Learning of Visual Navigation Strategies

Verena V. Hafner[*], **Ralf Möller**[**]

[*]*Artificial Intelligence Lab, Dept. of Information Technology, University of Zurich, vhafner@ifi.unizh.ch*

[**]*Cognitive Robotics, Max Planck Institute for Psychological Research, Munich, moeller@mpipf-muenchen.mpg.de*

Abstract. This paper presents a mechanism of learning a homing strategy, which is mostly independent from the environment. It is comparable with theoretic homing models in insects like the snapshot model (Cartwright and Collett, 1983) or the Average Landmark Vector model (Lambrinos et al., 2000). In contrast to those models, the navigation strategies presented in this paper are not pre-defined but learned in interaction with the environment. The preconditions for the ability to learn are kept as minimal as possible. The learned homing strategies have been tested in both simulation and on a mobile robot.

1 Introduction

Many animals use visual cues for navigation. This paper relates to two visual navigation strategies that have been suggested for insects: the snapshot model, and the Average Landmark Vector (ALV) model. According to the snapshot model, the insect stores a panoramic view at the target location, which is compared with a view taken at the current location, aligned with an external compass direction, to derive a homing vector (Cartwright and Collett, 1983). The insect can keep track of this compass direction by using for example the polarisation pattern of the blue sky (Lambrinos et al., 1997). A more parsimonious model is called the Average Landmark Vector (ALV) model, which can be derived from the snapshot model (Lambrinos et al., 2000). This model has the advantage that only a single vector (average of the vectors pointing towards landmarks) has to be stored instead of a view, and no feature matching is required to determine the home vector.

Interestingly, in none of these navigation strategies does the animal know about its current location, but is still able to find its target following a constantly updated homing vector. An answer to the question "Where am I?" is therefore not necessarily a prerequisite for the decision "Which direction do I have to take to get home?" (see the discussion in (Franz and Mallot, 2000)).

The given navigation strategy is assumed to be genetically fixed in both the snapshot and the ALV model. In this paper, we examine whether such a navigation strategy — which does not depend on the specific structure of the environment — can be *learned* in interaction with the environment.

2 Neural Network structures and Learning Methods

2.1 Training Data

The information available to the agent was restricted to a one-dimensional 360° view taken at the home position, and one taken at the current position. Both views have to be aligned to the same external coordinate system by using a compass. Given a pair of those views, it shall be possible to guess the orientation of the vector pointing from one to the other position where the views have been taken.

For the learning process, it is assumed that the agent can keep track of a home vector by path integration. In this manner, a *self-supervised* learning process could work by having the following information: Two orientation-invariant views as the input and the orientation of the vector connecting their locations as the output.

The first tests in simulation are performed with a training set of 2269 snapshots (2500 minus the snapshot positions lying within landmarks) taken at the grid positions of a 50×50 grid shown in figure 1. The simulated arena contains several black cylinders and is surrounded by a white wall.

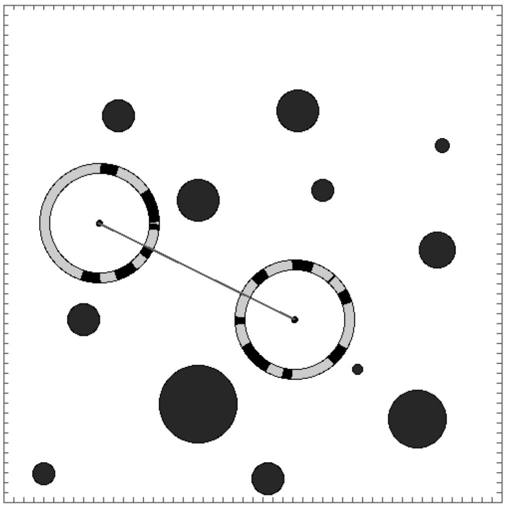

Figure 1: The world used for generating the training data consists of a plane with twelve cylinders (shown from above) of random size and distribution. The training views are taken at the crossing points of a 50×50 grid (indicated by tick marks on the frame). The circles around the end points of the line show an example of taking two snapshots in this environment. Note that the snapshots are filtered to one black pixel per landmark region before they are fed into the neural network as shown in figure 2.

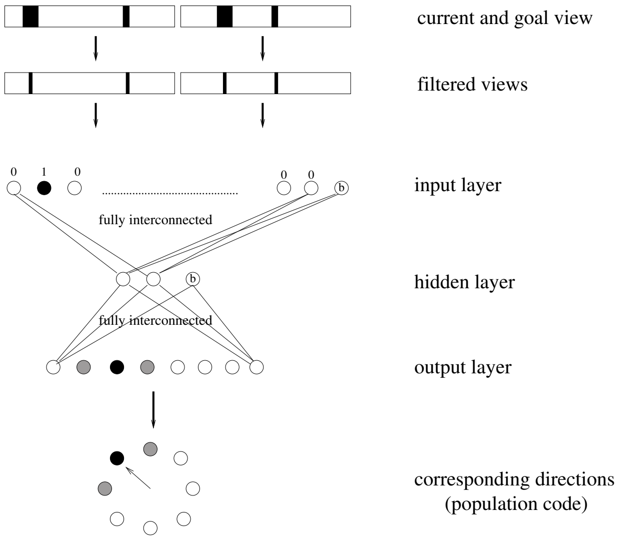

Figure 2: The neural network structure is that of a multi-layer perceptron with two layers, the learning algorithm is backpropagation with momentum term. The input layer consists of 180 neurons, 90 for each view, plus one bias neuron b. The binary views have been filtered in order to keep only one neuron active per landmark region. The number of hidden layer neurons varied from two to four plus one bias neuron b, the number of output neurons was eight, which represents eight different direction in $45°$ steps (population coding). The different grey values for neurons in the figure stand for their activation.

2.2 MLP with Backpropagation

As first strategy we choose a Multi-Layer Perceptron (MLP) with backpropagation (Rumelhart et al., 1986). The preprocessed input is fed directly into the input layer of the neural network which consists of 180 neurons, 90 for each view. We started with four hidden layer neurons, since we wanted to allow the network to keep the information of the two input views still separated in the hidden layer. Interestingly, the number of hidden layer neurons could be reduced to two without having a dramatic effect on the overall results (see section 3.3 for an explanation). If not otherwise indicated, 180-2-8 MLPs are used. For the encoding of the home vector in the output layer we used eight output neurons, which represented eight different directions in $45°$ steps. The direction was then determined from the centre of gravity of the output layer activity (see figure 2). This strategy is called population coding and is assumed to be more robust under noisy conditions, and also more biologically plausible than direct encoding (Georgopoulos et al., 1986).

2.3 Perceptron with Delta Rule

Since the number of hidden layer neurons in the MLP could be reduced to two, it might also be possible to just encode the x and y components of the normalised homing vector in a simple perceptron network with two output neurons (see figure 3). For this test, two neurons each with 180 inputs coming from the views, a $\tanh$ activation function, and real valued error correction (delta learning rule) have been chosen. Using local learning rules, this approach is more likely to be used in a biological system than backpropagation.

We chose those two learning structures and rules to show that visual homing can be learned in principle. We do not discuss the actual neural network structures in the ants' brain, of which is very little known to date.

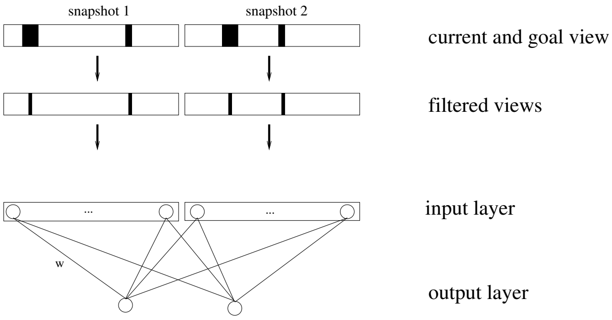

Figure 3: Network structure for the perceptron system. For the perceptron learning, a real valued error correction (delta learning rule) and a $tanh$ sigmoid function have been used.

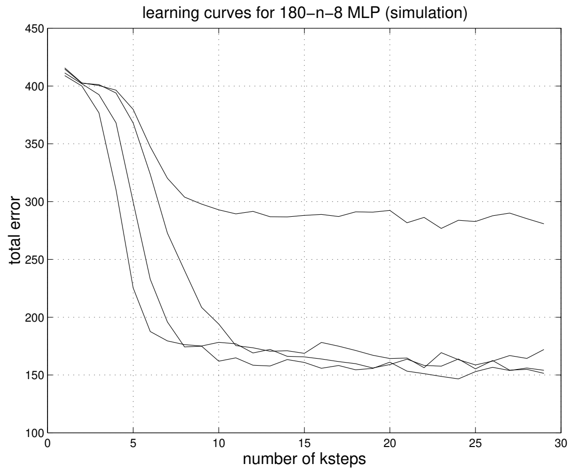

Figure 4: Total error for different number n of hidden layer neurons. $n = 1, 2, 4, 16$ from top to bottom.

3 Simulation Results

3.1 Generation of Training Data

In order to train the neural network with backpropagation, two randomly chosen views from the environment in figure 1 were fed to the neural network, the orientation of the vector connecting their positions was set as desired output (even if there happened to be obstacles between the two positions). Preprocessing of the views is performed by keeping only one neuron active per landmark region. The orientation of the views was adjusted to the orientation of the current view. For the training, the orientations of both views are adjusted to be the same random orientation. The total number of training pairs is therefore $(2269^2 - 2269) * 90 = 463148280$. After 100 runs of feeding 1000 randomly selected view pairs (online learning, views are different in each run), the training process has been stopped. Not all of the many training examples had to be used until the error converged.

3.2 MLP Performance

The error curves (sum of squared error in the output layer) of MLPs for different numbers h of hidden layer neurons are shown in figure 4. Starting with $h = 2$, the error does not decrease significantly with increasing h.

An agent with a neural network (180-2-8 MLP), trained with the training samples mentioned above, has been tested in the simulated environment. In figures 5 and 6, trajectories of this agent are shown. The agent could move continuously in all directions. If the home position was reached, the area around the starting point was marked in grey.

In order to explore the generalisation capability of the neural network, the agent which has been trained in the environment with the twelve cylinders (figure 5) was tested in a different environment with polyhedral landmarks (figure 6). The homing performance was still very good. It seems that the trained neural network is mainly independent of the environment, like the snapshot and the ALV model (discussed before). The filter mechanism already causes different environments to be more similar for the agent than without the filtering.

It is also interesting to note that the trajectories do not go straight to the target position as taught, but seem to avoid obstacles. This is an emergent property resulting from the learned weights. As can be seen later (in section 3.7), the components of the home vector pointing straight towards the goal cancel each other out, if a landmark is in between those two positions.

3.3 MLP Weights

To understand the navigation mechanism learned by the agent, we have to examine the weights. In figure 7, the weights between the input and the hidden layer (two hidden neurons) are arranged in the following way: The weights coming from neuron 0 to 89 are plotted normally, the weights coming from neuron 90 to 179 were shifted by 90 to the left and multiplied by -1. Both pairs of curves are overlapping with each other almost exactly. This indicates that the hidden layer neurons do not specialise in one particular input view. One view is directly subtracted from the other in each hidden layer neuron. Even when we increased the number of hidden layer neurons to four, the information of the two views did not stay separated in the hidden layer. A more detailed discussion of the properties of the weights in the perceptron case (similar to the weights between input

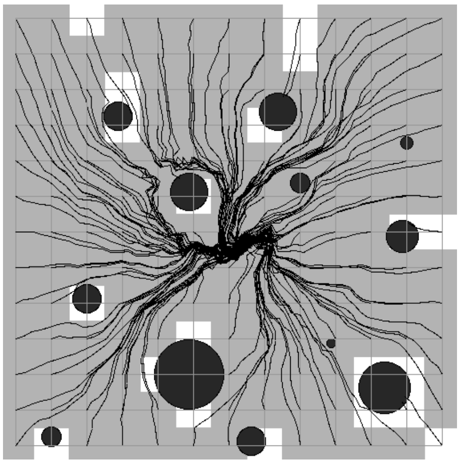

Figure 5: Trajectories of an agent trained with the data from figure 1 and tested in the same environment. If the home position was reached, the area around the starting position was marked in grey.

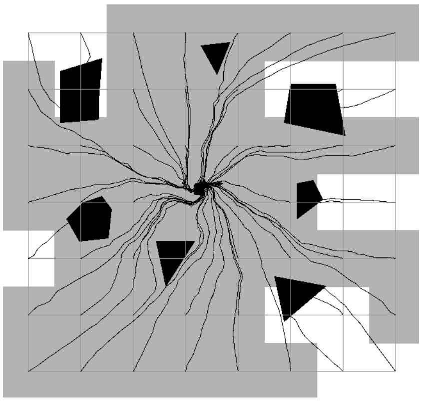

Figure 6: Trajectories of an agent trained in the 'cylinders world' and acting in the 'polyhedron world'.

and hidden layer in the MLP case) can be found in section 3.6.

In figure 8, the weights between the hidden and the output layer are shown. They map the hidden layer values (here: between 0 and 1) to the population code on the output layer. Examples for output activation patterns are shown in figure 9. If we use a sigmoid function with output values between -1 and 1, the weights between hidden and output layer directly map $\sin$ and $\cos$, the bias weight is constant (not shown).

3.4 Perceptron Performance

In figure 10, the error curves of the perceptron and the MLP are compared. The error curve for the 180-2-8

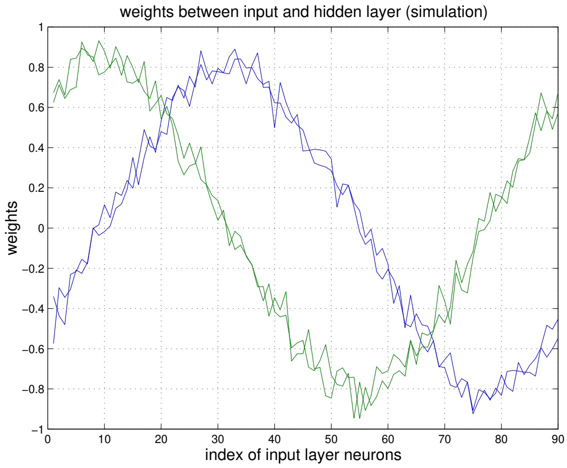

Figure 7: Weights between the input and the hidden layer of the 180-2-8 MLP arranged in the following way: The weights coming from neuron 0 to 89 are plotted normally, the weights coming from neuron 90 to 179 were shifted by a value of 90 to the left and multiplied by -1.

MLP starts with a higher error, but after a few iterations, both methods approach approximately the same error.

3.5 Perceptron Weights

We examine the weights for the case of the perceptron learning with just two output neurons each with 180 inputs. The output neurons directly encode the x and y component respectively of the homing vector. The input neurons n_0 to n_{89} encode the first view, n_{90} to n_{179} the second view. When trained in the world in figure 1, the weights shown in figure 11 were obtained. They are of similar shape as the weights between input and hidden layer of the MLP. The learning parameter η was gradually decreased from 0.1 by division by 1.1 after each run. This results in smoother weights than in the MLP case.

3.6 Analysis of the Perceptron Weights

As can be seen in figure 7 and 11, the weights for both the hidden layer neurons of the MLP and the output neurons of the perceptron are sine shaped with different phase and amplitude. In this section, we will show that this directly results from the interaction with the environment.

The output values are calculated using the following formula:

$$o^j = f(\sum_{i=0}^{179} w_{ij} n_i),$$

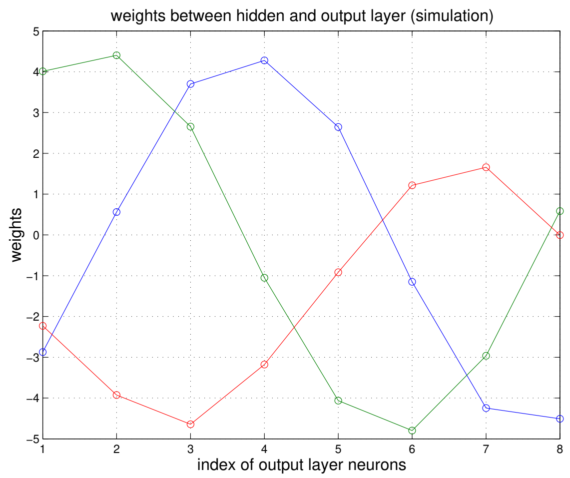

Figure 8: Weights between the hidden and the output layer of the 180-2-8 MLP. The weights starting at around -2 are the bias weights.

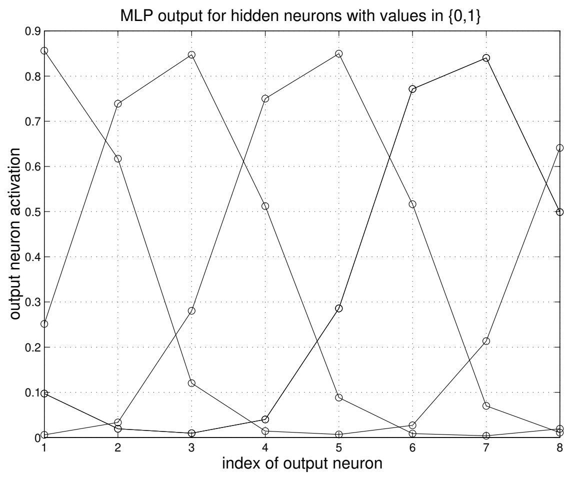

Figure 9: Different output activation patterns of the 180-2-8 MLP for hidden layer values in {(0,0), (0,1), (1,1), (1,0)}.

where w are the weights connecting input and output layer and f is a monotonically rising function (for example $f(x) = tanh(x)$ or $f(x) = x$). Each combination of two views results in a non-ambiguous output vector.

Since the network should be stable against small errors like shifts in the input views, the difference between neighbouring weights in the two views should be small. A small rotation of both views and home vector (which is the same as a small rotation of the environment or the robot) results in a small change in the output neurons. For an infinite number of input neurons, the weight functions for each view would be continuous. For each 90 input neurons, we get quasi-smooth weight curves.

Other characteristics of the system which directly result from the environment are:

1. If both views are rotated by α, the home vector is rotated by $-\alpha$.
2. If the two views are exchanged with each other, the home vector is rotated by π.
3. It follows: If the views are both rotated by π and exchanged, the home vector stays the same.

First, we examine the weights w for the perceptron encoding cos.
For simplification we set
$a_i = n_i$, $b_i = n_{i+45}$, $c_i = n_{i+90}$, $d_i = n_{i+135}$ and $w_{ai} = w_i$, $w_{bi} = w_{i+45}$, $w_{ci} = w_{i+90}$, $w_{di} = w_{i+135}$ for $i = 0, .., 44$, and set $l = 45$.

From (3.):

$$f(\sum_{i=0}^{l-1}(w_{ai}a_i + w_{bi}b_i + w_{ci}c_i + w_{di}d_i))$$

$$= f(\sum_{i=0}^{l-1}(w_{ai}d_i + w_{bi}c_i + w_{ci}b_i + w_{di}a_i))$$

$$\Rightarrow$$

$$\sum_{i=0}^{l-1}((w_{ai} - w_{di})(a_i - d_i) + (w_{bi} - w_{ci})(b_i - c_i)) = 0$$

Since this has to hold for all possible values of a_i, b_i, c_i and d_i,

$$w_a = w_d \text{ and } w_b = w_c$$

From (1.) with $\alpha = \pi$:

$$f(\sum_{i=0}^{l-1}(w_{ai}a_i + w_{bi}b_i + w_{ci}c_i + w_{di}d_i))$$

$$= -f(\sum_{i=0}^{l-1}(w_{ai}b_i + w_{bi}a_i + w_{ci}d_i + w_{di}c_i))$$

With the same explanation as above, it follows

$$w_a = -w_b \text{ and } w_c = -w_d$$

$$w_a = w_d \text{ and } w_c = -w_d \Rightarrow w_a = -w_c$$

$$\Rightarrow o = f(\sum_{i=0}^{l-1} w_{ai}((a_i - b_i) - (c_i - d_i)))$$

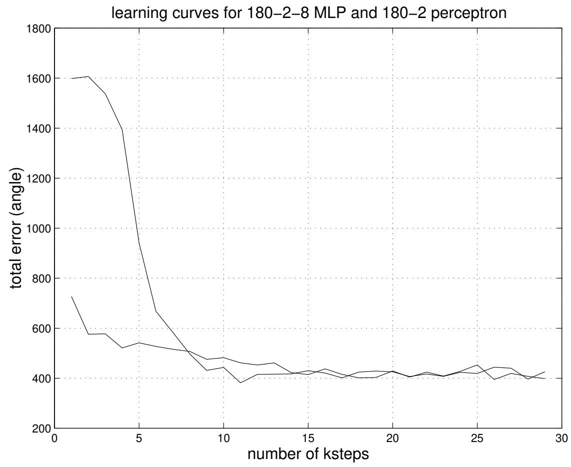

Figure 10: Performance of a MLP (180-2-8) and perceptron network (180-2) compared: The error is the summed squared error of the output angle. The error curve for the MLP starts with a higher error, but after a few iterations, both methods approach approximately the same error.

Let us consider the continuous case for input as well as weights, and set $r(x)$ as the input for $x = [0, .., \pi[$, and $w(x)$ as the weights for $x = [0, .., \pi[$ (corresponds to neuron a_0 to neuron a_{l-1}). Note that $r(x)$ and $w(x)$ are 2π periodic.

Since the weights are smooth, symmetric and periodic, $w(0) = -w(\pi)$, therefore $w(k) = 0$ for at least one $k \in [0, .., \pi[$.

Instead of rotating the views by an angle α to account for the home vector being turned by α, the weights w_a can be rotated by $-\alpha$.

Assuming $f(x) = x$, the output o of the perceptron rotated by α can be written as

$$o(\alpha) = \int_0^\pi w(x + \alpha) r(x) dx$$

If we adjust the rotation of the views in a way that $w(0) = 0$ (Note: $w(k) = 0$ for at least one $k \in [0, .., \pi[$), we get:

$$o(\alpha) = \cos(-\alpha) = \cos(\alpha)$$

(instead of $o(\alpha) = \cos(\alpha + \alpha_0)$, where α_0 is the corresponding phase shift).

The change in the activation of the first output layer neuron when the home vector is rotated by α is:

$$\frac{d}{d\alpha} o^1(\alpha) = -\sin(\alpha) = \cos(\alpha + \frac{\pi}{2})$$

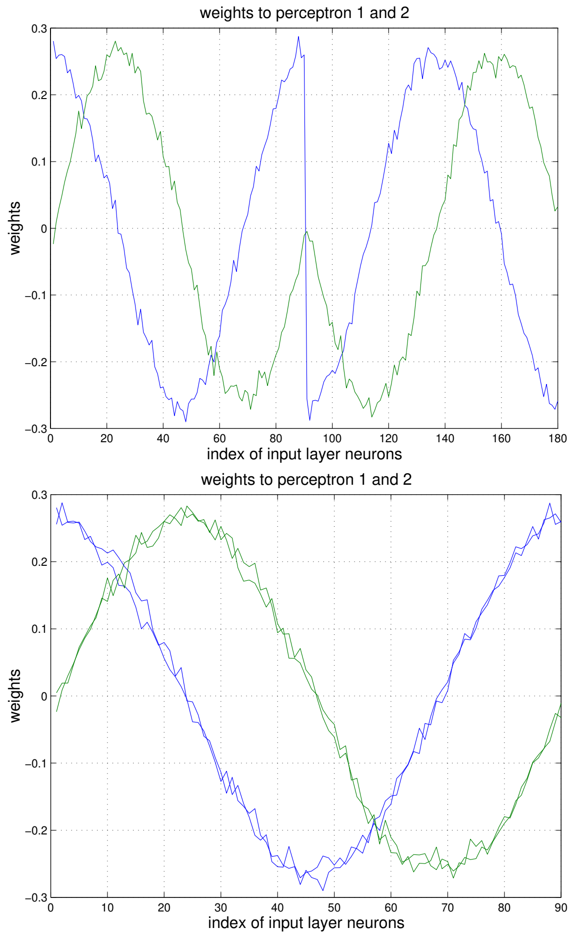

Figure 11: Weights for the perceptron. In the bottom image, the weights between the input and the output layer are arranged in the following way: The weights coming from neuron 0 to 89 are plotted normally, the weights coming from neuron 90 to 179 were shifted by 90 to the left and multiplied by -1.

$$\Rightarrow o^1(\alpha + \frac{\pi}{2}) = \frac{d}{d\alpha} o^1(\alpha)$$

$$\Rightarrow$$

$$\int_0^\pi w(x + \alpha + \frac{\pi}{2}) r(x) dx = \frac{d}{d\alpha} \int_0^\pi w(x + \alpha) r(x) dx$$

Since x, α are independent, it follows

$$\int_0^\pi w(x + \alpha + \frac{\pi}{2}) r(x) dx = \int_0^\pi \frac{d}{d\alpha} w(x + \alpha) r(x) dx$$

Since this equation has to hold for all possible functions $r(x)$, it follows

$$w(\alpha + \frac{\pi}{2}) = \frac{d}{d\alpha} w(\alpha)$$

This can be transformed to an ordinary (same variables) second order differential equation:

$$w(\alpha+\frac{\pi}{2}) = \frac{d}{d\alpha}w(\alpha) = \frac{d}{d^2\alpha}w(\alpha-\frac{\pi}{2}) = -\frac{d}{d^2\alpha}w(\alpha+\frac{\pi}{2})$$

$$\Rightarrow \frac{d}{d^2\alpha}w(\alpha) = -w(\alpha)$$

This is a harmonic oscillator with $w(0) = 0$ and $w'(0) = C$

$$\Rightarrow w(\alpha) = C\sin(\alpha+\alpha_0)$$

are the only solutions.
The other neuron with output o^2 encodes $\sin$, therefore:

$$w^2(\alpha) = C\sin(\alpha+\alpha_1) \text{ with } \alpha_1 = \alpha_0 - \frac{\pi}{2}$$

$$w^1(\alpha) = C\sin(\alpha+\alpha_0) = C\cos(\alpha+\alpha_1)$$

This shows that — if a homing vector can be learned — the weights between the views and the two output neurons are of the form $w^j(\alpha) = C\sin(\alpha+\alpha_{j-1})$.

Going back to the discrete case, we have:

$$\begin{aligned} o^j &= \sum_{i=0}^{l-1} w^j_{ai}((a_i-b_i)-(c_i-d_i)) \\ \text{let } k_i &= \frac{i\pi}{l}+\alpha_1 : \\ o^1 &= C\sum_{i=0}^{2l-1}\cos(k_i)(v_{1i}-v_{2i}), \\ o^2 &= C\sum_{i=0}^{2l-1}\sin(k_i)(v_{1i}-v_{2i}), \end{aligned}$$

where $v_1 = (a,b)$ and $v_2 = (c,d)$.

This is an important result since the resulting homing vector is now separated into its components and can be compared with the ALV model.

3.7 Comparison with the ALV Model

The Average Landmark Vector (ALV) model (Lambrinos et al., 2000) calculates the homing vector h by subtracting the AL vector at the target position from the AL vector at the current position:

$$h = ALV_{cur} - ALV_{tar},$$

where $ALV_{tar} = \sum_{i=1}^{n} lan_i^{tar}$ and $ALV_{cur} = \sum_{i=1}^{n} lan_i^{cur}$ with lan_i^{cur} and lan_i^{tar} being the landmark vectors. The assumption is that the number of landmarks in both views is the same, and that the size of the landmarks does not matter.

If we rewrite the resulting homing vector v_h from the perceptron, we get (let $k_i = \frac{i\pi}{45}+\alpha_1$):

$$\begin{aligned} v_h &= (o^1, o^2) \\ &= C(\sum_{i=0}^{89}\cos(k_i)(v_{1i}-v_{2i}), \\ &\quad \sum_{i=0}^{89}\sin(k_i)(v_{1i}-v_{2i})) \\ &= C((\sum_{i=0}^{89}\cos(k_i)v_{1i}, \sum_{i=0}^{89}\sin(k_i)v_{1i}) \\ &\quad -(\sum_{i=0}^{89}\cos(k_i)v_{2i}, \sum_{i=0}^{89}\sin(k_i)v_{2i})) \end{aligned}$$

When being normalised, the resulting homing vector v_h is equivalent to the homing vector h of the ALV model.

4 Real World Experiments

4.1 Training data

We investigate whether the same learning strategies can also be applied to real world visual data. The one-dimensional panoramic view is taken from the camera image of the mobile robot 'Samurai' moving around in an office room. The image itself as well as the robot position where the image has been taken, and the orientation of the robot, which is determined by a fluxgate magnetic compass, bear some noise. In addition, we do not have a fixed number of easily separable landmarks, but the brightness values of the image are used as input to the neural network.

Figure 12: Mobile robot 'Samurai' which has been used for the real world experiments. It is equipped with an omnidirectional camera, a fluxgate magnetic compass, and differential steering.

The image preprocessing is performed as follows: The original image (see figure 13) is transformed to a rectangular panoramic view. This image is vertically averaged and the resulting brightness curve is smoothed, normalised and transformed to 1×90 pixels (see figure 14).

We recorded 80 images on a 8×10 grid of 20×20 cm spacing. This results in $80^2 - 80 = 6320$ different input vectors with 90 different orientations, which are in total 568800 different input vectors.

Figure 13: World as seen through the robot's camera. The image has been taken in an unmodified office room.

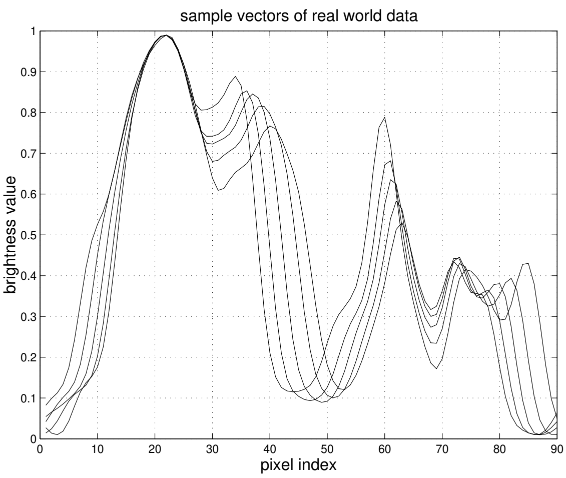

Figure 14: Examples of one-dimensional views of real world data as it is fed into the neural network. The images have been taken in a unmodified office room on a 8×10 grid with 20×20 cm squares.

4.2 MLP Performance Real World

The recorded images are used as the input for the MLP. Several filters for the image preprocessing have been tested. The filter that turned out to produce lower errors in training than the unfiltered view v so far is $v(x) = 0$ for $v(x)' \leq 0$ and $v(x) = v(x) * v(x)'$ for $v > 0$, ($v(x)'$ is valid since v has been smoothed).

Like in simulation, the network is learning an algorithm to perform homing; the total error is — as expected — higher than in simulation. In figure 15, the learning curves for different numbers n of hidden layer neurons are compared. From top to bottom, $n = 1, 2, 4, 16$.

With $n = 2$, the output angle is within a range of $\pm 45°$ in $\approx 84\%$ of the trials ($\approx 88\%$ in simulation) and within a range of $\pm 90°$ in $\approx 96\%$ of the trials ($\approx 97\%$ in simulation). This can be seen in a graph plotting the sorted angle errors of one run after learning (see figure 16).

The simulation environment has been used to plot the homing vectors of the real world data (figure 17). The vectors are more exact if the current view position is not in the direct vicinity of the target view position.

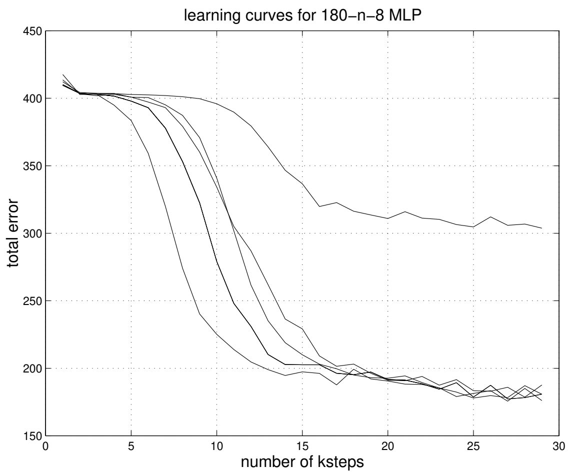

Figure 15: Learning curves with the real world data (filtered) and different number n of hidden layer neurons. From top to bottom, $n = 1, 2, 4, 16$.

4.3 MLP Weights Real World

The weights between input and hidden layer of the filtered version (100 learning cycles) are shown in figure 18. They are comparable to those learned with the data from the simulated environment.

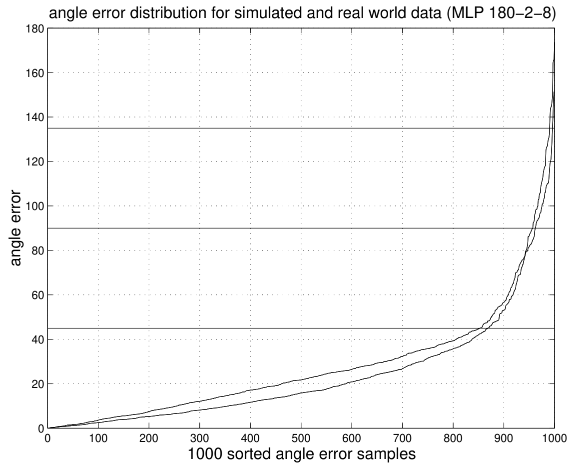

Figure 16: Sorted angle errors of the MLPs with real world and simulated data for 1000 random trials after learning. The errors for the simulation are smaller.

4.4 Performance on the Mobile Robot

A mobile robot (see figure 12) that has been trained offline with the image database is used for testing the resulting neural network in the real world. For this purpose, the robot is steered manually to a desired target position and is told to record a target view. After that, it is again manually steered to another position, and is told to find its way back to its target location. The only information the robot is allowed to use apart from the trained neural network are the two views and their orientation. The target position view and the current view (aligned to the target position view) are used as the input for the neural network, the output is used as the homing vector v_h.

The motor commands for the two wheels (differential steering) are easily gained from the transformed homing vector v_h (see figure 19). The robot navigates back autonomously into the direct vicinity of its home position (distance smaller than 50cm) in most of the trials when the neural network has been trained with all recorded views, it navigates back in less than half of the trials, if the neural network has been trained with all possible rotations of all recorded views.

We hope to further improve the performance by implementing more suitable image preprocessing.

5 Discussion

5.1 Results

We show that it is possible to learn a navigation algorithm, which allows visual homing that is widely independent of the environment. This mechanism uses two views to determine the direction from the location of one view to the location of the other. Such mechanisms have been suggested before, for example the snapshot model or the ALV model, but in these models the homing mechanism was pre-defined and not learned.

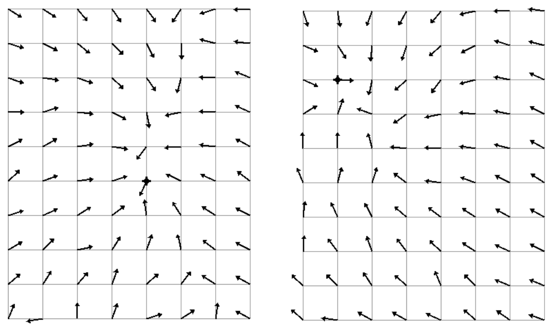

Figure 17: Homing vectors learned from the real world images taken with the mobile robot 'Samurai'. The 80 images were learned in pairs in all orientations. The vectors are more exact if the current view position is not in the direct vicinity of the target view position.

Figure 18: Weights between input and hidden layer for the filtered real world version with rotated views.

The proposition we make is that such an algorithm can be *learned*, more or less from zero knowledge, simply by performing some exploration tours. This does not exclude the possibility that some ability to perform homing is already present at birth in some animals, but even in this case, the algorithm has to be adapted to the agent's body and sensors by learning specific parameters. For example in the analogue robot (Möller, 2000) implementing the ALV model, the synaptic weights

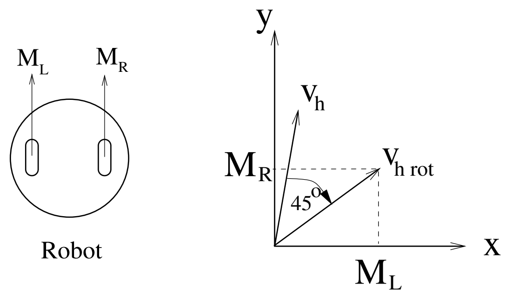

Figure 19: Transformation of the homing vector v_h to motor commands of the mobile robot. The vector is rotated clockwise by 45° to $v_{h_{rot}}$, the x component of $v_{h_{rot}}$ is used for the left wheel, the y component for the right wheel of the robot.

have to be explicitly coded with $\sin$ and $\cos$ values. Here we show that the weights can be adapted during a learning phase.

Another important result of this paper is the similarity between our learned navigation mechanism and the ALV model. The ALV hypothesis is strengthened since a simple learning strategy directly resulted in this model. It also suggests that there is no need for direct genetic encoding of the model in the animal, the existence of a basic neuronal structure with the ability to learn is sufficient. This has the advantage that the navigation mechanism (here: the weights) can adapt to both the specific sensor setup (type and morphology) and regularities of the environment.

The learning is self-supervised, this means that the agent just has to be aware of both the two views (input) and the movement vector connecting them, obtained from path integration (output), but does not require an external teacher. The learned homing behaviour is consistent with the homing behaviour observed in desert ants (Lambrinos et al., 2000).

5.2 Future Work

The next step will be an investigation into more suitable image preprocessing of the real world data to get more reliable results on the mobile robot. This technique might be adaptive to the specific environment (indoor, forest, desert etc.).

When this is achieved, the neural network weights should be learned online while the mobile robot is exploring its environment autonomously. Using path integration, the robot should be able to steadily increase the distances to its home position.

References

Cartwright, B. A. and Collett, T. S. (1983). Landmark learning in bees. *Journal of Comparative Physiology*, 151:521–543.

Franz, M. O. and Mallot, H. A. (2000). Biomimetic robot navigation. *Robotics and Autonomous Systems*, 30:133–153.

Georgopoulos, A. P., Schwartz, A. B., and Kettner, R. E. (1986). Neuronal population coding of movement direction. *Science*, 233:1416–1419.

Lambrinos, D., Maris, M., Kobayashi, H., Labhart, T., Pfeifer, R., and Wehner, R. (1997). An autonomous agent navigating with a polarized light compass. *Adaptive Behavior*, 6:131–161.

Lambrinos, D., Möller, R., Labhart, T., Pfeifer, R., and Wehner, R. (2000). A mobile robot employing insect strategies for navigation. *Robotics and Autonomous Systems, special issue on Biomimetic Robots*, 30:39–64.

Möller, R. (2000). Insect visual homing strategies in a robot with analog processing. *Biological Cybernetics, special issue: Navigation in Biological and Artificial Systems*, 83:231–243.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.