sonars. When using camera input, the scheme described in this paper could provide a useful means of defining sensor signatures for objects that are to be visually identified, by assigning identification tags to cluster reference images rather than actions.

Conclusion

Growing Neural Gas is an effective technique for creating a model of a robot's environment that simplifies the programming of reactive behaviors relying on visual input. It runs sufficiently fast for real-time processing, both when learning and when selecting actions.

Significant work remains to be done to improve upon the results of this preliminary study. A particular focus will be made on investigating techniques for iterative refinement of the GNG network, as well as the incorporation of information from a piloted phase targeted at determining appropriate action selection.

References

Baldassarri, P.; Puliti, P.; Montesanto, A.; and Tascini, G. 2003. Self-organizing maps versus growing neural gas in a robotic application. In Mira, J., and lvarez, J. R., eds., *IWANN (2)*, volume 2687 of *Lecture Notes in Computer Science*, 201–208. Springer.

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* 2(10).

Cox, P. T., and Best, S. M. 2004. Programming an autonomous robot controller by demonstration using artificial neural networks. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, 157–159.

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.

Fritzke, B. 1997. A self-organizing network that can follow non-stationary distributions. In Gerstner, W.; Germond, A.; Hasler, M.; and Nicoud, J. D., eds., *Proceedings of the Seventh International Conference on Artificial Neural Networks: ICANN-97*, volume 1327 of *Lecture Notes in Computer Science*, 613–618. Berlin; New York: Springer-Verlag.

Gunderson, L. F., and Gunderson, J. P. 2008. *Robots, Reasoning, and Reification*. Springer.

Holmstrom, J. 2002. Growing neural gas: Experiments with gng, gng with utility and supervised gng. Master's thesis, Uppsala University, Department of Information Technology Computer Systems Box 337, SE-751 05 Uppsala, Sweden.

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.

Kim, M. Y.; Cho, H.; and Kim, J. 2003. Obstacle modeling for environment recognition of mobile robots using growing neural gas network. *International Journal of Control, Automation, and Systems* 1(1).

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

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.

Shamwell, J.; Oates, T.; Bhargava, P.; Cox, M. T.; Oh, U.; Paisner, M.; and Perlis, D. 2012. The robot baby and massive metacognition: Early steps via growing neural gas. In *ICDL-EPIROB*, 1–2. IEEE.

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

Yan, W.; Weber, C.; and Wermter, S. 2012. A neural approach for robot navigation based on cognitive map learning. In *IJCNN*, 1–8. IEEE.

Yanik, P.; Manganelli, J.; Merino, J.; Threatt, A.; Brooks, J. O.; Green, K. E.; and Walker, I. D. 2012. Use of kinect depth data and growing neural gas for gesture based robot control. In *PervasiveHealth*, 283–290. IEEE.