Simulated Visual Perception-Based Control for Autonomous Mobile Agents

Daniel Flower and **Burkhard C. Wünsche** and **Hans W. Guesgen**
Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, New Zealand
dflo006@ec.auckland.ac.nz, {burkhard,hans}@cs.auckland.ac.nz

Abstract

Autonomous robots, such as automatic vacuum cleaners, toy robot dogs, and autonomous vehicles for the military, are rapidly becoming a part of everyday life. As a result the need for effective algorithms to control these agents is becoming increasingly important. Conventional path finding techniques rely on a representation of the world that can be analysed mathematically to find the best path. However, when an agent is placed into the real world in a place it has not seen before, conventional techniques frequently fail and a fundamentally different approach to path finding is required. The agent must rely on its senses, such as the input from a mounted camera, using this information to get around. We are especially interested in algorithms for use in highly interactive virtual environments such as computer games. In this paper we devise and analyse a technique which enables autonomous agents to navigate their way around a virtual city by using only what they see from their point of view. Since the scenes are computer generated we can use for the player's view and the agent's view representations with different visual complexity and hence improve the efficiency and effectiveness of the neural network. We show that by using neural networks agents can learn how to avoid obstacles, to follow the road, and we demonstrate that this method might even be useful for integration in path finding algorithm.

Introduction

The control of agents in a complex environment is a difficult problem that has been approached in a variety of ways, which can be classified into map-based navigation, map-building-based navigation and map-less navigation (DeSouza & Kak 2002). In the last class of algorithms the agent has no explicit representation of the world and must rely on visual and other sensors in order to navigate through the environment. In this case traditional navigation and path finding techniques do not work and instead object recognition, tracking and learning algorithms must be employed. This is achieved by identifying, extracting and observing relevant elements in the environment such as trees, walls and roads. Popular methods include optical flow, which makes use of motion parallax in binocular vision (Santos-Victor *et al.* 1995); appearance-based matching, which is based on "memorising" patterns in the environment and associating behaviour with it (Pomerleau 1995; Weng & Chen 1998); and object recognition which is based on identifying simple features (road outlines, trees) and associating controls with them (Turk *et al.* 1988; Kim & Nevatia 1998).

The goal of this project is to investigate the use of neural networks to analyse the visual input of an agent in order to control its navigation. The use of neural networks was inspired by the difficulty in hand coding algorithms to control obstacle avoidance in an arbitrary environment, and the success of the ALVINN system (Pomerleau 1995) which successfully controlled the steering of a vehicle on real roads by analysing the visual input. We are especially interested in algorithms for use in highly interactive virtual environments such as computer games (Take-Two Interactive Software 2005). In the past such virtual environments were usually quite limited in space and it was possible to define agent behaviour by simple scripts. However, nowadays many games allow users to create their own worlds and research on randomly generated semi-infinite cities and worlds is emerging (Parish & Müller 2001; Greuter *et al.* 2003). Consequently we need simple and fast algorithms, which allow autonomous agents to navigate through such worlds in a realistic way.

In order to test our algorithms we created a virtual city that agents can navigate around. The agents are given no information about the city other than what they see. The goals of our research were to train an agent to follow the road in much the same way as the ALVINN system, train an agent to avoid obstacles, train an agent to obey traffic lights, and explore the use of neural networks for path finding.

Our paper demonstrates for which functions a neural network can be used when controlling an agent based on its visual perception. We also introduce a novel modification of neural networks which we call "commandeering". Our results are especially useful for highly interactive virtual reality applications where agents are frequently controlled by scripts and rule-based systems (Laird & van Lent 2000; Rabin 2004) which often require a long time to develop and can lead to unrealistic, predictable and repetitive behaviour. Such behaviours are undesirable in computer games.

Design

This section gives a high-level overview of our system, starting with an overview of the virtual city, and followed by a description of our neural network and the methods we use

for training it. We also mention several design features for enhancing the performance of the neural network.

A Virtual City for Testing of Autonomous Agents

The simulation takes place in a virtual city shown in Figure 1. The city is 1200 x 1200 metres large and was designed to be simple, yet at the same time provide a variety of situations for the agent.

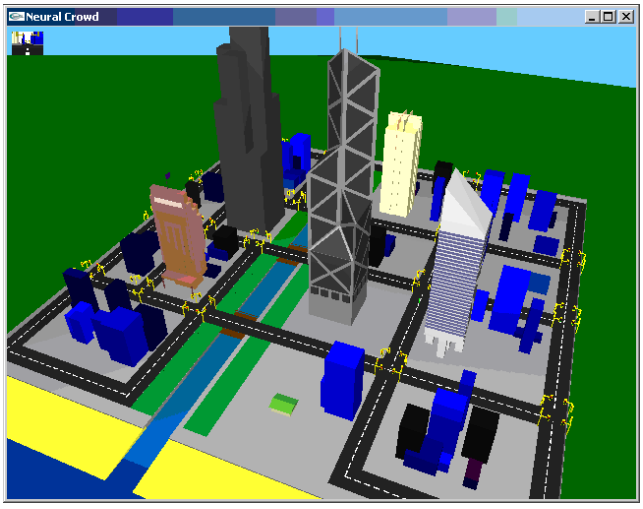

Figure 1: An overview of the city.

A major simplification in our design is that the agent moves on a plane, i.e., there are no hills, steps or slopes etc. While this constraint also makes the implementation easier, the main reason for it is that it facilitates the agent's learning process. For example, a staircase or a hill in front of the agent may appear to look the same as a wall or an obstacle, so it would take considerable longer for the agent to learn how to distinguish these scene components. We will investigate methods to overcome this limitation in future work.

In contrast to real-world object recognition systems our application uses a computer generated virtual environment. This allows us to use different representations for the player's view and the agent's view. While the player's view would usually use visually rich complex shapes (rendered using texture mapping and displacement mapping), the render pass for the agent's view can use simple objects with easily differentiable colours and patterns. Furthermore some objects which do not influence the agent's behaviour, such as small bushes, can be removed from the agent's view. The resulting simplified scene is faster to render and helps the neural network to recognize and distinguish different objects. Note that using two different representations does not represent a major overhead since most virtual environment use already multiple level-of-detail representations and bounding boxes for objects in order to increase efficiency.

In our application we only create a simplified scene as required for the agent's view. For example, the scene in figure 1 when used in a real game in the player's view might contain visually richer texture maps (including detail features such as dirt and oil stains) , billboards for small vegetation (grass, bushes), and more realistic water rendered using displacement mapping.

An important aspect of our implementation is the use of perspective and basic lighting (ambient and diffuse). The motivation for this is that perspective and illumination are important depth cues in human vision and we hope that the neural network also learns to make use of these. It is important to note that the agent only knows what it sees, i.e., it has no a priori information about the city, such as a map, position and shape of objects and distances between different objects.

Figure 1 shows several features of the city. The roads are dark-coloured and include fixed size, white coloured road markings, which help the agent identify where roads are, and to follow the roads. Surrounding the road are footpaths, which are light grey. There are also working traffic lights that agents should obey. The figure also shows the river and the grassy areas on its side along with bridges. These exist solely to give more variety to the city so that, for example, an agent will learn to follow the road regardless of whatever is surrounding the road. The existence of skyscrapers further challenges the neural network due to their size: distant skyscraper may have the same size and shape as small, close objects, so the network must learn to distinguish between these two scene components. Finally, there is the waterfront, which again exists to provide more variety in the world.

Training of the Neural Network

Training the neural network is composed of two parts: gathering training data, and applying back propagation, which is using the gathered training data to modify the weights of the neural network.

Gathering Training Data

Gathering training data is simply a matter of controlling the agent with the cursor keys. If the goal is to keep the agent in the centre of the road, then the human trainer must steer the agent around the city, ensuring that it stays in the middle of the road. Every 10 frames a screen shot is taken from the agent's point of view, which together with the steering amount and some other information creates a training instance and is saved into a text file.

The following issues are important when gathering training data for the type of system described by us.

- The screen shot taken is very small compared to the image shown on the computer screen. If the neural network simulation would utilise the full $640 \times 480 \approx 480,000$ pixels of the display window, the application would become very slow and memory intensive. However, we found that this high level of detail is not needed when controlling the agent and $32 \times 24 = 768$ pixels were sufficient.
- The agent sees the world in grey scale, i.e., the red, green, and blue pixel values in the frame buffer are converted into grey scale values between zero and one. Due to the three-dimensional nature of colour (i.e., hue, luminance and saturation), colour vision would increase the size of the neural network three-fold. This seemed unnecessary

since for tasks such as obstacle avoidance and path finding the actual colour of objects is not as important as seeing that they are there. Since most coloured objects map into different gray scale values the agent will be able to distinguish between them. Alternatively the agent's view could be immediate represented using gray scales only.

- It is sufficient to record every 10^{th} frame since from one frame to the next, neither the scene nor the amount steering will change dramatically. As a result we not only reduce the number of similar training instances but we also speed up the back propagation stage of training.
- Controlling the agent in the simulation is much like controlling a character in a first person shooter game using the keyboard. However, in most games of that type the character turns at a constant velocity until a key is released, upon which the character stops turning immediately. Using such a simplistic input makes it difficult for the neural network to learn when to turn gently (e.g. when there is an obstacle in the distance) and when to make a hard turn (e.g. when the obstacle is right in front of the agent). Therefore the agent has an angular velocity attribute which is increased and decreased depending on the trainer's input. Using an analogue input device such as a steering wheel would remove this requirement.
- Using neural networks of this type has the disadvantage that agents do not remember previous inputs even if they occurred in the last training instance. If the agent is heading straight for a wall, it may decide to turn left, but one frame later the inputs may have changed slightly such that it then decides to turn right. We found that such a zig-zag motion, like the ones novice human car drivers sometimes exhibit, was indeed a common problem in our simulation. The problem was solved by inputting ("remembering") the previous steering value into the neural network in order to "influence" its decision. Hence if in the example above the agent decided to turn left, this would would help to persuade the agent to continue turning left in the next frame. The forward velocity was also input, just in case this had an influence on the steering.
- While neural networks can tolerate some noisy data, it is beneficial to minimise the amount of such training data. For this reason we create a new file for each training instance which can be deleted if the trainer makes a mistake.

Figure 2 shows an example of what the human trainer sees (left) compared to what the neural network "sees" (right). The human trainer's window includes at the top of the screen an arrow pointing to the destination, a large red marker showing the destination in the scene (in this case on the left) and a visual representation of the current steering value at the bottom of the screen. The red square at the bottom left of the screen indicates that training is in progress, and a small window at the top left-hand corner of the screen shows what the agent is seeing. As explained previously the agent's view uses grey scales and is of a much lower resolution than the trainer's window. It is also without the auxiliary graphical components described above. However the neural network is given the numerical values of the angle and distance to the destination along with the linear and angular velocities.

Neural Network Structure and Back-propagation

There is no known algorithm for determining the best neural network topology for any given problem. Therefore, experimentation is needed to find a good network topology. The input layer of our neural network has 776 nodes: 32×24 pixels, the angle and distance to the destination, and three values each for the linear and angular velocities specifying the velocity in each axis[1]. We use just the one output layer. Having just one hidden layer with five nodes was sufficient to allow an agent to successfully avoid obstacles or follow the road.

We apply a standard back-propagation algorithm (Bishop 1995) to train the neural network. A separate program, called the "Learner", is used for performing the training of the neural network at the back-propagation stage. The program allows the trainer to steer the agent through the virtual city. Training instances are recorded and invalid instances can be deleted.

Commandeering

When providing training data for a neural network there are several aspects to consider. One of these is the amount of training data to provide, which in most cases is unknown until training has been completed and the network can be tested. Another is the need to provide a distribution of training examples which is not biased and which approximates what the neural network will see when used with new data. For example, if the training data contains significantly more left turns than right turns, then in the simulation the neural network is more inclined to turn left even when it should turn right. Without a variety of data the neural network will not learn to generalise and to distinguish between important and less important features.

For example, we found that if the training sets all use a road surrounded by gray foot paths, then the agent will perform well in unseen regions of the city as long as the roads are surrounded by such foot paths. However, when the road went through the section of the city surrounded by green grass the agent got confused, turning wildly to the left and then to the right. While the best strategy for the agent would have been to concentrate on the white lines in the middle of the road, the neural network learned to use the colour of the regions along the road as a guide. It is hence important to use a large variety of different situations as input for the training data set.

For these reasons we introduce commandeering which improves and speeds up training of the neural network. After the initial training of the neural network, the agent is allowed to run in the environment using its neural network to decide how to react. If the training data is biased, for example to always turn to the left, the human trainer can "commandeer"

[1]This allows information such as how fast the object is moving forward, sideways and vertically, along with the rotations in three directions. However, currently only the forward velocity and angular velocity around the Y axis are used, so all remaining velocities were set to 0.0 which means that the neural network ignores them.

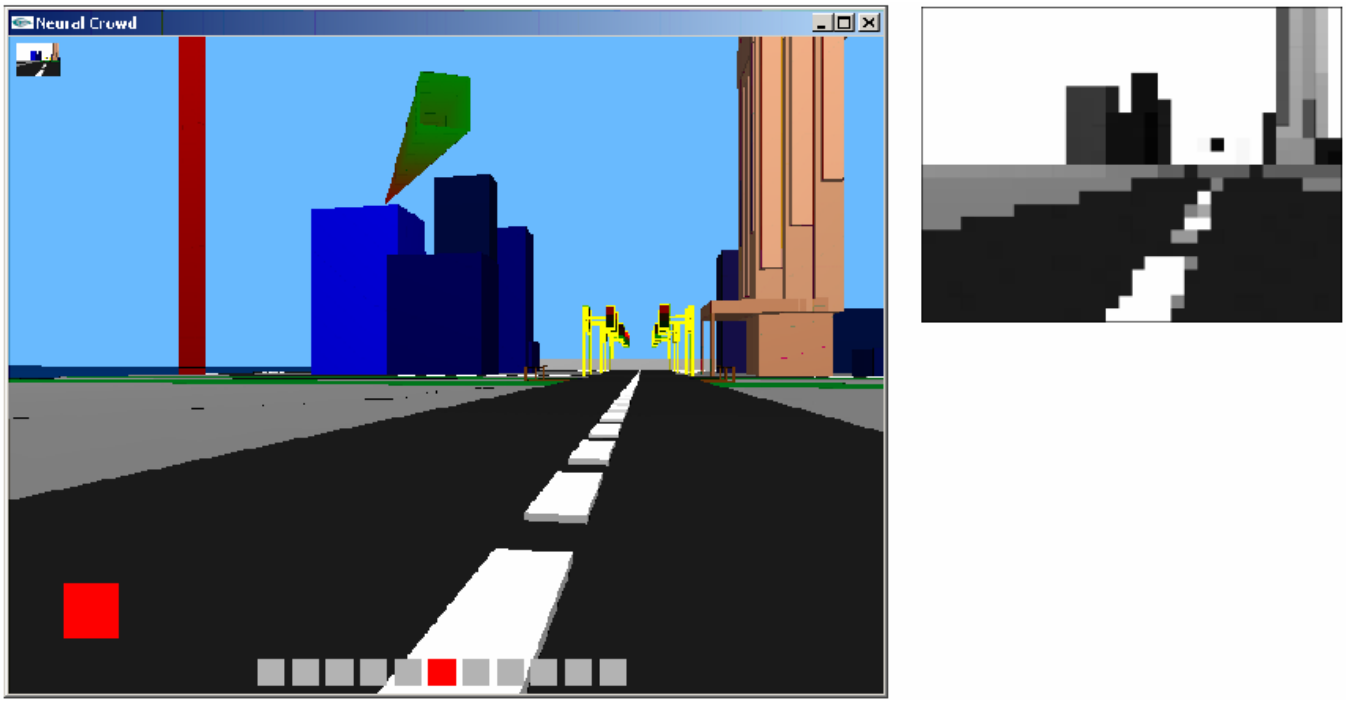

Figure 2: A screen shot from the human trainer's window (left) and the (enlarged) view the agent gets for the same scene (right).

the agent to the right, and then relinquish control to the neural network once again. During the time the trainer is controlling the agent, the agent adds this new data to its training set. In this way, the training set becomes less biased, and any unforeseen difficulties or situations that the agent encounters will become covered in the training set.

A potential drawback of this method is that a biased neural network will become biased in the opposite direction if too much commandeering is performed. The human trainers are relied upon to use their judgement to know when to stop training.

Alternative methods have been suggested to improve training data sets. For example, Pomerleau et al. (Pomerleau 1995) bound the size of the training set. If the training set is full and a new instance is added, then an old instance is removed so that the average steering direction stays neutral.

Results

As was stated in the introduction, four areas were being looked at when trying to train the neural network: obstacle avoidance, following the road, obeying the traffic lights, and path finding.

Obstacle avoidance

The first goal was to make sure the neural network could control an agent by analysing the pixel values. Therefore, a very simple world was created, which consisted of four enclosing walls with different sized obstacles distributed within them. There was neither a floor nor a ceiling, and the walls and obstacles were all the same dark colour. Figure 3 shows an overview of the world created for this test.

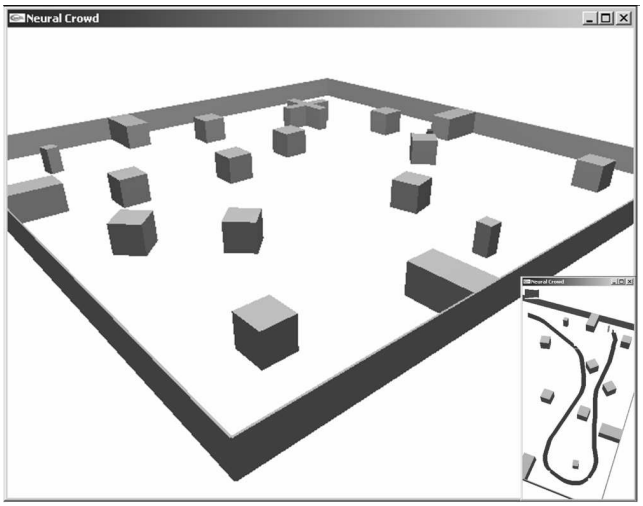

Figure 3: A simple world to test obstacle avoidance and the agent's path obtained using our simulation (see insert).

The results from this test were very promising. After only a few minutes of training the neural network was able to steer the agent around the enclosed area, successfully avoiding all obstacles. Furthermore when placed into a new environment the agent successfully avoided obstacles without any further training. The neural network learnt to ignore ob-

jects that were far away or to the sides of it. It learned that when an obstacle is close, the agent must turn left or right, depending on what is to the left or right of it. Because the neural network has no information about the objects other than what the agent sees, the results indicate that the neural network has learnt to categorise objects as being near or far.

This type of obstacle avoidance is fundamentally different from traditionally route finding algorithms using search trees, where the agent has knowledge of the position and size of objects. In those systems, the path taken by the agent is arrived upon by mathematically analysing the situation, which can give optimal solutions when the world follows certain assumptions. However, such algorithms ignore human reasoning such as if there is a gap between two obstacles, can the agent fit through it? This not only requires calculating the distance between obstacles, but also it requires reasoning about the shape and size of the agent. Hand coding a function to take care of all the possible contingencies would be extremely difficult, so training the agent by example seems ideal for obstacle avoidance in complex unstructured environments.

Road following

The next goal was to test the neural network's ability to follow the road, which was similar to the goal in (Pomerleau 1995). The agent was trained to drive down the centre of the road, so it was presumed that the neural network would identify the white lines of the road, and steer the agent so that the white lines were centred in the middle of the road. It would need to pass straight through intersections ignoring the road and its white lines perpendicular to it. It would need to turn corners, and if it approached a T-intersection and had to choose between steering left and right, then it should turn in the direction that was easier (e.g. if it was already steering slightly to the left, then it should turn left).

During training, it was important to train the agent in both directions (so that there were plenty of examples of turning both left and right), and it was also important to train the agent how to recover when it was not centred in the middle of the road by temporarily turning off the training, steering the agent away from the middle, turning training back on and then recovering. Commandeering was also needed to improve the accuracy. Figure 4 shows an example of the agent successfully navigating itself around the roads.

While the agent performed reasonably well, at times it did deviate from the centre of the road which is in general not a realistic behaviour for an agent in a virtual world (unless we want to simulate a drunken driver). In cases where the agent has a representation of the roads in its knowledge base, it may be better to use this representation to calculate the agent's position rather than using a neural network. Nevertheless, the research in (Pomerleau 1995) proves that with sufficient and high-quality training, a neural network can learn to steer safely on real-world roads.

Obeying Traffic Lights

For this step, the neural network was used to control the linear acceleration of the agent, with steering being human-controlled. It was hoped that the agent would learn to stop for red lights and go for green lights. The behaviour for orange lights should depend on the current velocity. Furthermore, traffic lights in the distance should be ignored, and the agent should stop only when it gets a certain distance away from the intersection.

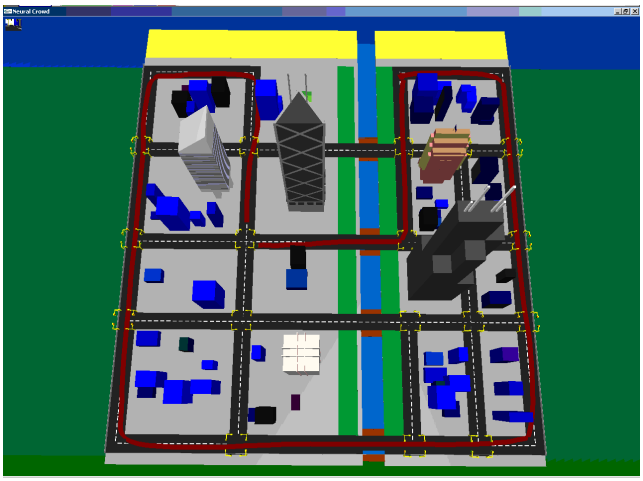

Figure 4: An example of an agent following the road. The agent was travelling in the counter clockwise direction, only turning when it needed to.

It was found that the agent was not able to learn this behaviour at all. One reason might be that the different colours of the lights were difficult to distinguish after conversion to grey scales.

Path finding

The final goal of the project was to use the neural network for path finding, which was implemented by telling the agent to move to a random destination, and when this destination was reached a new random destination was given. This task is more complex than the previous ones because the agent must head towards a destination while at the same time avoiding obstacles. To make the task easier we removed the requirement that the agent had to stay on the road and had to use bridges to cross the river.

The results were promising but far from perfect. While most of the time the agent was able to avoid obstacles and head in the general direction of the destination, its movements were sometimes erratic. It didn't take the shortest path, and it would sometimes head in the complete opposite direction to the destination.

Conclusions

We have investigated the use of neural networks to simulate vision guided navigation of autonomous agents. The agents were given no information about the layout of the world and objects within it. Instead they render the scene from their points of view and use the pixel values as input to a trained neural network which controls the movement of the agent. In order to test our application we implemented a virtual city.

In our tests we use the same representation for the player's and the agent's view.

The agents were separately trained to avoid obstacles, follow the road, obey traffic lights, and path find. Training consisted of a human instructor controlling the agents as in a video game. At each time step a log of the rendered scene and the action of the human controller was taken. The neural network was trained using the data from these log files.

The results of this project were mixed, with very good performance in obstacle avoidance and path following, but poor performance in obeying traffic lights. The path finding, which involves both obstacle avoidance and travelling towards a goal destination, showed much promise. We believe that with more training an improved performance can be obtained. Similarly a careful design of the agent's view using easily differentiable gray scale values and patterns is likely to improve object recognition.

In summary neural networks are a promising tool for controlling autonomous agents in complex, dynamic and noisy environments. A trained neural network can learn realistic behaviour based on complex scene features in a way which would be difficult to achieve by designing scripts and rule-based systems.

Future work

In order to make the simulation more realistic, especially with respect to traffic lights, the next step is to have the agents controlling the acceleration of their movement. This might require two neural networks: one for the steering and one for the acceleration, both working independently. It may be that a third neural network is required that classifies the colour of the lights, which would need to ignore the lights in the distance, and perhaps even the lights in front of it until the agent is close enough.

While it may be possible to combine this with some kind of rule-based system (e.g. "if (lights = red) then stop") another approach is to feed the result of this network into the acceleration network. This is because acceleration depends on more than just the traffic lights; it depends on whether there are corners, obstacles etc, which would make it difficult or impossible to combine with rules. It may be that more networks need to be chained together in this fashion to achieve optimal performance. Having specialised networks chained together has the advantage of allowing each part of the chain to be trained independently. For example, once the network has learnt to classify the traffic light colours correctly, there would be no danger of "untraining" it when training it to do something else. Also we hypothesise that it is easier to learn several separate simple concepts than one complicated concept.

Reynolds (Reynolds 1987) identified three rules that can be followed in order to simulate flocking behaviour seen in animals such as birds. These rules are "separation" (i.e., moving to avoid collisions with others), "alignment" (i.e., moving in approximately the same direction as the others who are close), and "cohesion" (i.e., staying close to the others). It would be interesting to see whether flocking can be implemented in the system by having the trainer follow those three rules rather than explicitly programming them. To achieve this, the set-up of the agent's point of view would have to be looked at carefully as animals that flock, such as birds, have their eyes located at the sides of their head to give a larger field of vision, compared to the relative tunnel vision that is currently used in the simulation.

There are many other possibilities that can be explored. For example, can a car learn to indicate before it starts turning? This would require the network to know when it will turn. Or, can one car learn to follow another? This problem may further expose the drawback that neural networks do not remember previous actions.

Possibly the most important future work is to determine which class of AI algorithms is most suitable for which types of problems encountered in highly-interactive virtual worlds such as computer games.

References

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition*. Oxford, England: Oxford University Press.

DeSouza, G. N., and Kak, A. C. 2002. Vision for mobile robot navigation: A survey. *IEEE Transactions on Pattern Analysis Machine Intelligence* 24(2):237–267.

Greuter, S.; Parker, J.; Stewart, N.; and Leach, G. 2003. Real-time procedural generation of 'pseudo infinite' cities. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, 87–94. New York, NY, USA: ACM Press.

Kim, D., and Nevatia, R. 1998. Recognition and localization of generic objects for indoor navigation using functionality. *Image and Vision Computing* 16(11):729–743.

Laird, J. E., and van Lent, M. 2000. Human-level AI's killer application: Interactive computer games. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, 1171–1178. AAAI Press / The MIT Press.

Parish, Y. I. H., and Müller, P. 2001. Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 301–308. New York, NY, USA: ACM Press.

Pomerleau, D. 1995. Neural network vision for robot driving. In Arbib, M., ed., *The Handbook of Brain Theory and Neural Networks*. Cambridge, Massachusetts: MIT Press.

Rabin, S., ed. 2004. *AI Game Programming Wisdom 2*. Charles River Media. chapter 9 - Scripting.

Reynolds, C. W. 1987. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics* 21(4):25–34. URL: `http://www.red3d.com/cwr/papers/1987/boids.html`.

Santos-Victor, J.; Sandini, G.; Curotto, F.; and Garibaldi, S. 1995. Divergent stereo in autonomous navigation: from bees to robots. *International Journal of Computer Vision* 14(2):159–177.

Take-Two Interactive Software. 2005. Civilization III home page. URL: `http://www.civ3.com/`.

Turk, M. A.; Morgenthaler, D. G.; Gremban, K. D.; and Marra, M. 1988. VITS - a vision system for autonomous land vehicle navigation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10(3):342–361.

Weng, J., and Chen, S. 1998. Vision-guided navigation using SHOSLIF. *Neural Networks* 11(7-8):1511–1529.