A Connectionism Approach to the Emulation of Multi-Agent Mobile Robots

S. Massey and A.M.S. Zalzala

Robotics Research Group,
Department of Automatic Control and Systems Engineering,
University of Sheffield
Email: rrg@sheffield.ac.uk

Research Report # 538
2 October 1994

Abstract

The work presented in this report deals with the modelling and subsequent neural network emulation of autonomous mobile robots, moving in pre-defined environments in response to given control signals. This work was undertaken with the intention of training neural network based controllers for the vehicles, in order to control them whilst they performed required navigational tasks, whilst avoiding collisions with each other and with environmental obstacles. The required modelling was carried out using a combination of trigonometry and geometry, the corresponding neural network emulators were trained using an algorithm based on back propagation, and the resulting emulator networks performed well enough for work to now begin on controller training.

A Connectionism Approach to the Emulation of Multi-Agent Mobile Robots

S. Massey and A.M.S. Zalzala
Robotics Research Group,
Department of Automatic Control and Systems Engineering,
University of Sheffield
Email: rrg@sheffield.ac.uk

Research Report # 538
2 October 1994

Abstract

The work presented in this report deals with the modelling and subsequent neural network emulation of autonomous mobile robots, moving in pre-defined environments in response to given control signals. This work was undertaken with the intention of training neural network based controllers for the vehicles, in order to control them whilst they performed required navigational tasks, whilst avoiding collisions with each other and with environmental obstacles. The required modelling was carried out using a combination of trigonometry and geometry, the corresponding neural network emulators were trained using an algorithm based on back propagation, and the resulting emulator networks performed well enough for work to now begin on controller training.

1. Introduction

Multi-agent robotics deals with the development of co-operative robots. Such robots can be used to replace humans in hazardous, unpleasant or inaccessible environments, or in situations where robot use is more cost effective or convenient. One approach to this development attempts to emulate human co-operation by combining robot autonomy with the use of neural networks for individual robot learning. The work presented in this report considers co-operative mobile robots navigating within pre-defined environments in order to accomplish some navigational task without colliding with each other or any environmental obstacle. Such work is applicable in any situation where mobile robots equipped with sensory devices could work in co-operation, for example, in a warehouse transportation system, or a mobile cleaning fleet.

The work set out in this report is based on that presented by Biewald,[1] in which he trained neural network controllers for the navigation of a single vehicle, using an extended version of Nguyen & Widrow's motion control architecture.[2,3] In order to achieve this, he first used back propagation to train networks to emulate both the vehicle's sensor readings for various environments and the vehicle dynamics, and then, using both the weights of the trained networks and the actual vehicle sensor readings and dynamics, trained the controller networks, by applying back propagation through time.[4]

The behaviour of simple co-operative mobile robots, or vehicles, was modelled, and neural networks were trained to emulate both their sensor readings, for a selection of environments, and their dynamics, making possible the subsequent training of neural network navigational controllers which can aim to navigate the vehicles to accomplish a set task whilst avoiding collisions.

The vehicles considered were equipped with sensors to allow them to detect obstacles within their environment, and moved in response to a control signal. Both vehicle dynamics and vehicle sensor readings for a selection of environments were modelled, environments being modelled both with and without obstructing vehicles, and the models developed were tested against expected results. Vehicle dynamics were considered in terms of forward and sideways movement and vehicle orientation change in response to a control signal containing stop/go and left/right commands and a steering control angle.

A neural network training algorithm, based on back propagation, was developed to train a network to emulate sensor readings for the simplest vehicle environment. This algorithm was subsequently improved by the introduction of convergence checking and the restriction of training pattern range, the introduction of a specific training set and training pattern cycling, the adoption of batch training, layer specific learning rates

and adjustable learning rates, the reversal of 'bad' training steps, the addition of momentum, and the addition of noise to the network weights if a local minimum was suspected. Once developed fully, the training algorithm was adapted to train networks both to emulate sensor readings for all other environments under consideration and to emulate vehicle dynamics.

Results will be given, showing both the continued improvement in the basic algorithm during its development, and the performance of the emulator networks trained.

The models and algorithms that were developed were implemented in Borland C, running under Microsoft Windows on a 486 personal computer.

2. Vehicle Modelling

2.1. Sensor Modelling

Each mobile vehicle was equipped with 'ultrasonic' sensors, the arrangement of which is illustrated below in Figure 2.1, and the sensors' readings were calculated from the vehicle's position with respect to environmental obstacles, its orientation and its dimensions.

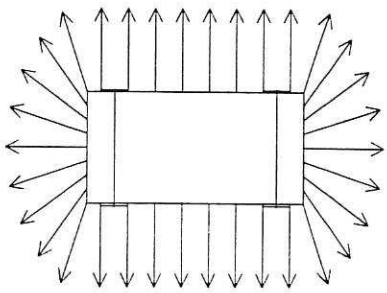

Figure 2.1 *Sensor Arrangement*

In order to make the continuation of controller training possible in the event of a collision, vehicles were permitted to overlap with walls and with each other, making it necessary to give meaning to their sensor readings under such circumstances. Several separate cases had to be considered, and the various possibilities are covered below in Table 2.1. In all cases sensor readings were defined by considering the information that would be of most use in controller training and were limited to lie within a pre-defined range.

2.1.1 Modelling for Single Vehicles

Modelling was carried out for a single vehicle, both in an environment containing a straight bounding wall and an environment bounded by an L-shaped wall, and was achieved using a straight-forward trigonometric approach, an example of which is given below.

The arrangement for calculating sensor readings for the vehicle's right sensors in a straight wall environment is shown in Figure 2.2.

Sensor Position	Size of Sensor Reading	Sign of Sensor Reading
Outside all obstacles	Distance to closest obstacle, measured forwards along line of sensor beam	Positive
In one wall, pointing into wall	Distance to wall, measured backwards along line of sensor beam	Negative
In one wall, not pointing into wall	Distance to closest obstacle, not including wall that contains sensor, distance measured forwards along line of sensor beam	Positive
In both walls, pointing into both walls	Distance to furthest wall, measured backwards along line of sensor beam	Negative
In both walls, pointing into one wall	Distance to wall that sensor points into, measured backwards along line of sensor beam	Negative.
In both walls, not pointing into either	Distance to closest obstacle, not including walls, distance measured forwards along line of sensor beam	Positive
In obstructing vehicle, but not in wall and pointing into that wall	Distance to side of obstructing vehicle, measured backwards along line of sensor beam	Negative

Table 2.1 *Dependence of Sensor Readings on Sensor Position In Relation to Environmental Obstacles*

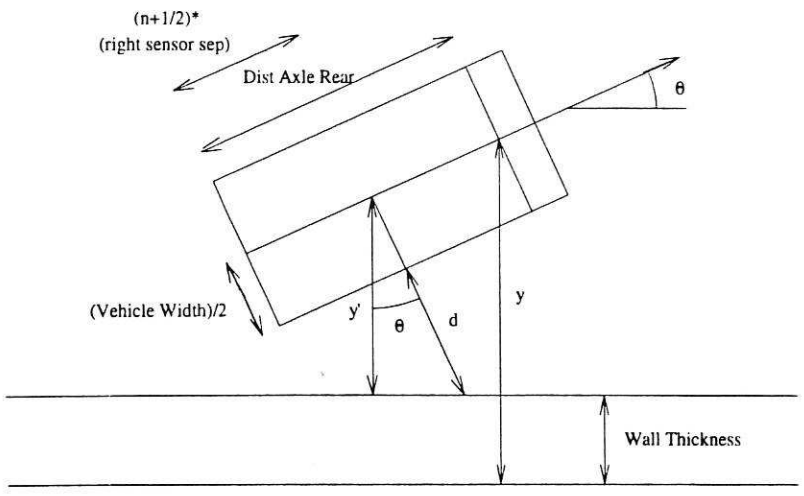

Figure 2.2 *Arrangement for Calculating Right Sensor Readings for a Single Vehicle in a Straight Wall Environment*

From Figure 2.2 it can be seen that:

$$y' = y - Wall\_Thickness - (Dist\_Axle\_Rear - (n+1/2)*right\_sensor\_sep)*\sin\theta \quad (2.1)$$

$$y' = (d + Vehicle\_Width/2)*\cos\theta \quad (2.2)$$

where y' is a temporary variable, y is the y co-ordinate of the centre of the front axle, θ is the orientation of the vehicle, n is the number of the sensor under consideration (sensors being numbered from the rear of the vehicle, starting from zero), and d is the distance that is interpreted as the sensor reading.

Hence, from equations 2.1 and 2.2:

$$d = (y - Wall\_Thickness)/\cos\theta - (Dist\_Axle\_Rear - (n+1/2)*right\_sensor\_sep)*\tan\theta - Vehicle\_Width/2 \quad (2.3)$$

2.1.2 Modelling for Multiple Vehicles

When considering a vehicle obstructed by one or more other vehicles in either the straight or the L-shaped wall environment, a purely trigonometric approach was judged to be too cumbersome and inflexible, and an approach involving a combination of trigonometry and geometry was adopted. The modelling algorithm used is summarised below, with items applicable in the presence of a vertical wall enclosed in parentheses. All distances from sensors to detected obstacles were found using straight line intersection and Pythagorus's Theorem, whilst trigonometry was used to find the positions of both sensors and vehicle corners.

- For each of the main vehicle sensors in turn:
  - Set sensor reading to sensor range.
  - If sensor would 'see' horizontal wall if there was no obstacle of higher priority, set sensor reading, with appropriate sign, as though horizontal wall is seen.
  - (If sensor would 'see' vertical wall if there was no obstacle of higher priority. Reset sensor reading accordingly if necessary by doing the following:)
    - (If sensor is in vertical wall and negative distance is less than sensor reading, reset sensor reading to minus distance from sensor to vertical wall along line of sensor beam.)
    - (If sensor is not in either wall and distance is less than sensor reading, reset reading to distance from sensor to vertical wall along line of sensor beam.)
  - (If sensor is in both walls, pointing away from both and parallel to either, then reset sensor reading to minus sensor range.)
  - If sensor is not in wall and pointing into that wall, sensor may 'see' one of the obstructing vehicles. Reset sensor reading accordingly if necessary by doing the following for each side of each obstructing vehicle:
    - If line of sensor beam intersects with side, find distance from sensor to side along sensor beam.
    - If sensor is not in vehicle under consideration, sensor beam points towards side, and distance is less than sensor reading, reset sensor reading to distance from sensor to side along line of sensor beam.
    - If sensor is in vehicle under consideration and sensor points away from side, then reset sensor reading to minus the distance from sensor to side along line of sensor beam.

The above algorithm has the advantage that it can be applied for any number of obstructing vehicles, assuming that sensors do not become embedded in more than one of the vehicles at any time. It should also be possible both to remove this restriction and to alter the arrangement of walls within the environment with relatively little effort.

2.2. Dynamics Modelling

The vehicle's dynamics were simplified for modelling purposes, by considering the front wheels and axle alone. The control signal was a vector consisting of three components, the first being a stop/go command, the second a left/right command and the third a steering control angle. The movement vector also consisted of

three components, the first being the distance moved forwards in the direction of the vehicle's orientation at the moment when the control was applied, the second the distance moved perpendicular to this and the third the change in the vehicle's orientation. The stop/go command was included so that it would be possible for vehicles to avoid each other by stopping if necessary, the steering control angle dictated the angle between the outside front wheel and the vehicle body, and both the distance moved per control interval in response to a 'go' command, and the scale factor by which the angle between the inside front wheel and the vehicle body differed from the steering control angle (referred to as *SF* below) were pre-defined.

Consider a command to turn right. The arrangements for calculating the movement vector from the control vector are shown below in Figures 2.3, 2.4 and 2.5.

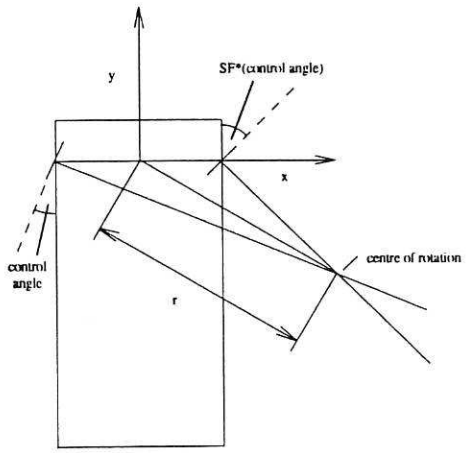

Figure 2.3 Arrangement for Calculating the Co-ordinates of the Centre of Rotation and its Distance from the Centre of the Front Axle

From Figure 2.3 it can be seen that if the centre of rotation has co-ordinates (x_{COR}, y_{COR}),

$$x_{COR} = Vehicle\_Width\ /\ 2*(\tan(SF*control\_angle)+ \tan(control\_angle))\ / (\tan(SF*control\_angle)-\tan(control\_angle)) \quad (2.4)$$

$$y_{COR} = -\tan(control\_angle)*Vehicle\_Width* \tan(SF*control\_angle)\ / (\tan(SF*control\_angle)-\tan(control\_angle)) \quad (2.5)$$

From Figure 2.3 it can also be seen that:

$$r^2 = {x_{COR}}^2 + {y_{COR}}^2 \quad (2.6)$$

where r is the distance from the centre of the front axle to the centre of rotation.

From Figure 2.4(a) it can be seen that

$$D = r*\sqrt{2*(1-\cos(Dist\_Travelled\_Per\_Control\_Int\ /\ r))} \quad (2.7)$$

where D is a temporary variable.

From Figures 2.3, 2.4(a) and 2.4(b) and equation 2.7 it can be seen that:

$$dist\_moved\_forwards = r*\sqrt{2*(1-\cos(Dist\_Travelled\_Per\_Control\_Int\ /\ r))}* \cos(Dist\_Travelled\_Per\_Control\_Int\ /\ (2*r)+ \tan^{-1}(-y_{COR}\ /\ x_{COR})) \quad (2.8)$$

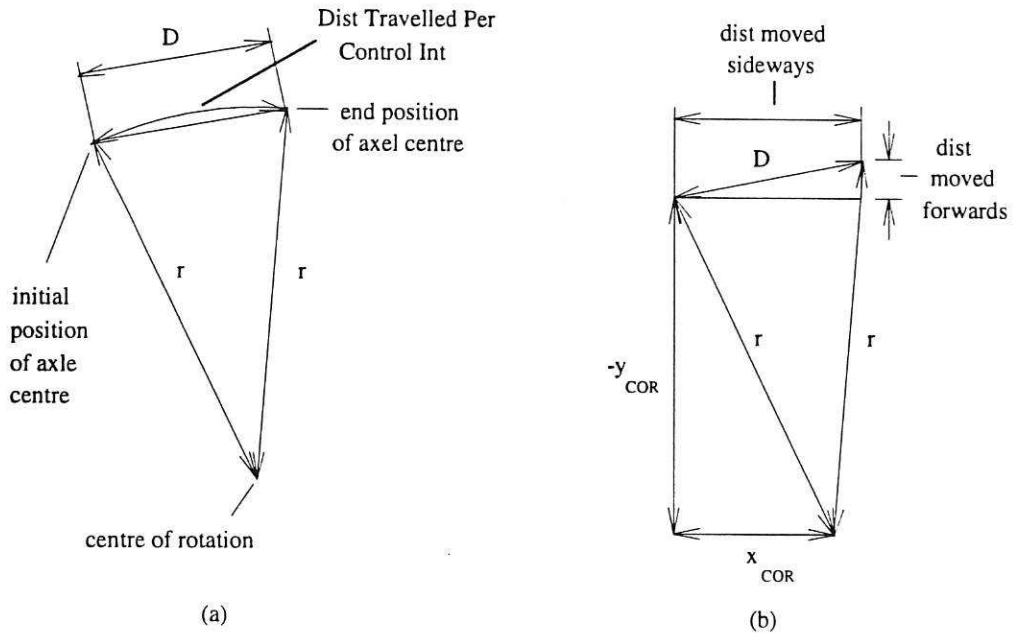

Figure 2.4 *Arrangements for Calculating Required Distances Moved in Response to Control Signal*

$$dist\_moved\_sideways = r*\sqrt{2*(1-\cos(Dist\_Travelled\_Per\_Control\_Int/r))}* \sin(Dist\_Travelled\_Per\_Control\_Int/(2*r)+ \tan^{-1}(-y_{COR}/x_{COR})) \quad (2.9)$$

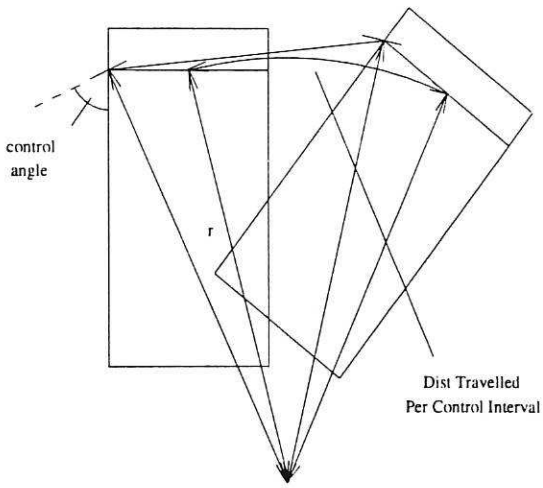

Figure 2.5 *Arrangement for Calculating Orientation Change in Response to Control Signal*

From Figures 2.3, 2.4(a) and 2.5 it can be seen that:

$$change\_in\_orientation = \tan^{-1}((x_{NOWC} - dist\_travelled\_sideways) / (dist\_travelled\_forwards - y_{NOWC})) - \pi/2 \quad (2.10)$$

If a command is given to turn left instead of right, the resulting movement can be obtained in the same way, but the sign of *change_in_orientation* must be changed. It should be noted that no movement will occur if the first element of the control vector is 'stop'.

3. Emulation Network Training Using Back Propagation

Emulation network training took place for all the models developed. The approach taken to achieve this involved the development and refinement of a basic back propagation algorithm to train an emulation network for the simplest sensor model, and the subsequent refinement of that algorithm and its adaptation to train networks for the remaining models.

3.1 Basic Algorithm

Network training was based on the minimisation of the cost function

$$E = \frac{1}{2} * \left[\sum_{k \in output\_layer} (e_k)^2 \right] \quad (3.1)$$

where

$$e_k = d_k - y_k \quad (3.2)$$

for a network with one hidden layer, using the output functions

$$y_k = \tanh(net_k) \qquad k \in hidden\_layer \quad (3.3)$$

$$y_k = net_k \qquad k \in output\_layer \quad (3.4)$$

where E is the cost function and d_k, y_k and net_k are the desired output, actual output and weighted sum of the inputs of neuron k respectively, d_k being obtained from the relevant model.

The network trained had three inputs, corresponding to the x and y co-ordinates of the centre of the vehicle's front axle, and its orientation, thirty-four outputs, each corresponding to a vehicle sensor, forty hidden neurons in a single layer and both a hidden layer and an output layer bias. To make the network easier to adapt for other applications, the inputs and desired outputs were all scaled to lie between -1 and +1, and the number of network inputs, outputs and hidden neurons were all pre-defined.

The standard back propagation algorithm which was initially used to train the network is given below in equation 3.5.

$$w_{ij}(t+1) = w_{ij}(t) + learning\_rate(t) * output\_equivalence\_error_i(t) * y_j(t) \quad (3.5)$$

where $w_{ij}(t)$ is the weight of the connection between neuron i and neuron j at time t and $output\_equivalence\_error_j$ is given below by equation 3.6 for neurons in the output layer and by equation 3.7 for neurons in the hidden layer.

$$output\_equivalence\_error_i = -e_i \quad (3.6)$$

$$output\_equivalence\_error_i = -(1 - y_i^2) * \sum_{k \in output\_layer} (output\_equivalence\_error_k * w_{ki}) \quad (3.7)$$

Random initial weights were generated for the training algorithm from ranges set to avoid initial saturation of the hidden neurons and keep the cost function reasonably small, and a constant learning rate was set to make initial weight adjustments small compared to these ranges. Training patterns were generated randomly from the range of vehicle positions considered to be of interest, and training occurred for each pattern as it was generated. The resulting algorithm, en1v1.cpp, was coded to run for a pre-defined number of training cycles and its performance was assessed by looking at the network root-mean-squared (r.m.s.) output error after training was completed.

3.2 Modification of Basic Algorithm

The performance of en1v1.cpp was unsatisfactory, and a series of modifications were made, as detailed below in Table 3.1.

Algorithm Name	Modification
en1v2.cpp	Algorithm coded to run until an accuracy goal was achieved and training pattern range restricted.
en1v3.cpp	Introduction of training set with number of training patterns set to approximately ten times the number of hidden neurons. Training set cycled through repeatedly throughout training.
en1v4.cpp	Introduction of batch training, with entire training set contributing to calculation of weight adjustment for each training cycle, initial weight range and learning rate being adjusted accordingly.
en1v5.cpp	Introduction of layer specific learning rates.
en1v6.cpp	Introduction of adjustable learning rates.
en1v7.cpp	Reversal of 'bad' training steps.
en1v8.cpp	Introduction of 'momentum'.
en1v10.cpp	Addition of noise to network weights if cost function local minima suspected.

Table 3.1 Modifications to Basic Training Algorithm

3.3 Modification of Algorithm to Train Networks for Remaining Models

The basic algorithm was modified to train networks to emulate vehicle sensor readings for a single vehicle in an L-shaped wall environment, and for two vehicles in both a straight wall and an L-shaped wall environment, and to emulate vehicle dynamics, giving algorithms en2v1.cpp, en3v1.cpp, en4v1.cpp and dyn1v1.cpp respectively. Modification was achieved by altering the number of network inputs and outputs appropriately, and in the case of the dynamics emulator network, halving the number of hidden neurons, and by altering the generation of training and test patterns, training pattern cycling, input and output scaling and desired network outputs accordingly.

4. Results

4.1 Comparison of Training Performance of Algorithms Applied to Sensor Emulation Training Problem for a Single Vehicle in a Straight Wall Environment

Algorithms en1v4.cpp, en1v5.cpp, en1v6.cpp, en1v7.cpp, en1v8.cpp and en1v10.cpp were run for 10,000 training cycles and their r.m.s. output errors were recorded every 1,000 cycles as a percentage of the desired output modulus range (en1v1.cpp, en1v2.cpp and en1v3.cpp being omitted due to poor performance, and en1v9.cpp being omitted since it contained no new training code). The results are shown below in Figure 4.1.

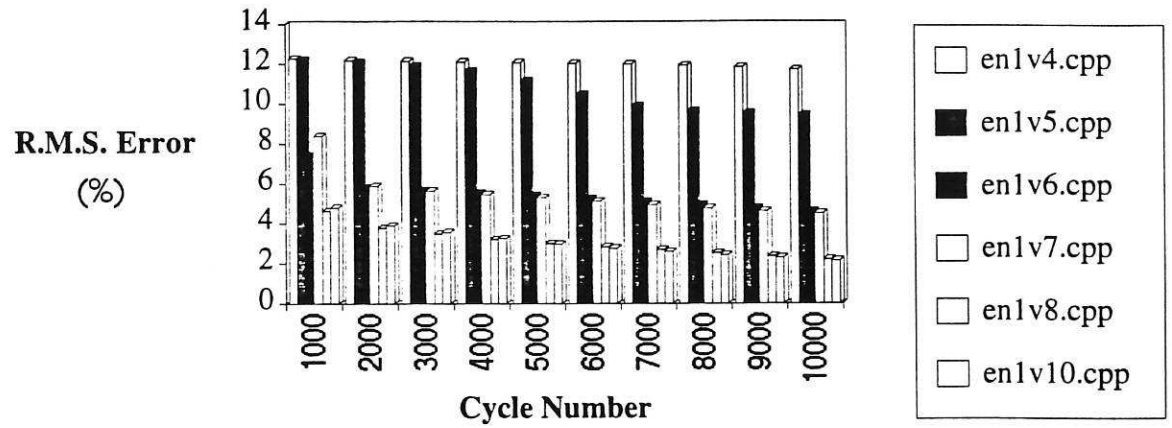

Figure 4.1 Comparison of Algorithm Training Performance

It should be noted that each training run started with different random weights, but never the less, it would appear that training improved as the training algorithm was modified.

4.2 Performance of Trained Emulator Networks

The trained emulator networks were each run for ten randomly generated input patterns taken from the same range as their training patterns. The r.m.s. error for each scaled output was found as a percentage of the desired scaled output modulus range, and the results are summarised below in Table 4.1. Note that training continued for several days, and was interrupted when I felt that no further significant improvement was likely.

Algorithm	No. Training Cycles	Training Pattern R.M.S. Output Error	Test Pattern R.M.S. Scaled Output Error
en1v10.cpp	22,661	0.95%	0.94%
en2v1.cpp	65,366	7.53%	9.68%
en3v1.cpp	63,242	11.37%	14.17%
en4v1.cpp	54,801	12.72%	12.23%
dyn1v1.cpp	118,088	0.33%	0.18%

Table 4.1 Summary of Results for Final Training Algorithms

From the results shown in Table 4.1 it can be seen that the trained emulator networks do give a fair, and in the more simple situations a very good, approximation to the models developed, although there does appear to be a significant upward trend in the r.m.s. output errors of the sensor emulator networks as environmental complexity increases. This is only to be expected, but it may give rise to emulator network accuracy problems if more complicated environments need to be considered, making it necessary to investigate the effect on accuracy of an increase in the number of hidden neurons in the emulator networks. However, comparing the results obtained with those of Biewald, it can be seen that the performance obtained from the training algorithms and emulator networks should be sufficiently adequate to permit controller training to take place for all the environments modelled.

5. Conclusions

Models have been successfully developed for simple co-operative mobile robots, or vehicles, covering both their sensor readings in a selection of environments, and their dynamics. A basic neural network training algorithm has also been developed, refined and adapted to give a set of algorithms that were successfully used to train neural networks to emulate both sensor reading models and the vehicle dynamics model, producing results which can now be used in the training of neural network navigational controllers.

Two main environments were covered, and sensor readings were modelled for each, both in the case where a vehicle was alone, and in the case where other obstructing vehicles were present. Vehicle dynamics modelling was based on the motion of the vehicle's front axle.

Sensor emulator networks were trained for each of the main environments modelled, both in the presence and in the absence of one obstructing vehicle, giving r.m.s. output errors between 0.94% and 14.17% for randomly generated test sets, with the errors appearing to increase with environmental complexity, as is to be expected. The dynamics emulator network produced an r.m.s. output error of 0.33% for its training set, with this error actually reducing to 0.18% for its test set, although this reduction can probably be put down to a fortuitous selection of test patterns.

Training of additional sensor emulator networks for the main environments in the presence of more than one obstructing vehicle could easily be accomplished using existing models by making minor adaptations to the network training algorithm, and it would also be possible to accommodate additional basic environments with relatively little effort.

The way is now open for the training of neural network navigational controllers for the modelled vehicles.

References

1. R. Biewald, 1993, "A Neural Controller for Navigation for Non-Holonomic Mobile Robots Using Sensory Information", in G. Orchard (editor) "Neural Computing: Research and Applications", Institute of Physics Pub., 235-142.
2. D.H. Nguyen, and B. Widrow, 1990, "Neural Networks for Self-Learning Control Systems", *IEEE Control System Magazine*, 10(3):18-23 (or 1991, *Int. Jour. of Control*, 54.2(6):1439-1451).
3. D.H. Nguyen, and B. Widrow, 1989, "The Truck Backer-Upper: An Example of Self-Learning in Neural Networks", *Proc. of IJCNN-89*, 2:357-363.
4. P.J. Werbos, 1990, "Backpropagation Through Time: What It Does and How to Do It", *Proceedings of the IEEE*, 78(10):1550-1560.