Chapter 6

Training

6.1 General

This chapter provides implementation details of the training strategy that was stipulated in the *Specification* stage.   This chapter begins by identifying the variables that are involved in training the behaviour modules as well as an overview of how the “default” values of the variables were determined. The training of the obstacle avoidance and light seeking modules is then discussed. Both modules are trained in a real environment with a real Khepera as well as in the simulator provided with the *Khepera Toolbox*. Finally, the training of the behaviour integration module is discussed.

6.2 Network Training Variables

6.2.1 Summary of Variables

Table 6-1 is a list of variables that were required to be set prior to the training of each of the modules. A description of each variable (most were visited previously) is given along with its “default” setting.

Network Variables		
Variable Name	**Description**	**Default**
LR Reward	Learning rate for rewards received	0.3
LR Punish	Learning rate for punishments	0.1
thresh	The difference between T and S at which the network is considered trained	0.01
tscale	Scaling factor for the sigmoid function in the hidden layer	0.04
randscale	Scaling factor for the variance of the random number generator	40
s1	Number of neurons in the hidden layer	8
s2	Number of neurons in the output layer	2
mot_delay	A loop delay variable which defines the granularity of each Khepera movement	60000

Table 6-1
List of Module Variables

6.2.2 Selection of Variable Default Values

The learning rates for reward and punishment were set according to the work by [Meeden94] who determined empirically that 0.3 and 0.1 (respectively) worked well. When an action is rewarded, it is clear that the action was justified and the rate of learning should reflect that justification. When an action is punished, the opposite action is used as a training target but it is not known if this action was the correct one given the circumstances, so the learning rate should not reflect a large confidence. The learning rates for punishment should thus be selected lower than that of reward and both should have magnitudes that promote stable learning in the network.

The value of *thresh* was set arbitrarily to bound the size of the network outputs and prevent *saturation* of the output neurons. Saturation occurs when the activity level at the input to a neuron causes the output of the activation function of that neuron to be very close to the limiting value of that function. When this occurs, the learning process can be slowed down by orders of magnitude and, in some cases, stopped altogether.

For example, Figure 6-1 shows the activation function response for the output layer (a log sigmoid). The x-axis represents the activity level at the input to the neuron which is just the weighted sum of all of the inputs from the neurons in the previous layer to which it is connected. Assume that this weighted sum equalled, say, 50. The activation function response (ie, the neuron output) using Figure 6-1 would be very, very close to 1.

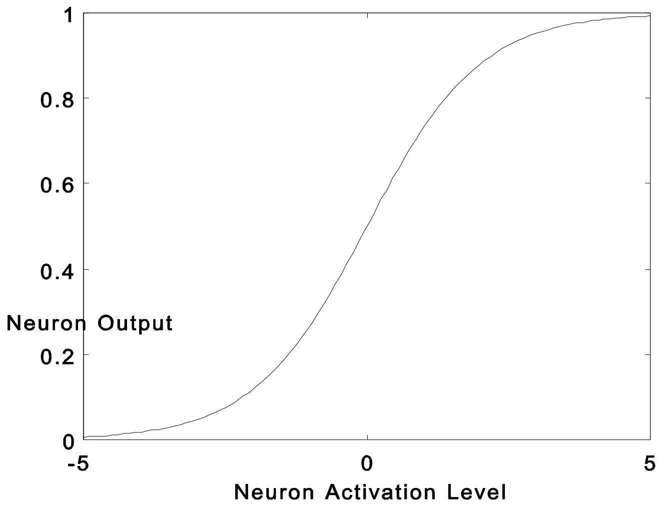


Figure 6-1
Log Sigmoid Activation Function

The computation of the weight changes to the output layer begins with a computation of the local gradient. Recall that the local gradient for neuron *i* of the output layer was defined as

$$\delta_i=(d_i-y_i)!\ F!(V_i)$$

where V_i represents the weighted sum of all of the inputs from the neurons in the previous layer and y_i is the activation output of neuron *i* (that is y_i=F(V_i)). By taking the derivative of the sigmoid activation function in the output layer, the above equation becomes

$$\delta_i=(d_i-y_i)!\ y^{!}_{iÀ}\underline{\quad}y_i)$$

The significance of of having an activation output y_i which is close to the limiting value of 1 becomes clear. The closer to 1, the smaller the local gradient, and the smaller will be the changes to the weights affecting that neuron. The weight changes are computed as

$$\Delta w = lr\grave{A}!\_\_\_!\_\_LDy_1$$

where *lr* is a learning rate parameter ($0<lr<1$) and $\mathbf{y}_1$ is a vector of activations from the previous layer. In fact, when the activation of the neuron gets so close to 1 that it is set to 1 by the activation function (**logsig.m**), the local gradient *becomes* zero and learning in that neuron is frozen.

In order to keep the output neurons from saturating, it therefore makes sense to prevent the activations of each neuron from moving too close to the limiting values of the activation function. This is accomplished for the output layer by comparing the value of *T*-*S* (the difference between the target and network output vectors) to *thresh*. If all of the elements of the difference vector are less than *thresh*, then back-propagation of the error does not occur and the network outputs do not progress any further towards the limiting value. Effectively, training is stopped when the network outputs are sufficiently close to the target for the given state-action pair.

The variable *tscale* was introduced to give control over the scaling of the hyperbolic tangent transfer function of the *hidden* layer in order to prevent the saturation of the neurons there. Unlike the output layer, we could not stop the back-propagation process if the hidden layer outputs were within some threshold of the limiting values of the activation function. However, by providing a scaling factor, *tscale*, as an argument to the activation function, we could squash the function and widen its active region. By carefully examining the values appearing at the inputs to the neurons in the hidden layer during some typical time frame, it was possible to anticipate the maximum weighted sum and then adjust *tscale* so that this value would not approach too close to the limiting values of -1 or 1.

The process of "squashing" shown in Figure 6-2. Figure 6-2A shows the default sigmoid while Figure 6-2B shows the sigmoid with a *tscale* value of 0.04. The value of 0.04 was determined experimentally by observing the activation levels on the hidden layer during the operation of the obstacle avoidance module (with inputs represented by 10 or 0 via the input representation function).

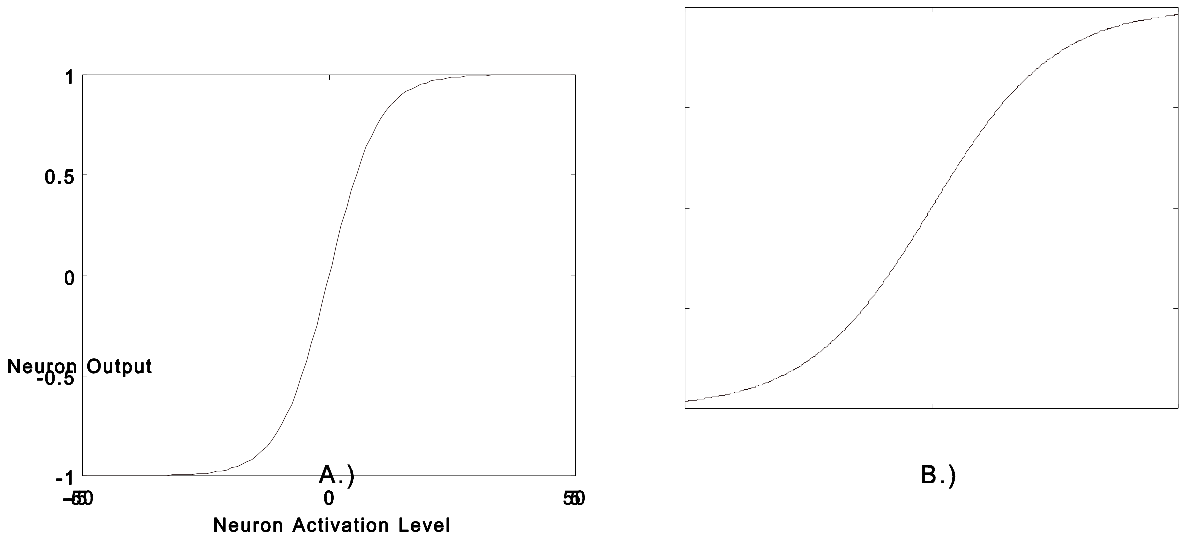


Figure 6-2
A) Default Hyperbolic Transfer Function B)tscale=0.04

The variable *randscale* is used to adjust the variance of the Gaussian random exploration rule. Random numbers are computed in **xplore.m** for the Gaussian exploration rule by the Matlab function **randn**. These numbers are initially normally distributed with a mean of zero and variance of 1. The mean is reset to 0.5 by adding this quantity to these numbers. The variance is adjusted by dividing the numbers generated by **randn** by the quantity in *randscale*. A value of 40 sets the distribution of random numbers squarely between 0.4 and 0.6 as shown in Figure 6-3 below. Thus, random exploration can only occur when the values of the output layer fall in this region.

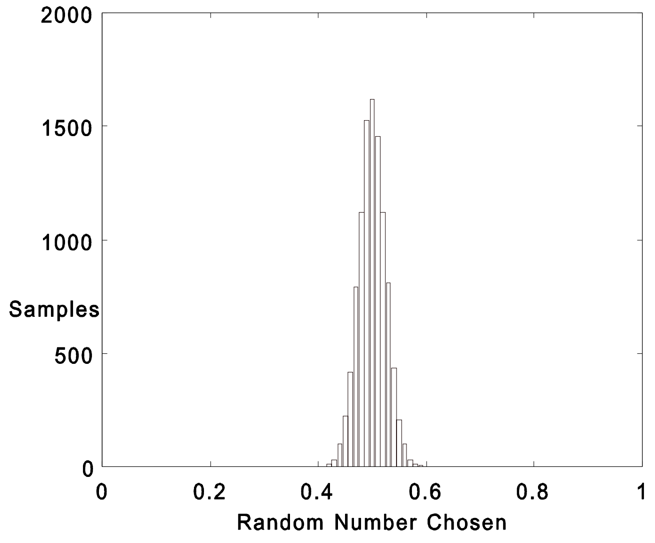


Figure 6-3
Histogram of Gaussian Distribution
for *randscale*=40

The variables *S1* and *S2* set the size of the hidden and outputs layers. The input layer size is fixed by the number of sensors being used by the network and the size of the context layer. The output layer size, *S2*, is fixed by the desire to have two bits to select action sizes.

It has been demonstrated that one hidden layer is all that is required for a multilayer neural network to approximate an arbitrary function ([Hornik89],[Funahashi89]). Thus, more than one hidden layer was not seen as necessary. However, there is currently no ideal method to automatically determine the number of neurons in a hidden layer required to perform a desired approximation [Rummery95]. Given that the sensor data for both light seeking and obstacle avoidance are passed through an input representation function (which greatly simplifies the search space of the network), various values of *S1* between 5 and 12 were investigated to evaluate the effect of network learning speed and convergence to the desired behaviour. A value of *S1*=8 was determined to satisfy the training requirements of both modules.

The final variable that required setting prior to experimentation is *mot_delay*. This variable sets the number of times the computer perform a “do nothing” loop just after sending a motion command to the Khepera. The requirement for this delay is to allow the Khepera to actually move before the sensors are sampled by the training module. Rewards and punishments can only be assigned when there is an observable difference between the current state of the robot and its last state. Without a delay, the robot will barely move before the sensors are sampled by the trainer making the assignment of the reinforcement more difficult. This variable corresponds to setting the granularity of each movement.

6.3 Behaviour Module Training: Obstacle Avoidance

It was postulated previously that an exploration rule based on a Gaussian distributed random numbers (versus a uniform distribution) would be more effective for learning in CRBP. This hypothesis was evaluated in the context of training the obstacle avoidance module only. For each training rule, five training runs of 5000 cycles each were performed in the real world training environment. A cycle consisted of one complete iteration of CRBP algorithm. In order to evaluate the effectiveness of the Khepera Simulator, five more training runs using each of the training rules was performed in the simulated training environment. The weights and biases of each of the simulated runs were then transferred to the real robot and training continued for another 1000 cycles.

6.3.1 Real World Training Environment

The walls of the playpen were arranged as shown in Figure 6-4. The robot was constrained to the corner area. This smaller area provided a richer mixture of training situations than having the robot move about in the entire space available in the playpen.

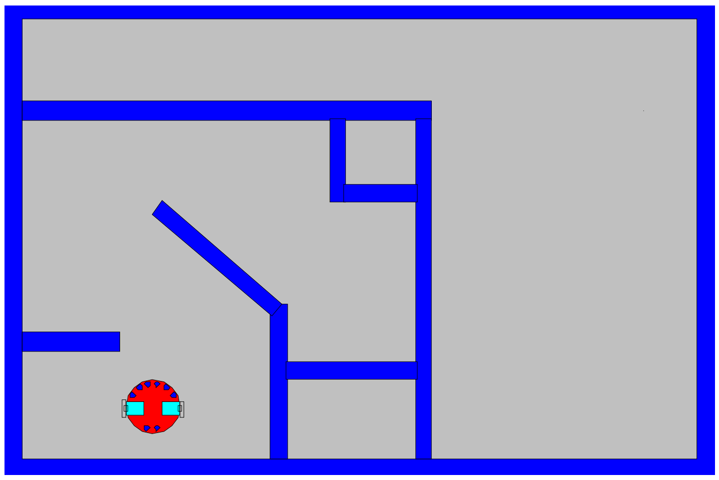


Figure 6-4
Playpen Arrangement for Training in Real World

6.3.2 Simulation Training Environment

6.3.2.1 Sensor Calibration

Normally, before using the Khepera simulator, the sensor model would require calibration with the real robot in the intended environment. This calibration procedure is outlined in the *Khepera Toolbox User's Guide*. However, in this case, the simulator was created using the actual Khepera under study so the proximity sensor model is already calibrated.

6.3.2.2 Simulation Environment

The training playpen was created using the **Utilities**! **Khepera Simulation**! **Create New Playpen** option in the *Roborunner* graphical user interface of the *Khepera Toolbox*. This playpen is shown in Figure 6-5. The simulaton symbol for the Khepera was abstracted as shown in the Figure in order to reduce programming overhead.

Figure 6-5
Simulator Training Environment

The simulated playpen is conceptually similar to the real environment. The graphic rendered in Figure 6-5 uses false colours so that it can be more easily displayed in black-and-white.

6.4 Behaviour Module Training: Light Seeking

The light seeking module was trained using the Gaussian distributed random exploration rule only. Five modules were each trained for 5000 cycles and one was chosen for placement in the main robot controller. As with obstacle avoidance, five more modules were trained for 5000 cycles each in the Khepera Simulator and then transplanted to the real Khepera for a further 1000 cycles of training.

6.4.1 Real World Training Environment

Lamps were arranged at either end of the playpen as shown in Figure 6-6 but only one was on at a time. These lamps were sequenced by hand during learning so that, as the Khepera approached to within several centimetres of a lamp that was on, it was switched off and the other was illuminated. In this manner, many instances of having to find the light and then approach it were generated.

Since the objective of the learning was to approach light sources, no obstacles were involved in the training of this module (except for the retaining walls).

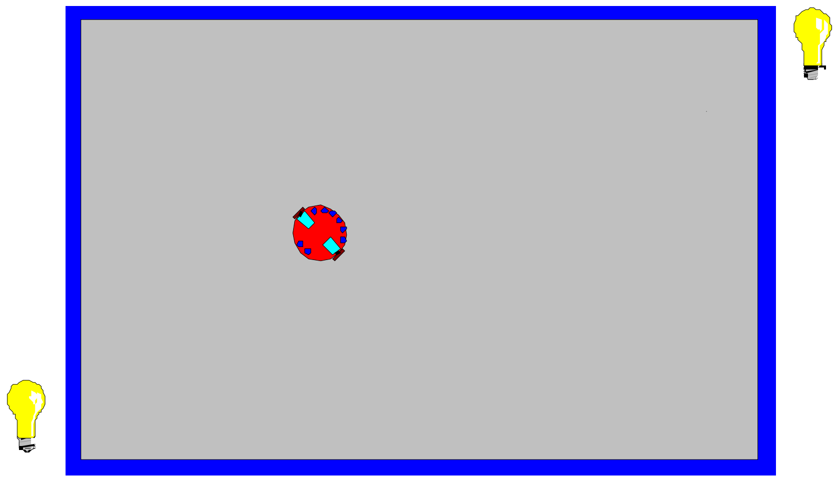


Figure 6-6
Playpen Arrangement for Training in Real World

6.4.2 Simulation Training Environment

6.4.2.1 Sensor Calibration

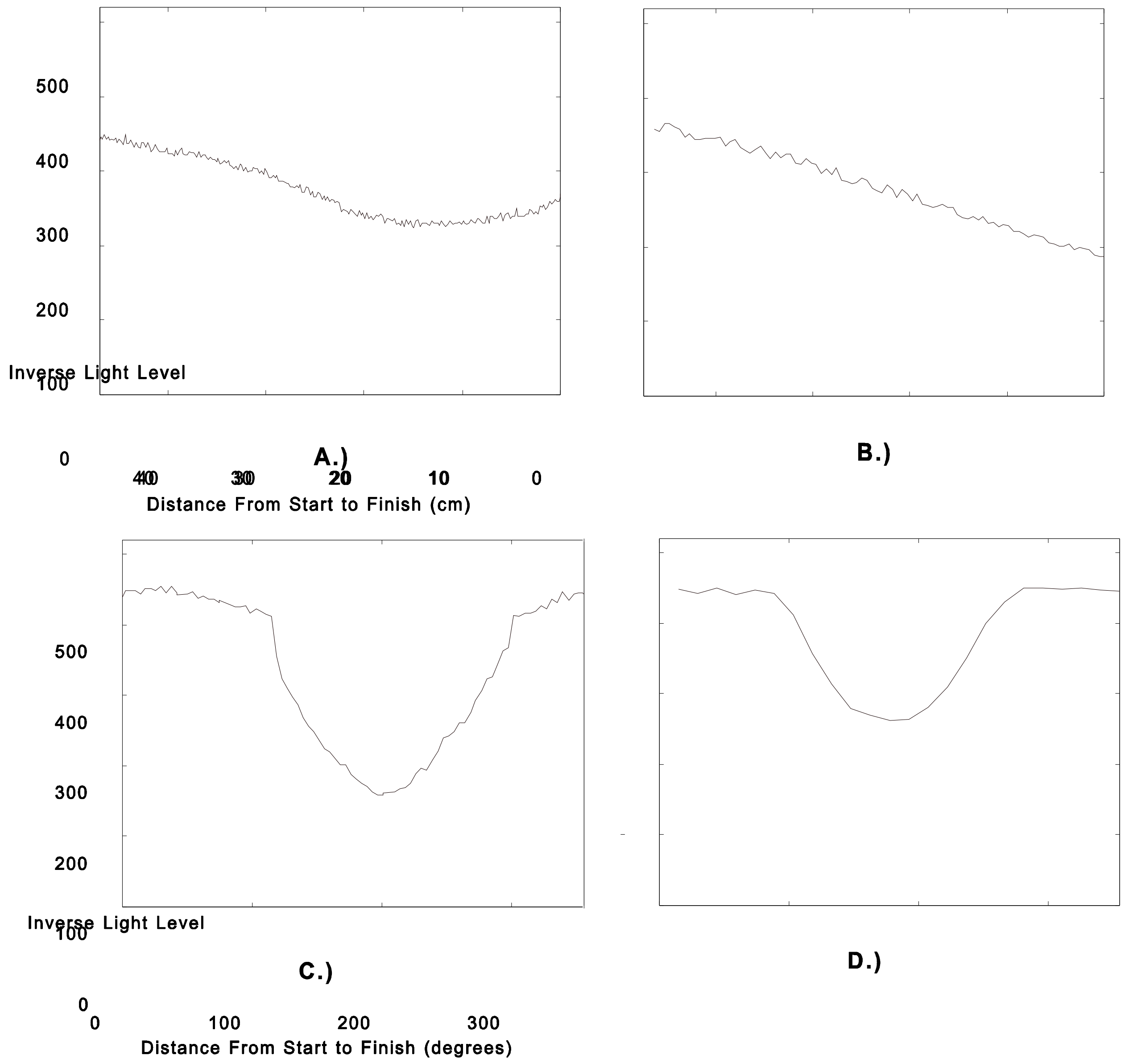
The values produced by the ambient light sensors on the Khepera depend to a great extent on the light environment in which the robot is situated. For example, the distance from any sensor to a light source, the intensity of the source, its colour, vertical position, and ambient and reflected light are all factors that affect the reading. It is therefore not possible to construct a "universal" model of the sensor response that will work in all environments.

Thus, calibration of the simulator sensors were required to bring the modelled response closer to that of the response expected in the actual environment under study. The procedure for this calibration is given in the *Khepera Toolbox User's Guide* and all of the software tools are provided as part of the toolbox. This calibration was performed using the real environment described above. Figure 6-7 shows the response of the real robot and the simulated robot during both a linear approach test and a rotational test from a fixed distance. The graphs for the linear approach test show the left 10 degree sensor while the rotational test graphs show the back right sensor. The left 10 degree sensor was chosen since it points directly into the light throughout the test and therefore best describes the effect of distance and Wattage on the sensors. The back right sensor was chosen since it allows the total arc of the "dip" in the readings to be measured easily.

The values of the calibration variables required to produce the results of Figure 6.7 are shown in Table 6-2 below. As outlined in the calibration procedure, the variables were hard coded in **`upsv.m`** after they were determined.

Calibration Variables	
Amblite	450
Squash	0.045
Xlate	200
Max_angle	110
Cutoff	0
Noise	5

Table 6-2



Calibration Variable Settings for Simulated Light Sensors
Figure 6-7
Calibration Results
A.) Linear Approach to Light Source, Right 10 Degree Sensor, Real Khepera
B.) Linear Approach to Light Source, Right 10 Degree Sensor, Simulated Khepera
C.) 360 Degree Turn at 20 cm From Light Source, Real Khepera
D.) 360 Degree Turn at 20 cm From Light Source, Simulated Khepera

Note that the modelled responses of Figure 6-7 are only rough approximations. For the work being done here, this is approximation is appropriate. Since all of the light readings are passed through an input representation function that searches for the minimum, the actual values of the sensor responses do not need to be exact. However, if a closer response was required, more time could be spent iterating the calibration procedure and adjusting the actual environment until the simulated response was satisfactory.

6.4.2.2 Simulation Environment

Creation of the training playpen was not required. From the *Roborunner* interface, the **Open Default** button was selected. The default environment is basically an empty playpen (i.e. without any obstacles). For training purposes, four lamps were placed near the centre of each of the four retaining walls. During the training, the lamps were sequenced to the on state one at a time by selecting **Sequenced** from the **Lamp Control** pulldown menu. The environment (in false colours) is shown in Figure 6-8.

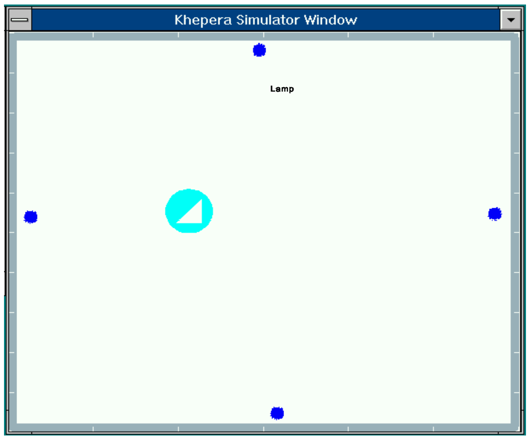


Figure 6-8
Simulator Training Environment

6.5 Integration Function Training

As mentioned in the *Specification* stage, it was opted to provide the integration function via a switch that does not require training.