17 Learning to Detour[1]

F. J. Corbacho and A. Weitzenfeld[2]

17.1 Introduction

Anurans (frogs and toads) show quite flexible behavior when confronted with stationary objects on their way to prey or when escaping from a threat. *Rana computatrix* (Arbib, 1987), an evolving computer model of anuran visuomotor coordination, models complex behaviors such as detouring around a stationary barrier to get to prey on the basis of an understanding of anuran prey and barrier recognition, depth perception, and appropriate motor pattern generation mechanisms based on sensory perception. This chapter presents a model of detour in *Rana computatrix* with an extension to learning of new schemas "How are schemas combined to form new schema assemblages acquired for the system to become more efficient?" We describe the construction mechanisms and interactions with the environment necessary to achieve higher levels of detour performance. This chapter describes a model that includes all these phenomena implemented in NSL. More details on some of the model components can be found in (Corbacho and Arbib, 1995) whereas Corbacho et al. (1996) present more behavioral data. This is a specific model in Schema-based Learning (SBL) but it serves to exemplify some of the general points and mechanisms included in the general framework of SBL. For the general framework we refer the reader to (Corbacho, 1998).

In this chapter we present a Schema-based model of learning to detour including different schemas implemented in some cases as functional units and in other cases as neural networks. The motivation for the study of Learning to Detour in frogs as our case study in Schema-based learning (SBL) is three-fold:

1. SBL is constrained by data on a neuro-ethologycally sound system -both the task, the environment and the agent.
2. The study of Rana Computatrix allows for horizontal integration (across many integrated functionalities) and not just vertical integration (action-perception within one central functionality, e.g., saccadic eye movements).
3. Learning to Detour has proved to be a very adaptive process relaying on important processes of learning (Corbacho et al., 1996).

Problem Background

Ingle (1983) and Collett (1983), to cite some examples, have observed that a frog/toad's approach to a prey or avoidance from a threat are also determined by the stationary objects in the animal's surround. A frog or toad, viewing a vertical paling fence barrier through which it can see a worm, may either approach directly to snap at the worm, or detour around the barrier. However, if no worm is visible, the animal does not move. Thus, it is the worm that triggers the animal's response but, when the barrier is present, the animal's trajectory to the worm changes in a way that reflects the relative spatial configuration of the worm and the barrier. Corbacho and Arbib (1995) modeled the different behavioral responses to different barrier configurations, as well as the learning involved in the behavioral transitions. The present section is based on behavioral studies of frogs, *Rana pipiens* (Corbacho et al., 1996). Here we sample a few of our observations of the main capabilities of frogs for detour behavior that set challenges for our learning model.

Experiment I: Barrier 10 cm Wide

Frogs that started from a long enough distance (15–25 cm) in front of a 10cm wide barrier (and with the worm 10 cm behind the barrier) showed (in 95% of the trials) reliable detour behaviors from the first interaction with the 10 cm barrier. They produced an immediate approach movement towards one of the edges of the barrier (see 17.1A). This experiment shows that an adult frog has the capability without training to perform detours when the barrier is narrow enough (10 cm long) and the frog is at a far enough distance (15-20 cm) from the barrier.

Experiment II: Barrier 20 cm wide

From now on we will refer to a frog which has not been exposed to the barrier paradigm as naive. If the chopsticks are placed the same distance apart, so that the gaps have the same width, and the barrier is 20 cm wide, then the naive frog tends to go for the gap in the direction of the prey (this was the case for 88% of the trials). The frog starts out approaching the fence trying to make its way through the gaps. During the first trials with the 20 cm barrier the frog goes straight towards the prey thus bumping into the barrier. When the frog is not able to go through a gap towards the prey it backs-up about 2 cm and then reorients towards one of the neighboring gaps (see figure 17.1B).

Observation: After 2 (43%) or 3 (57%) trials, the frog is already detouring around the barrier without bumping into the barrier (see figure 17.1C). The behavior involves a synergy of both forward and lateral body (sidestep) movements in a very smooth and continuous single movement.

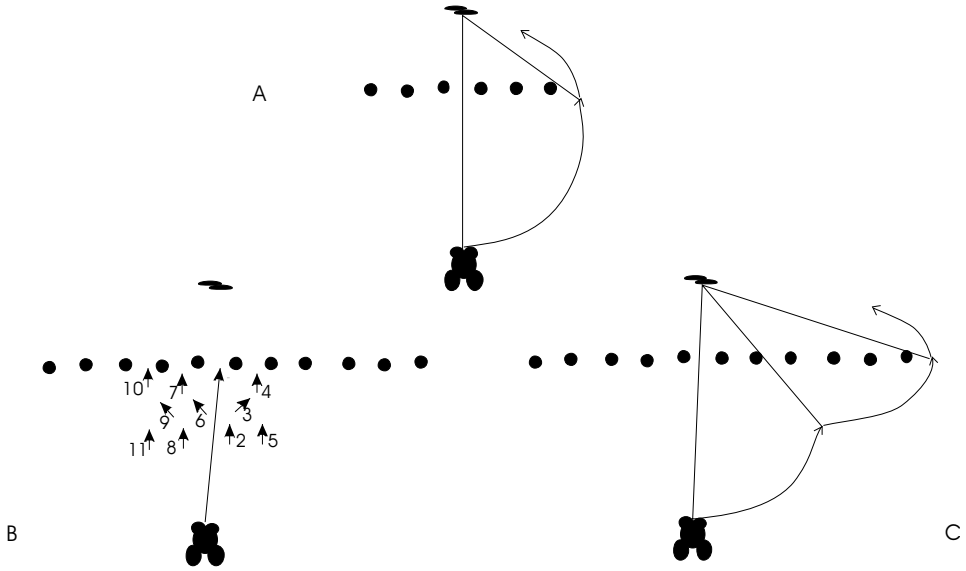

Figure 17.1
A. Approach to prey with single 10 cm barrier interposed. B. Approach to prey with single 20 cm barrier interposed: first trial with frog in front of 20 cm barrier (numbers indicate the succession of the movements). C. Approach to prey with single 20 cm barrier interposed: after 3 trials with frog in front of 20 cm barrier. Arrowheads indicate the position and orientation of the frog following a single continuous movement after which the frog pauses.

17.2 Model Description

We start by defining the environment and the agent (frog in this case). The environment provides the agent with an interaction space. Ultimately the behavior of any agent is very dependent on its environment so that the behavior can only be understood in relation to the synergy agent-environment. In order to define the structure of the agent we start by defining the spaces of interaction/communication with the environment and then follow with the functional units that constitute the agent.

Definition. An *Environment* is a space that includes a collection of entities and their relations (interactions). A particular instance configuration at time *t* will be denoted as

Environment(t). *Environment* is a 150x150 grid where different entities e.g., *frog*(x_f,y_f), *barrier*(x_b,y_b,*wh*,*g*). The simulation system contains simplified *Environment* functions designed to allow for an adequate interaction between the simulated agent and its environment, for instance the simulation system performs simple "shifts" of the agent's visual field as it moves in the environment and its coordinates change. The environmental functions will be described in more detail in the Model Architecture section.

Basically, the visual field of the agent corresponds to a sector of the *Environment*, and the coordinates of this sector are updated as the agent moves around. This 2D sector corresponding to the agent's visual field is projected upon the retina of the agent, which is the front-end visual perception system. The agent may also perform several actions that may cause environmental and agent parameters to change.

Component Schemas: Architecture

The detour model incorporates schemas (functional units) and neural modules (structural units) described in table 17.1 and shown in figure 17.2.

Table 17.1
Frog schemas according to their functional (schema level) and structural organization (neural level).

Function	Schema Level Modules	Neural Level Modules
Perceptual	Visual, Depth, Tactile, PreyRec, SoRec	Retina, T5_2layer, TH10layer
Sensorimotor	PreyApproach, SoAvoid	Motor Heading Map (MHM)
Motor	Forward, Orient, Sidestep, Backup	

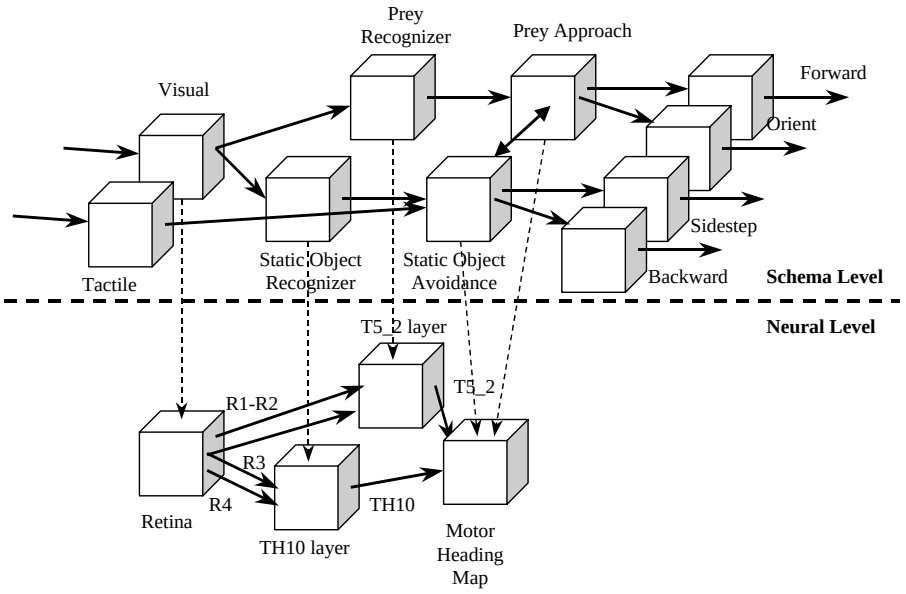

Figure 17.2
Schema Architecture for Detour Model consisting of two levels: a schema level and a neural networks level. The schema level consists of Perceptual Schemas: Visual and Tactile, Prey Recognition, Static Object Recognition (*SOR*); Sensorimotor Schemas: Prey Approach and Static Object Avoidance; and Motor Schemas Orient, Forward, Sidestep and Backup. The neural level consists of four modules: Retina, T5_2layer, TH10layer and the Motor Heading Map (*MHM*).

Perceptual Schemas

Perceptual schemas involve both sensors and recognizers based on these sensors.

Visual

The Visual schema simulates a visual sensor discriminating among different objects in the visual field, mainly prey and barrier in this model.

Depth

The Depth schema generates a depth map for the objects of interest, primarily barrier in order to avoid hitting it and generating appropriate responses according to how close the frog is to it.

Tactile

When the frog hits an object, in the current case the barrier, the Tactile schema gets triggered. The simulation environment checks when the frog comes level with the barrier (equal y-coordinates), and then checks whether there is a passable opening (we have chosen 3 cm wide or more for our simulations—this would change as the frog grows) at the frog's current x-coordinate. If the gap is not passable then the Tactile schema gets triggered:

$$Tactile = \begin{cases} 1 & \text{if } f_y = b_y \text{ and the closest to } f_x \text{ is less than 3cm wise} \\ 0 & \text{otherwise} \end{cases} \quad (17.1)$$

where (f_x,f_y) are the (x,y) coordinates of the "snout" of the simulated frog in the 2D world, and b_y is the "depth" coordinate of the barrier.

Prey Recognizer

Cervantes-Perez et al. (1985) presented a detailed neural network implementation for prey recognition. Here we present a schema (PreyRec) that approximates this neural network mapping. The presence of prey within the visual field of the animal produces a 2D pattern of activity in the prey recognition system, while absence of prey leaves the system at rest. This is here implemented by simplified feature detectors but it is open to more detailed implementations.

Ewert (1971) found in toad's pretectum near the ventral part of the pct (postero-central thalamic nucleus), units that give continued discharge in the presence of a large dark stationary object. This occurred even when the stationary object was revealed by turning on the room lights without prior motion: Class th10 neurons—with an ERF of about 30–90°—exhibit prolonged discharge to large contrast stimuli that are stationary in their ERF.

Static Object Recognizer

A model of Stationary Object Recognition in anurans was proposed by Lee (1994) based on these findings. In this paper we provide a schema (SorRec) that approximates this model providing the output through the th10 cells map.

Sensorimotor Schemas

Sensorimotor schemas integrate between sensory perception and motor action.

Prey Approach

Epstein (1977) introduced, and Arbib & House (1987) refined, the notion of prey attractant field. A prey sets up a symmetric attractant field whose strength decays gradually with distance from the prey. Arbib & House (1987) described the mask for prey objects as projecting very broadly in the lateral direction and somewhat less broadly in the forward direction This "prey-attractant-field" represents the location of the stimulus accurately as the center of mass of the representation. It also provides the system with neighbor positions available as targets were the accurate position impossible to reach, thus providing the system with a coarse representation of prey location.

PreyApproach projects this excitatory field onto the MHM (motor heading map) explained below. We hypothesize the projection of activity giving rise to coarse coding of prey location.

$$prey(i,j,t)@kp(i,j) \quad (17.2)$$

where i and j are indices for 2D arrays of neurons, t is time, k_p is a kernel, and @ denotes spatial convolution. In general, each kernel in the present model will be a truncated Gaussian of the general form

$$k(x,y,t) = \begin{cases} W \exp[-(x^2 + y^2)/2s^2] & \text{if } x^2 + y^2 \le R^2 \\ 0 & \text{otherwise} \end{cases} \quad (17.3)$$

where R is the receptive field size.

Static Object Avoid

Analogously, the model also includes a repellent vector field associated with each fence post. Its effect is more localized to its point of origin than is that of the prey field.

$$th10(i,j,t)@k_s(i,j,t) \quad (17.4)$$

We hypothesize the inhibitory pattern of connectivity to be also Gaussian shaped.

Bump Avoid

The BumpAvoid schema produces a reorientation that triggers the projection of an activity pattern (with quite large eccentricity) to the *MHM*. This field gives rise to excitation on the neighbor regions thus encoding the reorientation under bumping. It takes the form of

$$reorient(i,t) \quad (17.5)$$

Motor Heading Map

Cobas and Arbib (1992) propose that a motor heading map (*MHM*) determines the direction to jump: i.e., prey-catching and predator avoidance systems share a common map for the heading of the responding movements (coded in body coordinates), as distinct from a common tectal map for the direction of the stimulus. Note that the direction of prey and the direction of prey catching are the same, but the directions of a predator and the escape are different. Thus, in the latter case, the sensory map and the motor map must be distinguished. Projections to the *MHM* must differ depending on whether a visual stimulus is identified as prey, predator or obstacle.

In our model, the outputs of the previously defined schemas (*th10* and *prey*(*T5_2*) respectively) are projected to *MHM* through kernels.

In the current study the "neural field" generated in the *MHM* will be 1D (vs. 2D *prey* and *th10* maps) - we restrict here to the eccentricity component since the elevation component is not important for the problem at hand. That is, the height of each fence-post (for fences high enough that the frog could not jump over them) does not affect the detour behavior. The eccentricity component which actually represents the target heading angle in the *MHM* will be the key "feature" in determining the sidestep to detour around the barrier.

In our model, then, the total input I_{in} to *MHM* becomes

$$I_{in}(i,t) = \sum_j th10(i,j,t) * k_s(i,j,t) + \sum_j prey(i,j,t) * k_P(i,j) \quad (17.6)$$

Thus the total input to *MHM* when including reorientation due to bumping becomes

$$I_{in}(i,t) = \sum_j th10(i,j,t) * k_s(i,j,t) + \sum_j prey(i,j,t) * k_P(i,j) + reorient(i,t) \quad (17.7)$$

Winner-take-all dynamics over *MHM* assure the selection of the strongest target angle, upon which a transformation from retinotopic to motor coordinates takes place. This is the input (besides different gating signals from the sensory apparatus) to the different motor schemas. The motor schemas are then selected based upon competition and cooperation dynamics. Corbacho and Arbib (1995) present a winner-take-all model (Amari & Arbib, 1977; Didday, 1976) which uses a competition mechanism to obtain a single winner in the network.

Heading Transform

The Heading Map in Cobas and Arbib (1992) is differentially connected with the Orient schema depending on the region represented. The more lateral the stimulus is, the more strongly the Orient schema will be activated. The central portion of the heading map has a very light projection onto the oriented schema, and thus a prey falling into that region will only elicit a weak activation and consequently a very small turning movement or perhaps no turn at all.

We have implemented the transformation from spatially coded to population coded in a similar manner. The output of the sensory motor transformation codes for the amplitude of the target-heading angle. To perform the transformation we use a gradient of weights with a "V" shape. The highest value corresponding to the highest eccentricity.

$$angle(t) = \sum_i gradient(i) \wedge \Theta[I(i,t)] \tag{17.8}$$

where "^" is a pointwise vector multiplication, and implements a thresholding function to avoid producing an orienting response until the motor heading map "settles down" on a target position. Before the winner-take-all dynamics settle down on a "winner" target heading angle several clusters of activity may coexist in MHM corresponding to the representation of several barrier gaps in *MHM*. We use (Eq. 17.8) so that during the winner-take-all dynamics, the cluster of activity with higher amplitude will reach this threshold first as it is growing faster than any of the other clusters of activity. This enables the model to avoid computing a heading angle that could be a linear combination of several clusters of activity in *MHM*.

Motor Schemas

In the current model, motor schemas are implemented as functional units/black boxes schematizing the neural interactions underlying behavior. The intrinsic motor patterns or muscle activations are not simulated. When active they simply change the coordinates of the agent (and/or environmental parameters) appropriately.

We postulate that each component of the behavior (sidestep, orient, approach, snap, etc.) is governed by a specific motor schema. We then see detour behavior as an example of the coordination of motor schemas. Ingle (1980, 1983) has offered some clues as to the possible neural correlates of the various schemas. Apparently, thalamic and tectal visual mechanism can operate somewhat independently (Ingle, 1973). Monocular frogs without a contralateral optic tectum can quite accurately localize barriers, and while visual input to the pretectal region of the caudal thalamus mediates barrier avoidance behavior, caudal thalamic lesions produce an inability to sidestep stationary barriers set in the frog's path during pursuit of prey.

Among other motor schemas we provide the system with forward movement and lateral (sidestep) movement. The forward schema when active produces a movement in the direction of the midsagittal axis of the body with frontal direction. The lateral sidestep movement is a movement orthogonal to the sagittal midline. Backup movement is similar to forward but in the opposite direction.

Cobas and Arbib (1992) proposed a general mechanism of motor pattern selection through the interaction of motor schemas. *MHM* contains target location but motor schema selection is the result of competition of many maps. Each of the motor schemas has a threshold so that its action on the controlled musculature is only enabled when its internal level of activation reaches or surpasses that threshold.

Schema Dynamics

Schemas consist of schema behavioral mappings and schema activity variables. The full formalization is beyond the scope of this chapter; here simply mention that schemas

correspond formally to port automata with activity variables indicating the degree of confidence. The schema activity dynamics is described by the leaky integrator. The equation describing the dynamics of the schema activity variables is

$$\tau_i \frac{ds_i(t)}{dt} = -s_i(t) + \sum_j S_j(t) \cdot R_{i,j}(t) \tag{17.9}$$

where S is the result of a saturation by a sigmoid transfer function that guarantees that the activity variables remain within the interval [-1, 1],

$$S_i(t) = \Theta(s_i(t)) \tag{17.10}$$

R is the matrix of support. It indicates how the activation of a schema supports the activation of another schema. The leaky integrators time constants may be different for different schema activation variables since some schemas may have a faster dynamics e.g. Tactile must reset quickly and with it BumpAvoid.

Schema Assertion

Schema assertion takes place when the schema activity variable surpasses certain threshold hence indicating enough confidence on the application of that particular schema to the particular context. Once asserted the schema mapping output is produced, this pattern may in turn become the input for other schema mapping output. For many schemas once they are asserted they must be reset to avoid successive unrealistic activations. For instance once a motor schema has been asserted its activity variable is reset to 0.

Schema Interactions

There are some "reflex" dynamics corresponding to fast pathways e.g. Tactile activates Backup in one step (instantiation and activation). Also Tactile must reset quickly and with it BumpAvoid. Tactile momentarily inhibits Forward since otherwise Forward would be too active and lower down Backup activity variable to the point where Backup could not get activated. In general many schemas will be simultaneously active interacting with each other, for instance Sidestep and Forward schemas are simultaneously active when "detouring" after learning.

17.3 Model Implementation

The **Detour** model is composed of the **World** module—a 3D input stimulus library, **Prey** and **Frog** modules, as shown in figure 17.3. The static objects, in this case the **Barrier**, are interactively specified from the scripting language as opposed to the other two.

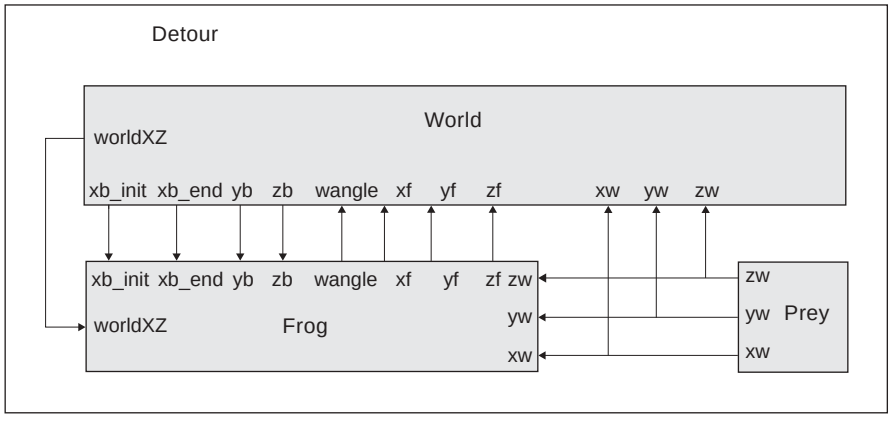

Figure 17.3
Schema Architecture showing the top-level world topology.

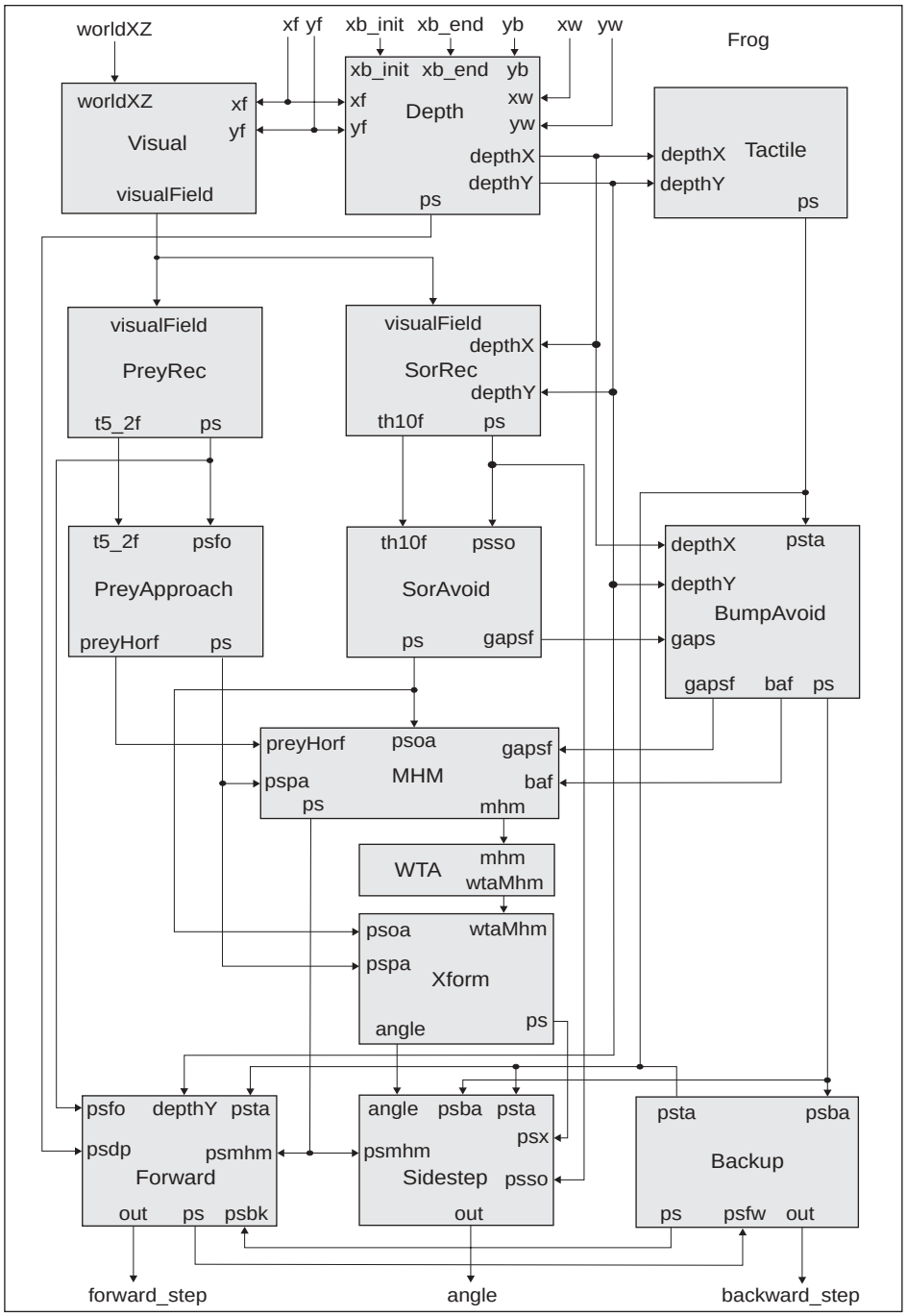

Figure 17.4
Schema Architecture showing the frog schemas topology.

World

We have provided for simple interactions with the *World* module. We simulate a simplified 3D environment by defining two different 2D projections or views: *worldXY* corresponding to the top down view only available to the user, and *worldXZ* corresponding to the view of an agent immersed in the environment. The *worldXZ* view is used as visual input to the frog.

The model takes advantage of the input layer components (see Appendix III for details) in generating external visual stimuli. A **NslInputFloat3** 3d input layer of **sizex1**x**sizex2** in the *x*-direction and **sizez1**x**sizez2** in the *z*-direction is instantiated by the *World* module as follows,

```
private NslInputFloat3 in(sizex1,sizex2,sizez1,sizez2);
```

Note that a the **NslInputFloat3** input layer actually involves two **NslInputFloat2** layers: an *xy*-matrix and a *xz*-matrix corresponding two the two different views in the 3d space. Input processing takes places as follows,

```
public void simRun(){
    ...
    in.run();
    worldXZ = in.get_xzview();
}
```

The *in* object is processed by applying a the *run* method to it. This generates a new *xy*-matrix together with a new *xz*-matrix assigned to *worldXZ* for further processing in the model.

Prey

The prey (worm in this case) is a static entity (although movement can be added to it, such as twiggling). In the current version the prey is described by its location and size.

Barrier

The obstacle (barrier in this case) is also a static entity composed of multiple posts separated by gaps, wide enough to let the frog see the prey behind it. The barrier gaps don't let the frog pass through it and are tall enough so the Frog won't jump over it. The size and gaps between barrier posts can be modified interactively as will be seen later on.

Frog

The agent (frog in this model) is the heart to the detour model. The frog model includes a number of perceptual, sensorimotor and motor schemas instantiated within the frog, as shown in figure 17.4, and described each in the following sections.

Perceptual Schemas

Perception for the frog in the model is based on **Visual** and **Tactile** sensors, where also **Depth** is computed. In particular, the frog perceives the prey, **PreyRec** (Prey Recognizer), and the barrier, **SoRec** (Static Object Recognizer).

Visual

The visual input to the frog correspond to 2D image projections of the virtual 3D world reflected on the eyes (or camera) of the agent. The model computes a *visualField* corresponding to the section of *worldXZ* that the frog can see at each time step. As the frog moves - frog coordinates *xf, yf* change - the *visualField* needs to be recomputed.

The **simRun** methods computes the new *visualField* from the complete *worldXZ* view depending on the size of its receptor field *recsize*

```
public void simRun()
{
    int recsize = visualField.getRows();
    int isize = worldXZ.getRows();
    int jsize = worldXZ.getCols();

    visualField = worldXZ.getSector(isize-recsize,isize-1,
       jsize/2-recsize/2,jsize/2+recsize/2-1);
}
```

The *getSector* method obtains the portion of the *worldXZ* view perceived by *visualField*.

Depth

The **Depth** module computes the distance in both *x* (*depthX*) and *y* (*depthY*) to the barrier or prey depending on the Frog's current position in the world. This information is passed to the static object recognition and bump avoidance modules as well as the Forward module in avoiding hitting the barrier. The module output *ps* is a confidence level describing when *depthY* is greater than *safeDistance*, where *safeDistance* corresponds to the minimum distance the Frog should be to avoid hitting the barrier.

The **simRun** method computes the dynamic location of the frog, and then calculates through simple subtractions the depth of the barrier and finally the *ps* confidence level that the frog is not too close to the barrier as output to other modules.

```
public void simRun()
{

...

if (depthY > safeDistance || depthX > safeDistance)
       ps = 2.0;              // Go up fast
   else
       ps = 0.0;
}
```

Tactile

The **Tactile** module simulates the frog hitting the barrier from its current position to the barrier computed by **Depth**.

The **simRun** method computes the output confidence level *ps* depending on whether the frog is close enough, both *x* and *y*, to the barrier.

```
public void simRun()
{
   if (depthY > 0 && depthY <= safeDistance &&
    depthX <= safeDistance)
       ps = 2;          // Go up fast
   else
       ps = 0.0;
   ps = nslSigma(ps, -1.0, 1.0, -1.0, 1.0);
}
```

Prey Recognizer

The *Prey Recognizer* (**PreyRec**) module recognizes and localizes prey stimuli within the visual field of the frog. The *Prey* is defined as a set of features. In this particular implementation we have simplified this perceptual schema a great deal (see Corbacho and Arbib 1995 for a more detailed implementation). The module receives *visualField* input from the frog visual module and generates both an output confidence level and simulates the behavior of the *t5_2* neural cells.

The **simRun** method computes the prey recognizer output in terms of filtering the *visualField* for a prey stimulus.

```
public void simRun()
{
    t5_2 = DetourLib.filter(visualField,2);
    if (nslSum(t5_2) >= 1)
        ps=0.9;
    else
        ps=0;
    ps = nslSigma(ps,-1.0,1.0,-1.0,1.0);
    if (ps > th)
        t5_2f = nslRamp(t5_2);
    else
        t5_2f = 0;
}
```

In particular

$$t5\_2 = DetourLib.filter(visualField, 2); \tag{17.11}$$

defines a prey in terms of a feature "2" corresponding to preys. The function filters out all elements in the matrix that do not have a corresponding value, in this case "2". This filtering function can be made more realistic including color, spatial frequency, complex shape filters, etc.

The output *t5_2* is still a 2D map representing the *retinotopic* position of the prey (vs. allocentric prey coordinates).

Since this is a "seed" perceptual schema it must also provide "seed support" for its schema activation variable *ps*.

$$ps = \begin{cases} 0.9 & \text{if nslSum}(t5\_2) >= 1 \\ 0 & \text{otherwise} \end{cases} \tag{17.12}$$

Then, once the schema is asserted, $ps > th$,

$$t5\_2f = \text{nslRamp}(t5\_2); \tag{17.13}$$

corresponding to the activation of the output port.

Static Object Recognizer

The *Static Object Recognizer* (**SoRec**) module recognizes and localizes static objects within the visual field of the frog. The *Static Object* is defined as a set of features. The module receives *visualField* input from the frog visual module and generates both an output confidence level and simulates the behavior of the *th10* neural cells.

The **simRun** method computes the prey recognizer output in terms of filtering the *visualField* for a barrier.

```
public void simRun()
{
   th10 = DetourLib.filter(visualField, 1);
   if (nslSum(th10) >= 1)
      ps=0.9;
   else
      ps=0;
   ps = nslSigma(ps,-1.0,1.0,-1.0,1.0);
   if (ps > th)
      th10f = nslRamp(th10);
   else
      th10f = 0;
}
```

Similarly to the *Prey Recognizer*, the stationary object recognition *filter stationary objects*,

*th*10 = *DetourLib.filter*(*visualField*, 1); (17.14)

corresponding to feature "1" defining stationary objects.

Then, once the schema is asserted, *ps* > *th*,

*th*10*f* = nslRamp(*th*10); (17.15)

corresponding to the activation of the output port.

Sensorimotor Schemas

The frog model incorporates a number of sensorimotor schemas: **PreyApproach**, **SoAvoid**, **BumpAvoid**, Motor Heading Map (**MHM**) and Heading Transform (**Xform**).

Prey Approach

The **PreyApproach** module integrates the horizontal projection of *t5_2* cells generating a 1D representation (parcellation), since it is more efficient to make the 1D projection before convolving with the gaussian kernel. *preyHor* corresponds to the eccentricity component of the *prey attractant field* (horizontal component).

The **initSys** method reinitializes variables to 0, sets the confidence level input weight *rs* to 1, and initializes the excitatory gaussian kernel *t*5_2_*erf*,

```
public void initSys()
{
   preyHor = 0;
   preyHorf = 0;
   rsfo = 1.0; // Prey & Prey Approach.
   ps = 0;
   DetourLib.gauss2D(t5_2_erf,t5_2_erf_sig);
   t5_2_rf = t5_2_erf_wgt * t5_2_erf;
}
```

The **simRun** method computes the module activity,

```
public void simRun()
{
   preyF = t5_2_rf * t5_2f;
   preyHor = nslReduceRow(preyF); // Parcellation:
      horizontal comp
   ps = rsfo*psfo;
   ps = nslSigma(ps,-1.0,1.0,-1.0,1.0);
   if (ps > th) {
      float mx = nslMax(prey_hor);
      if (mx != 0.0)
         prey_hor_f = prey_hor/mx; // Normalize.
   }
   else
      prey_hor_f = 0;
}
```

Once the schema is asserted, *PreyHor* contains the field (normalized by the maximum value) to be projected to the other modules (e.g. *MHM*).

Static Object Avoid

The *SorAvoid* schema is implemented in a similar manner. IT integrates *th10Hor* as the 1D horizontal component corresponding to a parcellated representation (C&A95). *Gaps* corresponds to the inhibitory obstacle repellent field. Once the schema is asserted, *gapsf*, which is normalized by the maximum value, is projected to the other schemas (e.g. *MHM*).

The **initSys** method reinitializes variables to 0, sets the confidence level input weight *rs* to 1, and initializes the inhibitory gaussian kernel *tm_irf* and the final resulting kernel *tm_rf*

```
public void initSys()
{
   rsso = 1.0; // Obstacle & Obstacle Avoid
   ps = 0;
   DetourLib.gauss1D(tm_irf,tm_irf_sig);
   tm_rf = - tm_irf_wgt * tm_irf;
}
```

The **simRun** method computes the schema activity,

```
public void simRun()
{
   th10Hor = nslReduceRow(th10f); // Parcellation (from 2D to 1D)
   gaps = tm_rf * th10Hor;          // Convolve with kernel
   ps = rsso*psso;
   ps = nslSigma(ps,-1.0,1.0,-1.0,1.0);
   if (ps > th) {
      float mx = nslMax(gaps*-1);
      if (mx != 0.0)
      gapsf = gaps/mx;
   }
   else
      gapsf = 0;
}
```

Bump Avoid

The **BumpAvoid** schema contains two components: field projection *baf* for reorientation (avoid keeping bumping on the same point) and, tuning of the *SorAvoid* module.

The **simRun** method computes the activity as follows

```
public void simRun()
{
    if (depthY <= safeDistance && depthX <= safeDistance)
       //Bumping ps = 0;
    else
       ps = -1.0;      // Go down fast: -2.0
    if (tune_tm < 1.5)        // saturate tune_tm.
          tune_tm = tune_tm + tune_tm_base;
    tune_tm_layer = tune_tm;
    tune_tm_layer = tune_tm_layer ^ nslStep(-gaps);
    gaps = gaps - tune_tm_layer;
    gapsf = gaps;
    ps = ps + rsta*psta;
    ps = nslSigma(ps,-1.0,1.0,-1.0,1.0);
    if (ps > th){
       field_center = field_center + 2;
       baf[field_center.getValue()] = field_A;
    }
    else
       baf = 0;
}
```

The function modulates the kernel. Every time it bumps it increases *tune_tm* until it reaches a saturation point. It tunes the bump avoid field by increasing eccentricity, modulating only already active neurons. Every bump it increases *tune_tm* until it reaches a saturation point.

Motor Heading Map

The *Motor Heading Map (MHM)* schema then integrates the different fields *preyHorf*, *gapsf* and, *baf*. Another input to *MHM, in*, contains further modulating fields learned by the system. In particular, it will contain fields generated by newly constructed schemas (e.g. detour schema, at the moment the only one in the model).

The **simRun** method computes the schema activity,

```
public void simRun()
{
    if (d_mhm > d_norm && gapsf ! = 0)
    {
       baf[field_center.getValue()] = field_A;
       in = baf; // New Field “inserted”
    }
    else
       in = 0; // reset input (cf. antidromic
    mhm_hat = mhm;

    // Predictive MHM.
    mhm = gapsf + preyHorF + baf + in;
    // Fields over MHM.
    d_mhm = 0.03 * DetourLib.dist(mhm, mhm_hat);
    ps = ps + rspa*pspa + rsoa*psoa;
    ps = nslSigma(ps,-1.0,1.0,-1.0,1.0);
}
```

The above code computes the dynamics of Motor Heading Map (MHM), integrating several fields, while new fields can also be added while learning. The "if" section computes learning dynamics. It detects an incoherence and hence a trigger for a new schema.

Heading Transform

The *winner take all* selects a single target where *maxim* returns the vector normalized (subtraction) by its maximum, where only the maximum is above threshold (by 0.01).

```
public void simRun()
{
    wta_mhm = DetourLib.maxim(mhm);
}
```

Heading Transform

The *Xform* schema transforms from retinotopic (vector) to population code (scalar) the representation of the target.

The **simRun** method computes the schema activity,

```
public void simRun()
{
    int i = nslAvgMaxValue(wta_mhm);
    angle = 0;
    if (i != 0)
        angle = i - wta_mhm.getRows()/2;
    ps = ps + rspa*pspa + rsoa*psoa;
    ps = nslSigma(ps,-1.0,1.0,-1.0,1.0);
}
```

The method computes the transformation to population coding corresponding to

angle = nslSum(*gradient* ^ *wtaMhm*); (17.16)

coding the heading angle as a scalar (cf. population coding).

Motor Schemas

We have included three motor schemas as explained in the Model Description section, *forward*, *sidestep* and *backup.*

Forward

The **Forward** motor schema receives confidence contributions from other schemas as well as depth information to avoid hitting the barrier. *step* is a scalar coding the amplitude of forward movement. The **simRun** method computes the motor schema activity,

```
public void simRun()
{
    ps = ps + rsfo*psfo + rsta*psta + rsdp*psdp +
    rsmhm*psmhm + rsbk*psbk;
    ps = nslSigma(ps,-1.0,1.0,-1.0,1.0);
    if (ps > th) {
        ps = -1.0; // Reset
        out = step;
    }
    else
        out = 0;
}
```

Sidestep

The **Sidestep** motor schema receives confidence contributions from other schemas. *angle* is a scalar coding the amplitude of the sidesteps. The **simRun** method computes the motor schema activity,

```
public void simRun()
{
    ps = ps + rsso*psso + rsta*psta + rsba*psba + rsmhm*psmhm +
    rsx*psx;
    ps = nslSigma(ps,-1.0,1.0,-1.0,1.0);
    out = angle;
    if (ps > th) {
        ps = 0; // -1.0; // Reset
    }
}
```

Backup

The **Backup** motor schema receives confidence contributions from other schemas. *step* is a scalar coding the amplitude of the backup movement (we omit the sign). The **simRun** method computes the motor schema activity,

```
public void simRun()
{
    ps = ps + rsta*psta + rsba*psba + rsfw*psfw;
    ps = nslSigma(ps,-1.0,1.0,-1.0,1.0);
    if (ps > th) {
        ps = -1.0; // Reset
        out = step;
    }
    else
        out = 0;
}
```

Learning Dynamics

Schema dynamics previously presented are a simplification of Relaxation Labeling (Hummel & Zucker, 1983). Additionally, we provide a broad description of some of the learning mechanisms involved in both constructing a new schema and in tuning a existing schema. For the overall Schema-Based Learning (*SBL*) framework please refer to (Corbacho, 1998).

Schema Learning

We explain how a "new" field of activity over *mhm* is able to reproduce a previously successful pattern of interaction. Concretely the field of activity projected over *mhm* that caused the frog to reach the edge of the barrier.

Learning of a new schema is triggered when incoherence is detected. In this case the unexpected interaction when the frog gets to the edge of the barrier is reflected internally as incoherence in *mhm*. For every field projecting to *mhm* the predictive response is calculated by simply storing the previous value corresponding to the result of activating that field.

$$mhm\_hat(t+1) = mhm(t) \tag{17.17}$$

The incoherence is measured as the distance between the current and the expected result,

$$d\_mhm(t+1) = Detour\_lib.dist(mhm(t+1), mhm\_hat(t+1)) \quad (17.18)$$

when the incoherence is larger than a threshold it indicates "unexpected". In this case the "culprit" is the field of activity *baf* (triggered by the *BumpAvoid* schema in the first place), and hence this field is internally stored so that it can be "played back" in future interactions with the barrier.

$$\text{if } (d\_mhm > d\_norm) \quad (17.19)$$
$$in = baf$$

On second presentation of the barrier *in* (reflecting a pattern of activity similar to *baf*) projects a field of activity over *mhm* which in turn gives rise to a large value in *angle* hence activating the *Sidestep* motor schema and detouring around the barrier.

Schema Tuning

In terms of schema tuning, the kernel for *SorAvoid* is tuned every time the *BumpAvoid* schema is asserted.

$$\text{if } (tune\_tm < 1.5) \quad (17.20)$$
$$tune\_tm = tune\_tm + tune\_tm\_base$$

Additionally in *tuningField*

$$tune\_tm\_layer = tune\_tm \wedge nslStep(-gapsf) \quad (17.21)$$
$$gapsf = gapsf - tune\_tm\_layer \quad (17.22)$$

updates the field obstacle avoidance field (*gapsf*) by subtracting the modulation component.

17.4 Simulation and Results[1]

Different experiments were carried varying the barrier size (10cm and 20cm) as well as applying learning to the 20cm barrier experiment. The main simulation files are described in table 17.2:

File	Description
detour.nsl	contains all the model parameters
detour_sti.nsl	contains all the model stimulus specifications
detour_fields.nsl	displays different fields
detour_env.nsl	displays a top down and visual view of the environment

Table 17.2
NSLS script files needed to run the different simulations.

To execute the model do:

```
nsl source detour
nsl run
```

The stimuli specifications are done using the NSL input library described in Appendix III. The *detour_sti.nsl* file includes parameters for the input layer as follows,

```
nsl set detour.world.in.dx 1
nsl set detour.world.in.dy 1
nsl set detour.world.in.dz 1
nsl set detour.world.in.xz 0
nsl set detour.world.in.yz 0
nsl set detour.world.in.zz 0
```

The worm specification is given by an input stimulus defined from the NSL input library as follows,

```
nsl create BlockStim prey -layer detour.world.in -val 2 \
    -xc $xw -yc $yw -zc $zw -dx 1 -dy 1 -dz 1 -spec_type center
```

Note that all variables preceded by the "$" symbol corresponds to variable values from Tcl (see the NSLS scripting language description in chapter 7). The values for these variables are chosen according to the particular experiment selected through variables *learning* and *trial* as will be described next.

The frog specification is given similarly by an input stimulus defined from the NSL input library as follows,

```
nsl create BlockStim frog -layer detour.world.in -val 1 \
    -xc $xf -yc $yf -zc $zf -dx 3 -dy 3 -dz 3 -spec_type center
```

The barrier (or fence) specification is a little more involved given this time by a set of input stimuli defined from the NSL input library as follows,

```
for {set xb $xb_init} {$xb <= $xb_end} {incr xb $gap} {
    nsl create BlockStim fence -layer detour.world.in -val 1 \
        -x0 $xb -y0 $yb -z0 $zb -dx 1 -dy 1 -dz 100 -spec_type
           corner
}
```

Note that in the above specification the notation and expressions correspond to the NSLS scripting language extended from Tcl as described in chapter 7.

Experiment I

For experiment I (barrier 10 cm wide) set the following variable in *detour_sti.nsl*

```
set learning 0
set trial 10
```

After executing "nsl run" the system displays on one of the windows the different module fields as shown in figure 17.5.

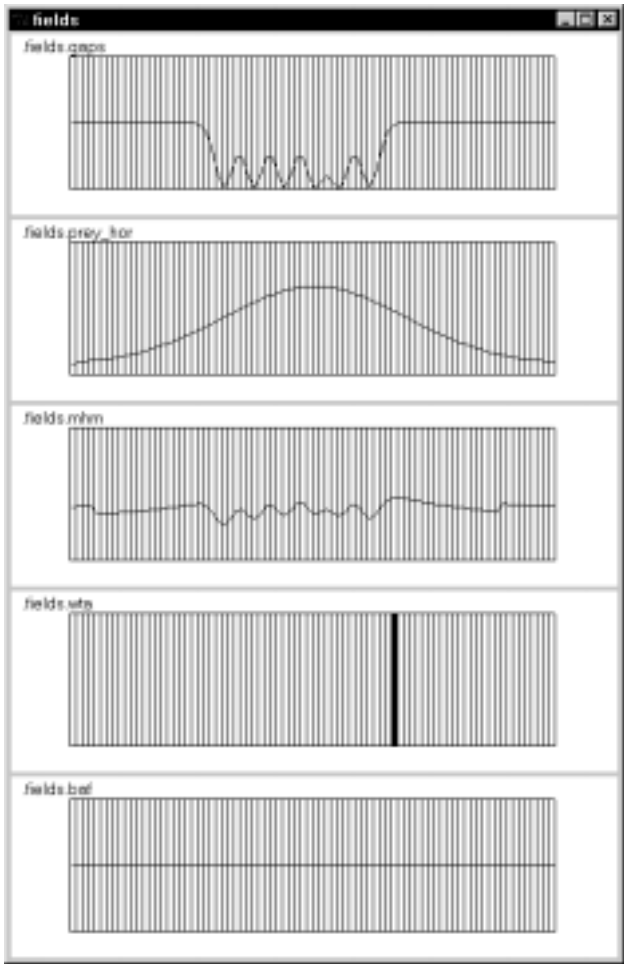

Figure 17.5
Different activity fields for the 10cm barrier experiment due to *visual_field* processing in the frog with the exception of the bottom one processed after the *tactile* field. The top display (*gaps*) shows the repulsive field generated from the barrier (note that it is negative). The next display down (*prey_hor*) represents the attraction field generated from the prey (note that it is positive). The next display down (*mhm*) represents the combined *gaps* and *prey_hor* fields. The next display down (*wta*) represents the winner-take-all element from the above *mhm* field. This winning element results in the heading or frog's orientation when moving forwards. The last display (*baf*) is currently empty and represents activity due to bumping against the barrier.

The most important factor in the frog movement direction results from the *wta* field, resulting itself from the combination of the prey attraction and barrier repulsion fields. In this experiment the direction of movement is towards the side of the barrier, heading towards the right since the frog was positioned just a bit to the right from the axis joining the center of the prey and barrier. The resulting path motion is shown in figure 17.6.

Figure 17.6
Rana Computatrix interacting with the 10 cm wide barrier. Note how the frog heads itself towards the side of the barrier.

Experiment II

For experiment II (barrier 20 cm wide) set the following variable in *detour_sti.nsl*

```
set learning 0
set trial 20
```

After executing "nsl run" the system displays on one of the windows the different module fields as shown in figure 17.7.

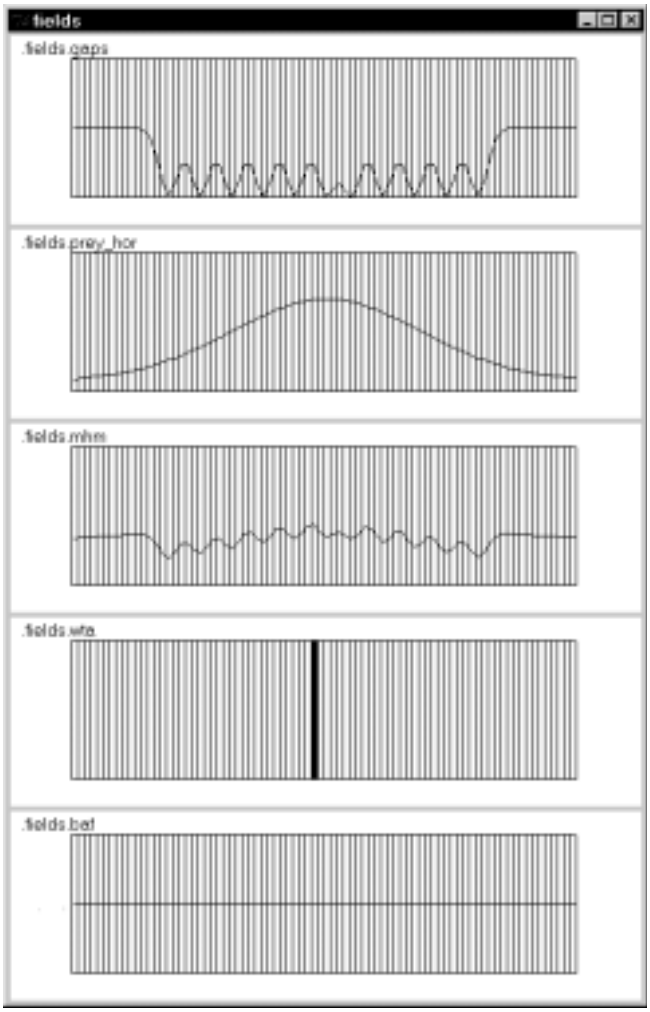

Figure 17.7

Different activity fields for the 20cm barrier experiment before bumping due to *visual_field* processing in the frog with the exception of the bottom one processed after the *tactile* field. The top display (*gaps*) shows the repulsive field generated from the barrier (note that it is negative). The next display down (*prey_hor*) represents the attraction field generated from the prey (note that it is positive). The next display down (*mhm*) represents the combined *gaps* and *prey_hor* fields. The next display down (*wta*) represents the winner-take-all element from the above *mhm* field. This winning element results in the heading or frog's orientation when moving forwards. The last display (*baf*) is currently empty and represents activity due to bumping against the barrier.

Again, the most important factor in the frog movement direction results from the *wta* field, resulting itself from the combination of the prey attraction and barrier repulsion fields. In this experiment the direction of movement before bumping into the barrier is towards the middle of the barrier. Once the frog hits the barrier a bumping (baf) field is generated. The purpose of this field is to redirect the movement towards a different heading. Before that occurs the frog will backup. The resulting field after bumping is shown in figure 17.8.

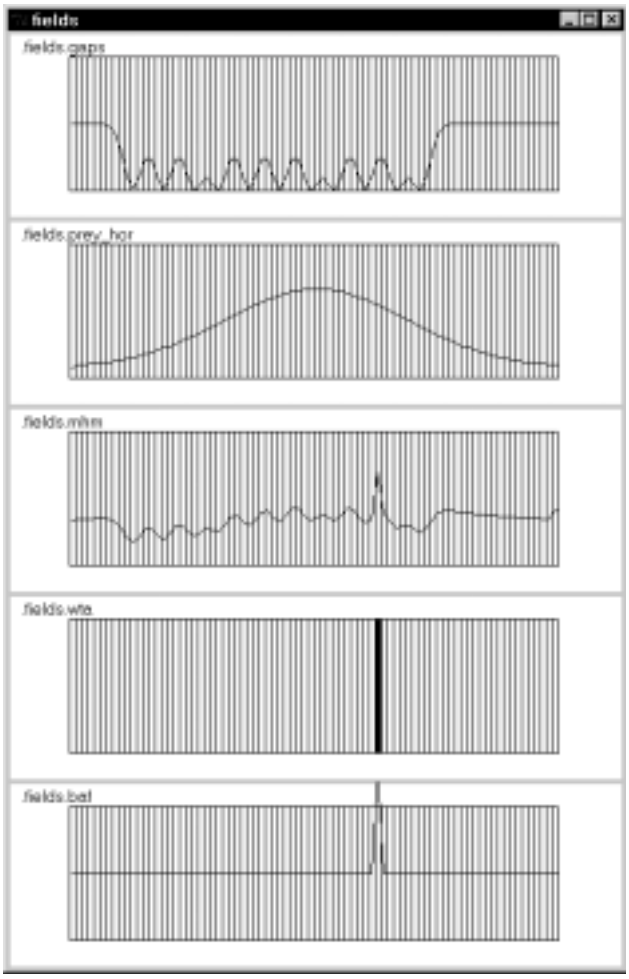

Figure 17.8
Different activity fields for the 20cm barrier experiment after bumping due to *visual_field* processing in the frog with the exception of the bottom one processed after the *tactile* field. The top display (*gaps*) shows the repulsive field generated from the barrier (note that it is negative). The next display down (*prey_hor*) represents the attraction field generated from the prey (note that it is positive). The next display down (*mhm*) represents the combined *gaps* and *prey_hor* fields. The next display down (*wta*) represents the winner-take-all element from the above *mhm* field. This winning element results in the heading or frog's orientation when moving forwards. The last display (*baf*) is represents activity due to bumping against the barrier.

The resulting path motion after hitting the barrier several times is shown in figure 17.9.

Figure 17.9
Rana Computatrix interacting with the 20 cm barrier before learning. We have added numbers corresponding to the frog's position in time. In this experiment the frog hits the barrier three times before perceiving the side of the barrier.

Experiment III

For experiment III (barrier 20 cm wide with learning) set the following variable in *detour_sti.nsl*

```
set learning 1
set trial 20
```

We change the threshold of *d_norm* to simulate that after one interaction with the 20cm barrier the frog would have learned and from then on it would detour when presented with the 20 cm barrier. The resulting behavior is shown in figure 17.10.

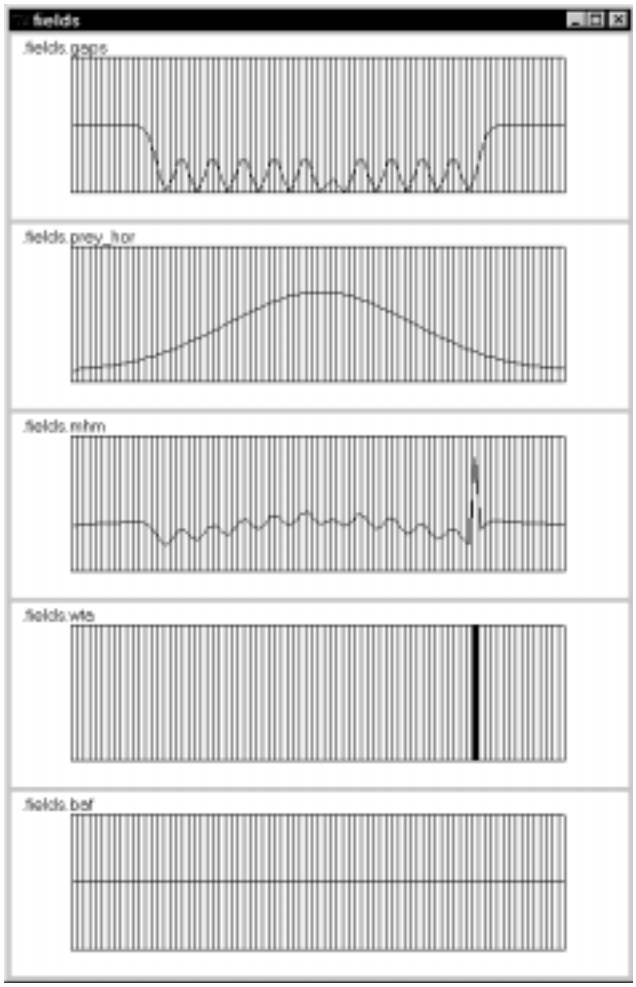

Figure 17.10
Different activity fields for the 20cm barrier experiment after learning due to *visual_field* processing in the frog with the exception of the bottom one processed after the *tactile* field. The top display (*gaps*) shows the repulsive field generated from the barrier (note that it is negative). The next display down (*prey_hor*) represents the attraction field generated from the prey (note that it is positive). The next display down (*mhm*) represents the combined *gaps* and *prey_hor* fields. The next display down (*wta*) represents the winner-take-all element from the above *mhm* field. This winning element results in the heading or frog's orientation when moving forwards. The last display (*baf*) is currently empty and represents activity due to bumping against the barrier.

Note that although no bumping occurs, the *mhm* field involves a similar integration where heading is explicitly generated, in this case by learning. The resulting behavior is shown in figure 17.11.

Figure 17.11
Rana Computatrix interacting with the 20 cm wide barrier after learning.

17.5 Summary

The model explains basic facts about detour behavior. If the retinotopic representation of the edge of the barrier in *SorRec* falls within the prey-attractant-field spread, then the summation of activity from the prey-attractant-field and from the *SOR*-repellent map on *MHM* at the retinotopic position just beyond the barrier's edge is stronger then the summation at the "center" of the barrier where the prey is located. Hence, the winner-take-all dynamics will select the cluster of activity corresponding to the retinotopic position of the edge of the barrier, thus predicting that frogs would detour around narrow barriers. On the other hand, for wide barriers the prey-attractant-field extent falls within a much wider barrier field. Hence, at the *MHM* retinotopic position corresponding to the barrier's edge there will be no input activity from the prey map. On the other hand, there will be a great projection of activity on *MHM* at the retinotopic position of the prey; and this in turn will trigger approach to a point within the barrier map, so long as the peak of prey attraction exceeds the trough of barrier inhibition. Thus, the model predicts that the naive frog would approach wide barriers rather than detour around them.

Notes

1. Preparation of this paper was supported in part by award number IBN-9411503 for Collaborative Research (M.A. Arbib and A. Weerasuriya, co-Principal Investigators) from the National Science Foundation.
2. A. Weitzenfeld developed the NSL3.0 version and extended the original NSL2.1 model implementation written by F. Corbacho as well as contributed Section 17.3 and part of 17.4 to this chapter.
3. The Detour model was implemented and tested under NSLC.