15 Crowley-Arbib Saccade Model

M. Crowley, E. Oztop, and S. Mármol

15.1 Introduction

The visual system provides the primary sensory input to human brains. Nearly every activity we undertake first requires some information to be obtained from the visual system, whether it is identifying a face, or locating an object for manipulation. To obtain this information, the visual system must first move the eyes so that the region of interest falls upon the fovea, the most sensitive part of the eye. Additionally, moving objects must be tracked once they are foveated. These two aspects of "target" acquisition illustrate the two types of movements the oculomotor system are capable of producing. The former is called as saccades, which are quick eye movements to bring an object into the fovea. The latter is referred as smooth pursuit eye movements which are for tracking moving objects.

Crowley-Arbib model is a saccade model with an emphasis on the functional role of the Basal Ganglia (BG) in production of saccadic eye movements. It is based on the hypothesis that the BG has two primary roles the first being the inhibition of a planned voluntary saccade until a GO signal is established by the prefrontal cortex and the second being the provision a remapping signal to parietal and prefrontal cortex, through thalamic projections, that is a learned estimate of the future sensory state based upon the execution of the planned motor command.

The hypothesis that one of the basal ganglia roles is to inhibit a planned motor command prior to its execution was also used by Dominey and Arbib (1992) but is different than the action selection proposed by Dominey, Arbib, and Joseph and by Berns and Sejnowski (1995). However, both ideas require the involvement in the BG in motor preparatory activity. The issue is whether this preparatory activity assists cortical areas in selecting an action, or whether it instead is involved in "freezing" the execution of the motor command until the planning cortical areas, e.g., prefrontal cortex, execute a go signal. We suggest that nearly all motor planning occurs in cortical areas and that these areas use subcortical regions to provide specific information to aid in the motor planning.

15.2 Model Description

This model includes a number of cortical and subcortical areas known to be involved in saccadic eye movements: Lateral Intraparietal Cortex (LIP), Thalamus (Thal), Prefrontal Cortex (PFC), Superior Colliculus (SC), Frontal Eye Field (FEF), Basal Ganglia (BG), Brainstem (BS). For each of these areas we will arrange one or more modules depending on their individual functionality (i.e., for each of the two main roles of the BG described above we are going to create two different modules: Lateral Basal Ganglia (Lat) and Medial Basal Ganglia (Med)). In addition each module could be an assemblage of more submodules, creating with this a hierarchy where the leaves implement the details of the neurons involved. figure 1 shows the top level modules and how they are interconnected, as implemented by means of the Schematic Editor. We will discuss more about each module in the next paragraphs.

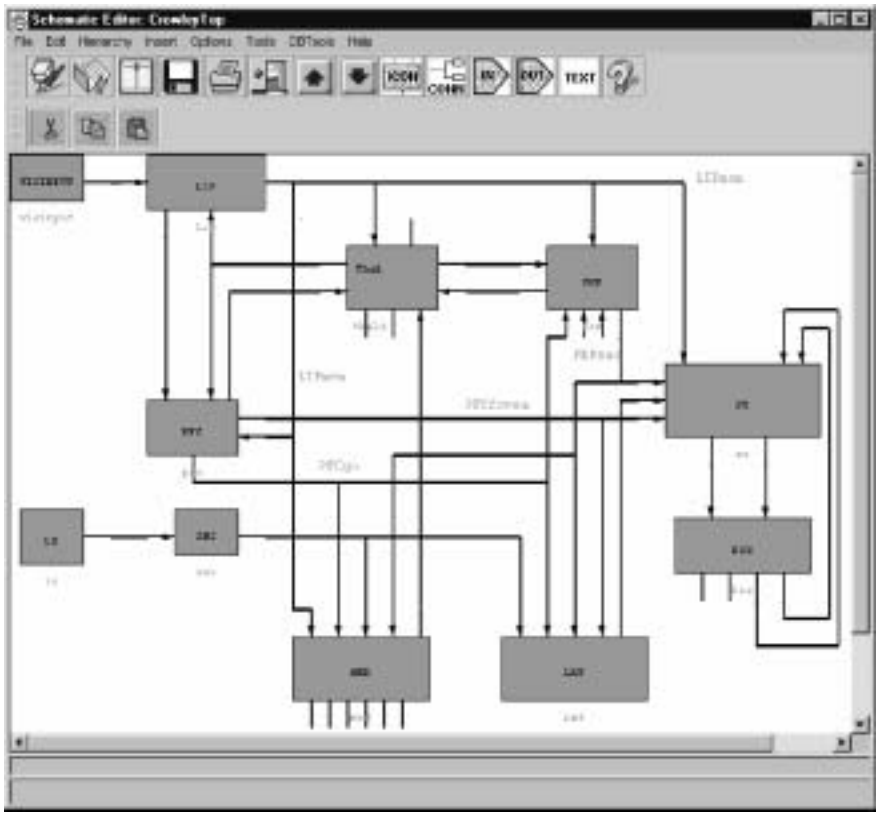

Figure 15.1
Top Level of the Crowley and Arbib Model

Lateral Intraparietal Cortex (LIP)

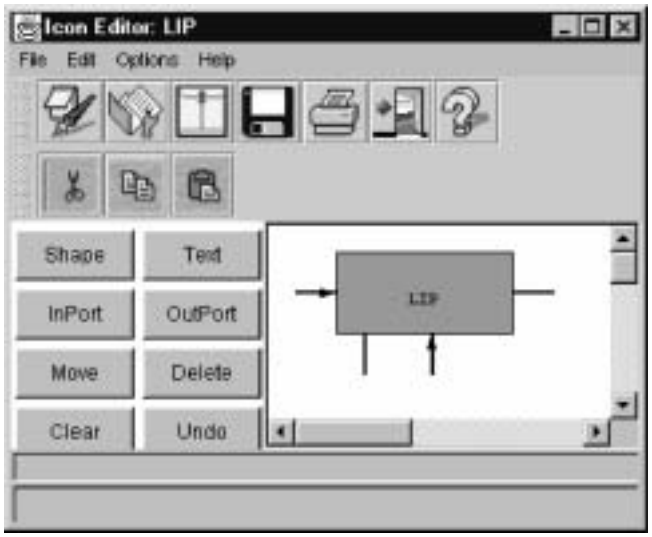

Figure 15.2
Lateral Inter Parietal Cortex I

LIP provides the retinotopic location of saccade targets through excitatory connections form its memory related neurons to SC, FEF and BG. It also exhibits the result of the remapping of saccade targets.

This module is modeled as composed of two types of cells due to the data from Gnadt and Andersen (1988). They found cells in area LIP that responded to a visual cue that did not last through the delay period in a delay saccade task. They also found sustained response cells whose firing was turned off by the eye movement. We will model

the first class of neurons as visually responsive neurons (LIPvis) and the second class as memory-responsive neurons (LIPmem).

Visual Response Cells (LIPvis) only respond to visual stimuli. These neurons are modeled as receiving visual input from primary visual centers. In order to obtain saccade latencies that match experimental data we have included a chain of primary visual cortex regions that simply pass the visual signal to the next layer with a slight delay.

Memory Response Cells (LIPmem) fire continuously during the delay portion of a delay saccade task. These cells would fire even if the stimulus never entered their receptive field when second saccade was arranged so that it matched the cell's movement or receptive field. We propose that a memory loop is established between these cells in LIP and mediodorsal thalamus. The connection strength between LIP and thalamus were chosen so that the memory of saccade targets would remain without the target. Also the strength had to be not too strong to disable BG's power to eliminate memory traces.

Thalamus (Thal)

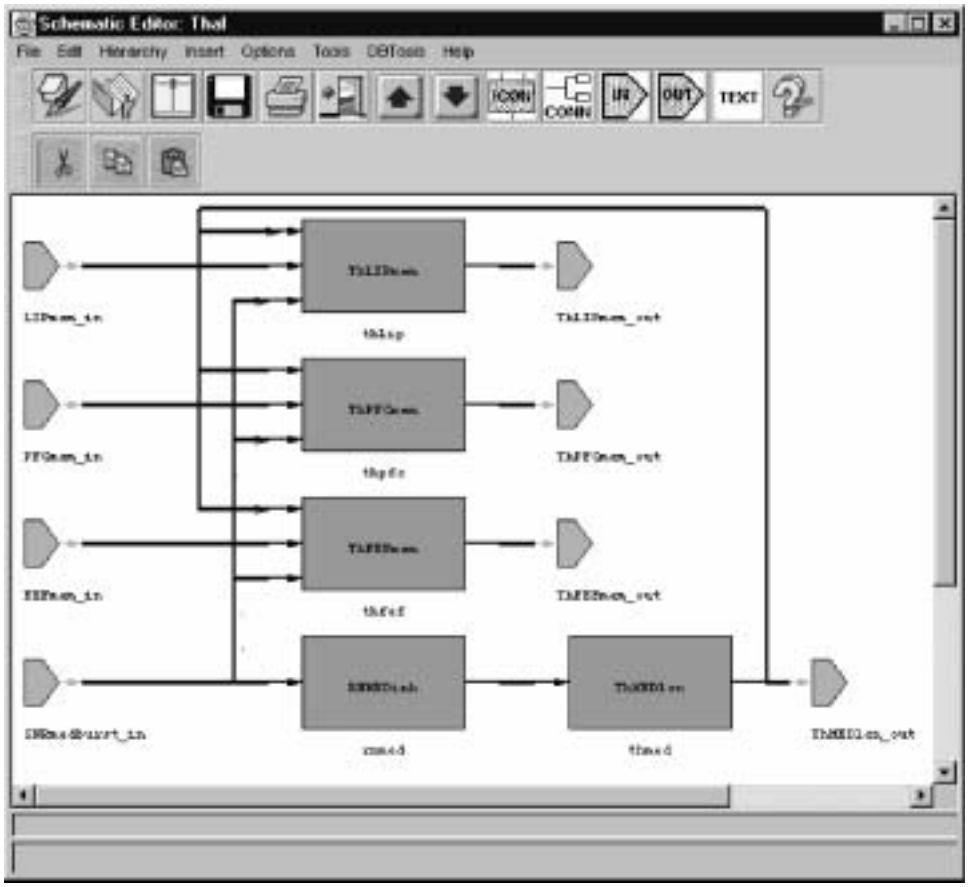

Figure 15.3
Thalamus

The thalamus relays sensory input to the primary sensory areas of the cerebral cortex, as well as information about motor behavior to the motor areas of the cortex. Based on the experimental data discussed below, we will consider only the mediodorsal nucleus and ventral anterior thalamic areas in our model. Additionally, these two areas will be implemented as a single layer within the model as the afferent and efferent connections are very similar between these two areas. In the model three types of cells in the thalamus and reticular nucleus are used as described next.

Thalamic Relay Cells (THrelay) have reciprocal connections with specific cortical areas. These are further divided into different sets for LIP, FEF, and PFC. These

reciprocal loops maintain neuronal activity during delay periods of memory tasks essentially forming a memory loop.

Thalamic Local Circuit Cells (THlcn) fire continuously providing inhibition to the relay cells. But they are controlled by the inhibitory actions of the thalamic reticular nucleus (RNinh) neurons. When inhibition upon these LCN neurons drops below a threshold, their increased inhibition puts the thalamic relay neurons into a bursting mode until the corresponding cortical cells begin firing and inhibitory activity of the SNr returns to its normal levels.

Reticular Inhibitory Cells (RNinh) are tonically firing neurons receiving inhibition from SNr. They provide inhibition of the thalamic local circuit neurons.

Prefrontal Cortex (PFC)

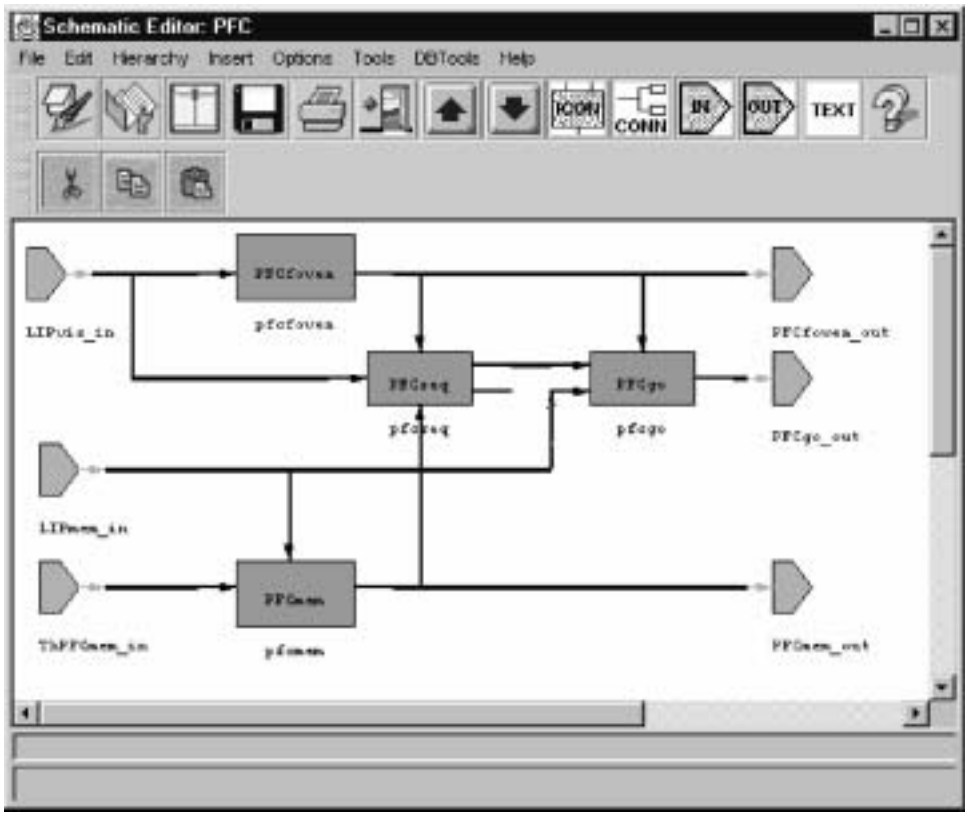

Figure 15.4
Prefrontal Cortex

It has been fairly well agreed that PFC is crucial for the process of working memory (Boussaoud and Wise 1993; Goldman-Rakic 1987; Kojima and Goldman-Rakic 1984; Sawaguchi and Goldman-Rakic 1994; Sawaguchi and Goldman-Rakic 1991). Lesions of this area render monkeys unable to perform spatial memory tasks even when the delay period is only a few seconds. We introduce the following layers in our model.

Visual Memory Cells (PFCmem) simulate the spatial working memory cells found in prefrontal cortex. These cells maintain a memory loop with the thalamus, as well FEF and LIP. We use the go signal (PFCgo) to inhibit the activation letting the remapped target information to be created. This mimics a memory state change from a current state to a future state. The connection strength was chosen to be strong enough to form target memory but not strong enough to disable BG from washing out the memory traces.

Go Cells (PFCgo) pass the trigger signal to FEF and the BG. This trigger will increase the receiving layers' activation for the selected target to cause the activation through to the superior colliculus to effect the saccade selected by prefrontal cortex. They

receive visual memory information from LIP (LIPmem), the next saccade target location (PFCseq) and a cortical fixation signal (PFCfixation).

Fixation Cells (PFCfixation) provide a fixation signal as FEFfovea cells. The difference is that these cells have a large time constant. Thus it takes longer time to activate and deactivate the PFCfixation cells.This allows for the maintenance of fixation without a foveal signal.

Sequence Memory Cells (PFCseq) maintain a representation of the order in which the saccades are to be performed. The target locations are channeled via PFCmem.

Saccade Selector Cells (PFCsel) select the next saccade to be performed. We use refractory period to control when saccades can occur. This is a decaying value that must be overridden by the level of excitation in the visual memory calls (PFCmem). These cells select the target memory in PFCmem that has the highest activation and project this signal to the go cells (PFCgo) to assist in the activation of a saccade.

Superior Colliculus (SC)

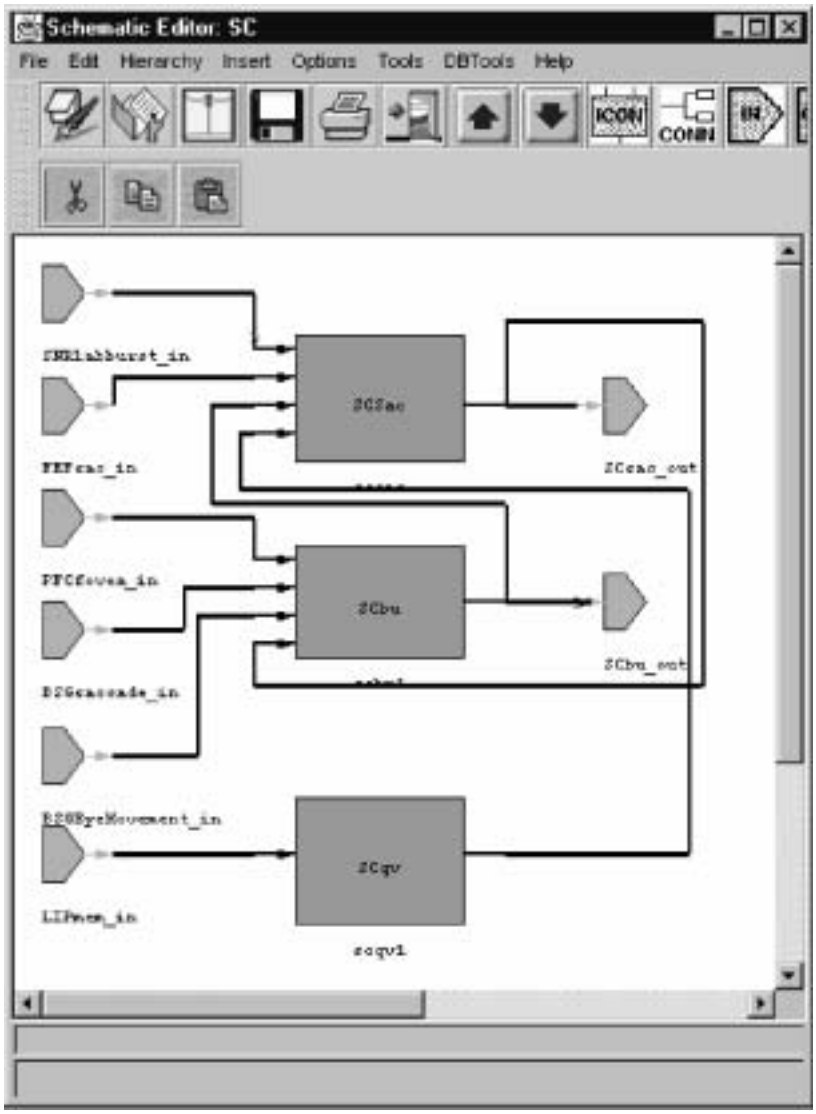

Figure 15.5
Superior Colliculus

The superior colliculus can be divided into two regions (Mason and Kandel 1991): the superficial layers and the intermediate and deep layers. The 3 superficial layers of SC

receive both direct input from the retina and a projection from striate cortex for the entire contralateral visual hemifield. Neurons in the superficial SC have specific visual receptive fields: Half of the neurons have a higher frequency discharge in response to a visual stimulus when a monkey is going to make a saccade to that stimulus. If the monkey attends to the stimulus without making a saccade to it, for example by making a hand movement in response to a brightness change, these neurons do not give an enhanced response.

Cells in the two intermediate and deep layers are primarily related to the oculomotor system. These cells receive visual inputs from prestriate, middle temporal, and parietal cortices, and motor input from FEF. In addition, there is also representation of the body surface and of the locations of sound in space. All of these maps are in register with the visual maps. Among the various saccade generation and control by superior colliculus hypothesis we are using the relatively recent one due to (Munoz and Wurtz 1993; Optican 1994). This revised theory proposes that the activity of one class of saccade-related burst neurons (SRBN) declines sharply during saccades, but the spatial location of this activity remains fixed on the collicular motor map. The spatial activity profile in another class of saccade-related cells, called buildup neurons, expands as a forward progression in the location of its rostralmost edge during the saccade. Eventually the expanding activity reaches fixation neurons in the rostral pole of the colliculus, which become reactivated when the balance between the declining activity of the SRBNs and the fixation cells again tips in favor of the fixation cells. Reactivation of these fixation neurons, which have been hypothesized to inhibit more caudally located burst neurons in the rest of the colliculus in turn functions to terminate the saccade. Buildup neurons may also be located in the intermediate layers, but are more ventrally situated with respect to SRBNs.

Since the superficial SC layer does not project directly to the intermediate/deep layer, we will only model the intermediate/deep layer. We also will not use the FOn cells in FEF to directly inhibit the SC, instead we will use rostral SC as the inhibitory mechanism. Thus, saccades will be inhibited when there is fixation on the fovea and saccades will be terminated when the buildup neuron activity reaches the rostral pole of the SC. Target locations for the SRBNs will be mapped as quasi-visual cells receiving their input from LIP. The model implement SC as composed of four types of neurons.

Quasi-Visual Cells (SCqv) are visually responsive neurons and receive topographically organized output from the LIP (LIPvis). They project to saccade related burst neurons (SCsac), passing the visual information they have received from LIP.

Saccade Response Cells (SCsac) are the SRBN neurons responsible for the initiation of saccades by their projection to the long-lead burst neurons in the brainstem. These cells receive inhibitory afferents from the substantia nigra pars reticulata (SNr) and excitatory input from the SCqv cells, they also receive excitatory input from the FEF saccade-related neurons.

Buildup Cells (SCbu) are retinotopically organized, but the activity that arises at the beginning of a saccade acts as a moving hill towards the central element of this array which represents the fixation cells described below (SCfixation). Corollary feedback from the eye movement cells (BSGEyeMove) provide the information needed as to how far the eye is being moved. This controls the rate of progression of activity in these neurons towards the fixation cells, determining when the saccade is to be terminated.

Fixation Cells (SCfixation) represent the rostral pole of SC. Once the locus of activity in the buildup cells (SCbu) reaches the central element of this array, an inhibitory signal is propagated to the burst neurons (SCsac) in SC and the brainstem (BSGsac) to terminate the saccade.

Frontal Eye Field (FEF)

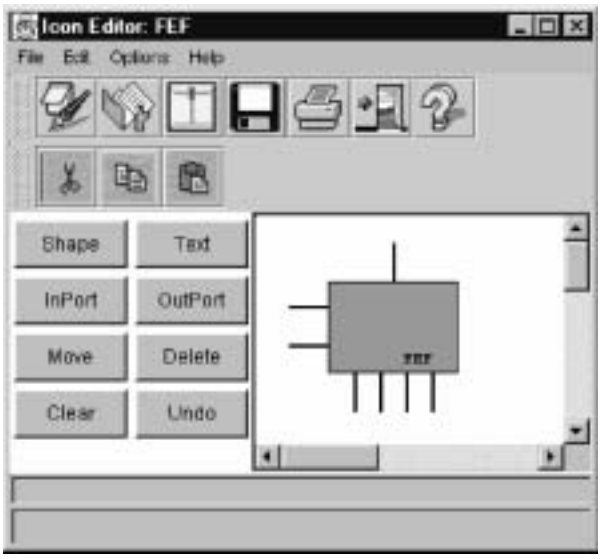

Figure 15.6
Frontal Eye Field

In FEF layer we model two types of saccade-related cells in FEF and a third type of cell relating to saccade inhibition (when a target is foveated)

Memory Response Cells (FEFmem) fire continuously during the delay portion of a delay saccade task. It has been found that there are neurons in FEF, in a double saccade task that would begin firing after the first saccade and continued firing until the second saccade (Goldberg and Bruce 1990). The cell would fire even if the stimulus never entered their receptive field when the second saccade was arranged so that it matched the cell's movement or receptive field. In the model a memory loop is established between these cells in FEF and mediodorsal thalamus (McEntee, Biber et al. 1976; Squire and Moore 1979; Fuster 1973). This memory loop is modulated by the inhibitory activity of BG upon the thalamus relay cells. The remapping of saccade targets performed by the BG is sent to thalamus. The connection strength between FEF and thalamus is chosen so that it is not strong enough to block BG from washing out the memory traces but strong enough to form a memory for the saccade targets.

Saccade Cells (FEFsac) are presaccadic movement neurons that respond to both visually and memory-guided saccades. These cells code for particular saccades.

Foveal Response Cells (FEFfovea) respond to visual stimuli falling on the fovea. They receive this input from LIP (LIPvis neurons) and project this information to BG and to the fixation neurons in SC (SCfixation neurons).

Basal Ganglia

The basal ganglia consist of five subcortical nuclei: the caudate nucleus (CD), putamen, globus pallidus, subthalamic nucleus, and substantia nigra. The neostriatum, or striatum, consists of both the caudate nucleus and putamen as they develop from the same telencephalic structure. The striatum receives nearly all of the input to the basal ganglia, receiving afferents from all four lobes of the cerebral cortex, including sensory, motor, association, and limbic areas. However, it only projects back to frontal cortex through the thalamus. This cortical input is topographically organized (Alexander, Crutcher et al. 1990; Alexander, R et al. 1986; Gerfen 1992; Parent, Mackey et al. 1983). There is also significant topographically organized input from the intralaminar nuclei of the thalamus (Cote and Crutcher 1991; Kitai, Kocsis et al. 1976; Sadikot, Parent et al. 1992; Wilson,

Chang et al. 1983): the centromedian nucleus projects to the putamen and the parafascicular nucleus projects to the caudate nucleus.

The model deals specifically with the saccadic oculomotor system within the brain. For this reason, the internal globus pallidus and the putamen in the model is not included in the model since they are more involved in motor control.

In terms of saccadic eye movement control the model proposes the purpose of the basal ganglia to be twofold:

- A lateral circuit that inhibits saccadic motor commands from execution until a trigger signal is received from higher motor centers, e.g., the prefrontal cortex.
- A medial circuit that estimates the next sensory state of the animal through an associative network for the execution of voluntary motor commands. This network receives as input the current sensory state, from LIP, and the currently planned motor command, from FEF, and outputs the next sensory state to limbic cortex and prefrontal cortex.

Lateral Basal Ganglia

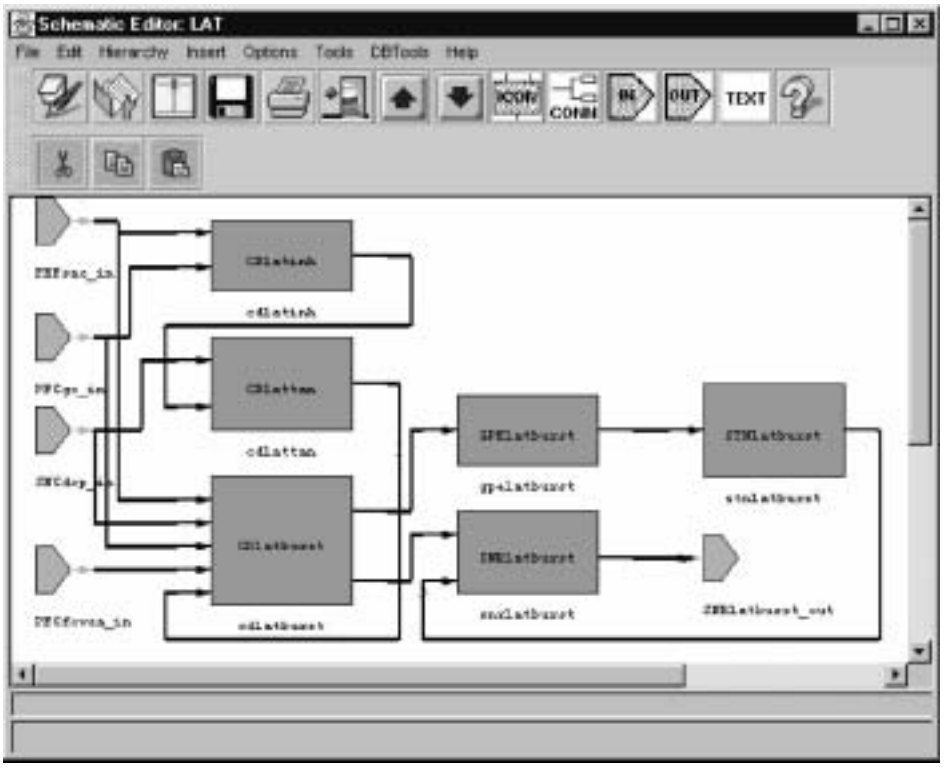

Figure 15.7
Lateral Basal Ganglia

The lateral basal ganglia circuit inhibits saccadic motor commands from execution until a trigger signal is received from higher motor centers, e.g., the prefrontal cortex. The lateral circuit is modeled as different set of cell groups. The following describes these cell groups.

Caudate Burst Cells (CDlatburst) are typically quiet and are tonically inhibited by the TAN interneurons. They receive excitatory input from cortex and the thalamus. These cells project to lateral SNr and GPe. They also receive afferents from the SNc dopaminergic cells.

Caudate Tonically Active Cells (CDlattan) are interneurons that fire continuously except when a go signal is received from prefrontal cortex. They are inhibited by the non-

dopaminergic interneurons in the caudate. These cells also receive inhibitory input from the SNc dopaminergic neurons.

Caudate Non-dopaminergic Interneuron Cells (CDlatinh) are normally quiet. In the lateral circuit, these neurons receive the motor command from FEF and a go signal from PFC. When this input exceeds a certain threshold, these cells will fire and inhibit the tonically active interneurons (CDlattan).

GPe Burst Cells (GPElatburst) are tonically active and receive inhibition from the caudate burst cells. These cells project to the STN burst cells.

STN Burst Cells (STNlatburst) receive tonic inhibition from the GPe burst cells. These excitatory cells project to the SNr topographically, but with a wider projection area than that of the direct part from the striatum to SNr.

SNc Dopaminergic Cells (SNCdop) project to the burst cells and tonically active cells in the caudate. They receive excitatory afferents from limbic cortex about primary reward related events.

SNr Burst Cells (SNRlatburst) are tonically active and receive inhibition from the caudate burst cells and excitation from the STN burst cells. These cells project to the thalamus and SC and are responsible for inhibiting the execution of a saccade motor command until deactivated by a corticostriatal "go" signal.

Medial Basal Ganglia

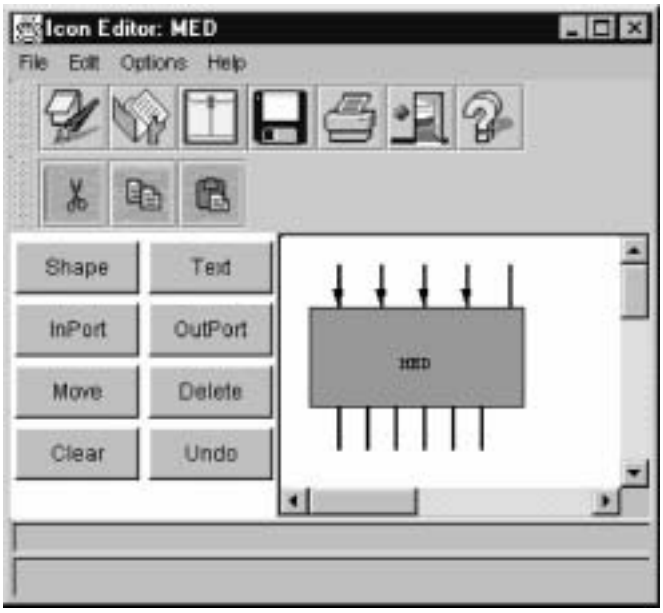

Figure 15.8
Medial Basal Ganglia

As in the lateral case the medial basal ganglia circuit is modeled as different set of cell groups. The following describes these cell groups that are modeled.

Caudate Burst Cells (CDmedburst) are typically quiet and are tonically inhibited by the TAN interneurons. They receive excitatory input from cortex and the thalamus. These cells project to medial SNr and GPe. They also receive afferents from the SNc dopaminergic cells.

Caudate Tonically Active Cells (CDmedtan) are interneurons that fire continuously except when a behaviorally significant, i.e., primary reward, signal is received from SNc through an increase in dopamine. They are inhibited by the non-dopaminergic interneurons in the caudate.

Caudate Non-dopaminergic Interneuron Cells (CDmedinh) are normally quiet. In the medial circuit, these neurons receive the motor command from FEF and the possible

saccade targets from LIP. When this input exceeds a certain threshold, these cells will fire and inhibit the tonically active interneurons (CDmedtan).

GPe Burst Cells (GPEmedburst) are tonically active and receive inhibition from the caudate burst cells. These cells project to the STN burst cells.

STN Burst Cells (STNmedburst) are typically quiet and receive tonic inhibition from the GPe burst cells. These excitatory cells project to the SNr topographically, but with a wider projection area than that of the direct part from the striatum to SNr.

SNc Dopaminergic Cells (SNCdop) project to the burst cells and tonically active cells in the caudate. They receive excitatory afferents from limbic cortex about primary reward related events. These are the same cells as in the lateral circuit.

SNr Burst Cells (SNRmedburst) are tonically active and receive inhibition from the caudate burst cells and excitation from the STN burst cells. These cells project to the thalamus and thalamic reticular nucleus and are responsible for inhibiting the thalamic activity for the current sensory state and facilitating the growth of activation for the next sensory state.

Brain Stem Saccade Generator

Figure 15.9
Brain Stem Saccade Generator I

Once the PFC issues a 'GO' signal the combination of increasing activity from PFC and decreased inhibition from BG allow activation to grow in the SC. This activation is projected to the brainstem where motor neurons are excited and cause the eye muscles to move the eyes to the new target location.

Brainstem Saccade Generator generates the saccade depending on the outputs of SC, where SCbu acts as inhibitory and SCsac excitatory. Once a saccade started SCbu neurons' activity start to grow and the activity of SCsac neurons start to decrease. Eventually the saccade ends (Note this is implemented in BSG). The BSG module is modeled as two types of cells:

BSG Saccade Generating Cells (BSGsac) are a composite of the burst, tonic and omnipause neurons in the brainstem. This layer receives the saccade command from SCsac and generates the saccade velocity and amplitude. They project to the BSGEyeMove layer. They also receive inhibitory feedback from SC buildup cells which inhibits saccades from occurring.

BSG Eye Movement Cells (BSGEyeMove) are equivalent to the brainstem motor neurons that actually drive the eye muscles. Corollary discharge from these neurons is received by SC buildup neurons (SCbu) to control the progression of their activity toward the rostral pole of SC. These neurons also receive activity from the SC buildup neurons (from the rostral pole only). This specific SC activity terminates an ongoing saccade.

15.3 Model Implementation

The complexity of this model in terms of the high number of brain areas and cell types involved, as well as the number of experiments implemented results in a wide use of the NSLM language functionality. Since the code is very long, we will focus on the special features that this model makes use of, like for example buffered ports to make processing order unimportant. The multiple experiments studied lead us to define new protocols and canvases. The protocols allow us to easily select and control the experiment to be simulated. The canvas offers an interactive way to collect and display experiment related information intuitively. Finally, we will explain how to extend NSLM to provide additional functionality not directly available in the language.

As always we need to define the top most module in the hierarchy, where we declare the model's constants, global variables, children modules, input and output modules and simulation methods such as initSys, initModule and makeConn. The top most module has to be defined with the reserved keyword nslModel. The children modules are those we previously saw in figure 1: Lateral Intraparietal Cortex (LIP), Thalamus (Thal), Basal Ganglia (Med, Lat, SNC), Prefrontal Cortex (PFC), Superior Colliculus (SC), Frontal Eye Field (FEF) and Brain Stem (BSG). To show the activity of the different cells we take advantage of the standard output interface (CrowleyOut). However, since we have different experiments we will extend the standard input interface with two canvases to collect the data for each of them (DoubleSaccadeInterface, GapSaccadeInterface).

At the instantiation of the model, the method initSys is called. Within this method we assign the values of the simulation parameters: simulation end time, step length and port buffering type. Once all the modules have been created, the scheduler executes the initModule method. Here we declare the protocols associated with each experiment using the method nslDeclareProtocol that adds new entries to the protocol menu (see figure 10). We will latter need to define which module will be part of which protocol. For that purpose we call the nslAddProtocolToAll function that add all modules to a particular protocol. Finally the method makeConn communicates the different modules by connecting their input and output ports. To connect siblings we use the nslConnect call, whereas nslRelabel allows a children module to inherit ports of their parents.

```
nslModelCrowleyTop()
{
    nslConst int CorticalArraySize = 9;
    nslConst int StriatalArraySize = 90;

    private int half_CorticalArraySize;
    public NslInt0 FOVEAX(half_CorticalArraySize);
    public NslInt0 FOVEAY(half_CorticalArraySize);

    // input modules that hold single output matrices
    VISINPUT visinput(CorticalArraySize);
    LC lc(CorticalArraySize);

    // LIP and Thalamus
    LIP lip(CorticalArraySize);
    Thal thal(CorticalArraySize);

    // Medial circuit
    Med med(CorticalArraySize, StriatalArraySize);

    // Lateral Circuit
    Lat lat(CorticalArraySize);
    SNC snc(CorticalArraySize);

    // Others
    PFC pfc(CorticalArraySize);
    SC  sc(CorticalArraySize);
    FEF fef(CorticalArraySize);
    BSG bsg(CorticalArraySize);

    // Graphic interfaces
    private   CrowleyOut
       crowout(CorticalArraySize,StriatalArraySize);
    private   DoubleSaccadeInterface doubleSaccade();
    private   GapSaccadeInterface gapSaccade();

    public void initSys(){
       system.setEndTime(0.55);
       system.nslSetRunDelta(0.001);
    //all output ports will be double buffered
       system.nslSetBuffering(true);

       half_CorticalArraySize = CorticalArraySize / 2;
    }

    public void initModule(){
       nslDeclareProtocol("gap", "Gap Saccade");
       nslDeclareProtocol("double", "Double Saccade");

       system.addProtocolToAll("gap");
       system.addProtocolToAll("double");
    }

    public void makeConn() {
       // LIP inputs
       nslConnect(visinput.visinput_out , lip.SLIPvis_in);
       nslConnect(thal.ThLIPmem_out     , lip.ThLIPmem_in);
    …
    }
}
```

Figure 15.10
Standard executive interface extended with new protocols.

As we mentioned before we utilize the standard output interface to graphically display the neural activity of the model. For that purpose we create a nslOutModule which includes all the functionality of a normal nslModule, but it incorporates a NslFrame, a window where graphs are displayed. Commonly this module contains the definition of input ports where the information will arrive. If we want this data to be displayed, we have to create a canvas and associate it with a particular input port. We do this with the nslAddCanvas methods family (e.g. nslAddAreaCanvas and nslAddSpatialCanvas).

```
nslOutModule CrowleyOut
    (int CorticalArraySize, int StriatalArraySize) {

    //input ports
    public NslDinFloat2
        visinput(CorticalArraySize, CorticalArraySize);
    public NslDinFloat2
        pfcGo(CorticalArraySize, CorticalArraySize);
    public NslDinFloat2
        lipMem(CorticalArraySize, CorticalArraySize);
    public NslDinFloat2
        thna(CorticalArraySize, CorticalArraySize);
    public NslDinFloat2
        fefsac(CorticalArraySize, CorticalArraySize);
    public NslDinFloat2
        scsac(CorticalArraySize, CorticalArraySize);
    public NslDinFloat2
        scbu(CorticalArraySize ,CorticalArraySize);

    public void initModule() {
        nslAddAreaCanvas(visinput,0,100);
        nslAddAreaCanvas(lipMem,0,100);
        nslAddAreaCanvas(thna,0,10);
        nslAddAreaCanvas(fefsac,0,100);
        nslAddAreaCanvas(scsac,0,100);
        nslAddSpatialCanvas(scbu,0,10);
    }
}
```

In order to build the new input user interface, two steps are required. The first one is the definition of a NslInModule. Within this module we associate an instance of the new canvas with an output port, where the information collected by the interface will be sent. For this purpose we use the nslAddUserCanvas method, which takes as parameters an outputPort and the name of the nslClass that implements the new canvas. In addition we call nslRemoveFromLocalProtocols function to remove this module and it's window from

the “manual” and “gap” protocols. This ensures that this interface will only be available when the “double” protocol is selected.

```
nslInModule DoubleSaccadeInterface() {

    NslDoutDouble1 params(8);

    public void initModule(){
        nslAddUserCanvas(params,”DoubleSaccade”);
        nslRemoveFromLocalProtocols(“manual”);
        nslRemoveFromLocalProtocols(“gap”);
    }
}
```

In the second step is the implementation of nslClass that defines the new canvas. This has to be a subclass of NslInCanvas from which it inherits methods to handle input events and display graphics. As a NslInCanvas subclass it has to take two parameter, the first of them being the NslFrame where the canvas will be displayed and second a wrapping object that contains the port given by the parent NslInModule.

Every time the canvas has to be repainted, the nslRefresh method is called. Within this method we can draw lines, shapes, strings, change colors, etc. Every simulation step, the nslCollect function is executed, allowing input data to be gathered and sent to all the involved modules.

```
nslClass DoubleSaccade (NslFrame frame, NslVariableInfo vi)
    extends NslInCanvas(frame,vi) {

    public void nslInitCanvas() {
        nslClearDisplay();
    }

    public void nslRefresh() {
        drawSaccadeTargetLocations();
        drawSaccadeTargetDurations();
        …
    }

    public void drawSaccadeTargetDurations() {
        int gx0, gx1, gy0, gy1, h, w;
        int x0, x1, y0, y1;

        float fix_start, fix_end;
        float t1_start, t1_end, t2_start, t2_end;

        NslString0 xTicks();

        int i;

        fix_start = (float) 0.;
           fix_end   = (float) 0.2;
        t1_start  = (float) 0.05;
        t1_end    = (float) 0.1;
        t2_start  = (float) 0.1;
        t2_end    = (float) 0.15;

        h = nslGetHeight();
        w = nslGetWidth();

        // Draw grid
        gx0 = w / 10;
```

```
        gx1 = w - gx0;
        gy0 = h / 5;
        gy1 = h - gy0;
        nslDrawLine(gx0,gy1,gx1,gy1,"black"); // X-axis
        // X-ticks
        y0 = gy1 - 5;
        y1 = gy1 + 5;
        for(i=0;i<=12;i++){
               x0 = x1 = gx0 + ((gx1 - gx0) * i)/12;
               nslDrawLine(x0,y0,x1,y1,"black");
               xTicks.set(i*5./100.);
               if(i%2 == 0)
                  nslDrawString(xTicks.get(),x0-5,y1+15);
           }

        // Draw time bars
        y1 = gy0/2;
        // Fixation
        y0 = gy0;
        x0 = gx0 + (int)((fix_start/.6)*(gx1-gx0));
        x1 = (int) (((fix_end-fix_start)/.6)*(gx1-gx0));
        if (x1<=0)
               x1 = 1;
        nslFillRect(x0,y0,x1,y1,"red");

        // T1
        y0 = gy0*2;
        x0 = gx0 + (int)((t1_start/.6)*(gx1-gx0));
        x1 = (int)(((t1_end-t1_start)/.6)*(gx1-gx0));
        if(x1<=0)
               x1 = 1;
        nslFillRect(x0,y0,x1,y1,"green");

        // T2
        y0 = gy0*3;
        x0 = gx0 + (int)((t2_start/.6)*(gx1-gx0));
        x1 = (int)(((t2_end-t2_start)/.6)*(gx1-gx0));
        if(x1<=0)
               x1 = 1;
        nslFillRect(x0,y0,x1,y1,"blue");

        }
        public void nslCollect() {

           NslNumeric1 params = (NslNumeric1)vi.getNslVar();

        params[0] = getXFixValue();
        params[1] = getYFixValue();
        …
        }
        …
}
```

To provide the continuous remmaping capability we utilized in the buildup neurons in the superior colliculus, we had to create a mechanism to keep track of the location of the centroid of the "moving hill", as we did not have enough neurons to allow the activity to propagate "naturally". We created a NSLM class called Target that had x and y coordinates as well as a variable to support a list of Target objects. We created a member

function called Move that accepted a two-element vector of an x, y delta to be moved. This function applied the movement to the current location of the Target. A separate function applied to the new location of the buildup neuron targets onto the buildup neurons to simulate the continuous movement across the buildup cells.

```
nslClass Target() {
   // This class provides for a linked list of target objects
   // that all have the size of a single array element.
   // The contents of this class are the x,y coordinates
   // of the corner closest to array element 0,0, and a pointer
   // to the next Target in the list.  The x-coordinate is the
   // first sort.

   private double  xcor, ycor;
   private Target  next;

   initTarget() {
      xcor = 0; ycor = 0; next = nslNull;
   }

   …

   void Move( NslDouble1 invec ){
   // This method applies the input movement vector to all of
      the
   // Targets in the linked list. The x,y-coordinates of each
   // Target have the input movement vector subtracted from
      their
   // corner coordinates as the motion of the Targets across
      the
   // visual space is in the opposite direction to the movement
   // of the eyes.
   Target cur;

   // Do the first target as it always exists

      xcor = xcor – invec[0];
      ycor = ycor – invec[1];

      cur = next;  //get pointer to next Target

      // The do-while will “move” the second and higher
      // Targets if they exist
      while ( cur != nslNull )      {
         cur.xcor = cur.xcor – invec[0];
         cur.ycor = cur.ycor – invec[1];
         cur = cur.next;
      }
   }

   …

   double X() {return xcor;}
   double Y() {return ycor;}
   Target Next() {return next;}
}
```

Our most comprehensive extension to NSLM was the ability to map arbitrary neurons in one layer onto a larger layer. This was the basis of out remapping algorithm between

the cortex and the basal ganglia. Specifically, we created a linked list mechanism for each element in the input layer (FEF, LIP and PFC) for our model that pointed to all neurons to which they project (striatum in or case). Thus, for any given input neuron, you only need to read the linked list out to determine its projections. We used the same mechanism to establish the remapping from striatum to SNr. In this case, however, there were multiple connections onto SNr from striatum, but the same principle applies. You can find out which striatal neurons talk to a specific SNr neuron by just indexing the linked list for that neuron. This "bi-directional" mapping made the teaching of the weights between cortex, striatum and SNr very simple, since we specified the cortical inputs and knew what SNr outputs we wanted. It was simple to match the linked lists that both pointed back to the striatum and then modify the weight matrix for the striatum. Summing the SNr inputs during runtime was also simplified as we accessed the linked list for each SNr neuron and summed the inputs for that neuron by reading the list only once per time step.

```
nslClass Element() {

    int x, y, xo, yo;
    Element next;

    nslConst int FOVEAX = 4;
    nslConst int FOVEAY = 4;
    …
    public void initElement() {
        x = y = x0 = y0 = -1;
        next = nslNull;
    }
    …
    public void Remap(int max, Element elem) {
        // This function "remaps" the calling Element and
           returns an
        // Element containing the remapped location.
        int xt, yt, xot, yot;

        xt = FOVEAX - x; yt = FOVEAY - y;

        xot = xt + xo;   yot = yt + yo;

        elem.x = FOVEAX; elem.y = FOVEAY;

        if ( ( xot > -1 ) && ( xot < max ) )
            elem.xo = xot;
        else
            elem.xo = -1;

        if ( ( yot > -1 ) && ( yot < max ) )
            elem.yo = yot;
        else
            elem.yo = -1;
        }
        public Element Next() { return next; }
        public int X() { return x; }
        public int Y() { return y; }
        public int XO() { return xo; }
        public int YO() { return yo; }
    }

    nslModule Med (int CorticalArraySize, int StriatalArraySize)
        extends Func (CorticalArraySize) {
```

```
   private nslConst int MaxConnections   = 50;
   private nslConst int NumberIterations = 10;
   // Output ports
   public NslDoutInt3
FEFxmap(CorticalArraySize,CorticalArraySize,MaxConnections);
   public NslDoutInt3
FEFymap(CorticalArraySize,CorticalArraySize,MaxConnections);
   public NslDoutInt3
LIPxmap(CorticalArraySize,CorticalArraySize,MaxConnections);
   public NslDoutInt3
LIPymap(CorticalArraySize,CorticalArraySize,MaxConnections);
   public NslDoutInt3
PFCxmap(CorticalArraySize,CorticalArraySize,MaxConnections);
   public NslDoutInt3
PFCymap(CorticalArraySize,CorticalArraySize,MaxConnections);

   // See MappingParameters
   private int FEFPatchCount;
   private int LIPPatchCount;
   private int PFCPatchCount;

   private Element LearnedElements();
   private Element UnlearnedElements();
   private Element Teacher();

   public void initRun () {
      MakeMapping();
      …
      LearnNewElements();
   }
   …
   public void MakeMapping() {
      int MapSize = StriatalArraySize/3;
      …
      // Establish the direct path mapping from CD to SNr
      SNRMapping(FEFxmap, FEFymap, FEFPatchCount, MapSize);
      SNRMapping(LIPxmap, LIPymap, LIPPatchCount, MapSize);
      SNRMapping(PFCxmap, PFCymap, PFCPatchCount, MapSize);
      …
   }
   public void learnNewElements() {
      LearnConnections(UnlearnedElements);
      LearnedElements.Merge(UnlearnedElements);
      UnlearnedElements.Remove();
   }
   public void LearnConnections(Element elem) {
      Element curelem(elem);
      while ( curelem != null ) {
      for (int ii=0; ii<NumberIterations; ii++ ) {
         //# of iterations
         // Set cortical excitation
         …
         // Determine correct remappings for non-neural
            Teacher
         curelem.Remap((int)CorticalArraySize, Teacher);
         MapToFovea(curelem.X(), curelem.Y());
```

```
                // Time to map the nonsaccade target as well
                …
                // increment weights between active CD neurons
                // and remapped location
                MapToOffset(curelem.X(), curelem.Y(),
                    curelem.XO(), curelem.YO());
                }
                curelem = curelem.Next();
            }
        }
}

nslModule SNRmedburst (int CorticalArraySize,
        int StriatalArraySize) {
    public NslDinDouble3 SNRweights(CorticalArraySize,
        CorticalArraySize, CorticalArraySize);
    public NslDinInt3 SNRxmap(CorticalArraySize,
        CorticalArraySize, CorticalArraySize);
    public NslDinInt3 SNRymap(CorticalArraySize,
        CorticalArraySize, CorticalArraySize);
    public NslDinDouble2
        CDdirmedburst_in (StriatalArraySize, StriatalArraySize);

    private NslDouble2 SNRcdinput
        (CorticalArraySize, CorticalArraySize);
     …
    public void SumCDtoSNR (NslDouble2 CD, NslDouble2 SNR) {
        // This function sums the activity in the medial CD
        // circuit onto the medial SNR circuit through
        // SNRweights, SNRxmap and SNRymap.

        int i, j, k, xmaploc, ymaploc;

        SNR = 0;  // Ensure new mapping only

        for (i = 0; i < CorticalArraySize; i ++) {
            for (j = 0; j < CorticalArraySize; j ++) {
                for (k = 0; k < SNRMapCount [i][j]; k ++) {
                    xmaploc = SNRxmap [i][j][k];
                    ymaploc = SNRymap [i][j][k];
                           SNR [i][j] = SNR[i][j]
                               + CD [xmaploc][ymaploc] *
                               SNRweights [i][j][k];
                    }
                }
        }
    }
    …
    public void simRun () {
        SumCDtoSNR (CDdirmedburst_in, SNRcdinput);
        …
    }
}
```

15.4 Simulation and Results[1]

The Crowley model can be tailored to test various experimental saccade paradigms. The current version of the model has two built-in paradigms: gap saccade and double saccade. The latter is more subtle. So we are going to go over the double saccade paradigm and show how the model can reproduce real world experimental results.

The double saccade can be briefly described as the following. The subject is presented a fixation point which he must maintain a fixation until the stimuli is there. After some certain time delay first target is flashed somewhere in the visual field of the subject. It is followed by a second flash of target which may or may not overlap with the first stimuli. While the targets are being shown the subject must still maintain his fixation. Only after the fixation goes off the subject can make the saccades. The saccades he makes must follow the right temporal order. That is the first saccade to the first target and the second saccade to the second target.

In order to enable user to modify certain paradigm specific parameters in a convenient way a custom user interface is designed for Crowley Model. The user can bring either of the paradigms' user interface by using via experiment menu. The double saccade user interface gives user the convenience of setting up experiment parameters by dragging and clicking. Figure 15.11 shows how the double saccade interface looks like:

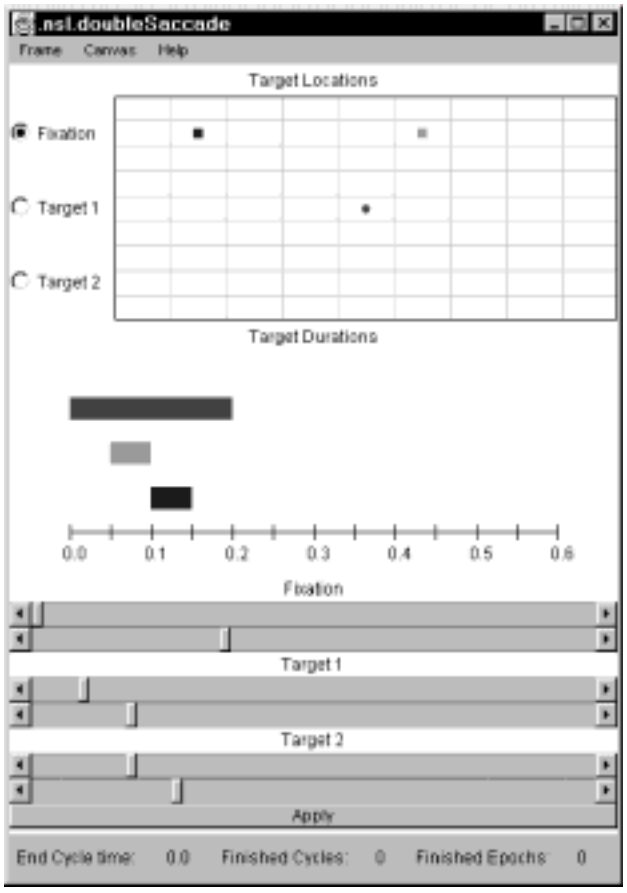

Figure 15.11
Double saccade experiment interface window

The upper part of the window is used to specify the location of the targets and the fixation point. The user needs simply to click on one of the buttons for fixation, target1 or target2 and point the position of the stimulus on the grid. Fixation point is denoted by blue, target 1 is by green and target 2 by red color. Once the user specifies the position of the stimuli then he/she has the opportunity to modify the timing of the stimuli by simply dragging the gauges on the lower level of the interface window. For example on the sam-

ple interface window the visual events that occur can be described as follows. At time 0.0 the fixation stimuli appears. At time 0.5 first target appears. At time 0.1 first target disappears and target two appears. At time 0.15 target two disappears. Finally the fixation stimulus goes away at time 0.2. Once the spatial and temporal characteristics of the double saccade paradigm is specified user has to click on Apply button to load the settings into the simulator. Then the double saccade experiment can be simulated by clicking the Start button. The simulator will, then, create the visual events defined by the user and simulate the model. NSL display window can be used to display the model variables as usual. The Crowley Model comes with a NSL display window with 8 graphic displays as shown in the below figure. The visual events specified occurs in the top left graph. Other graphs show the various model variables. For example the CrowleyTop.lip.LIPmem_out labeled graph shows the NSL variable LIPmem_out which is defined in lip module. In regard to model semantics this layers keeps the memory of the visual stimuli.

The activity of the buildup neurons in SC can be used to track the saccades that the model executes. The moving activity towards to center following by a decay at the center in SCbu1 layer corresponds to a saccade that the model executes. Thus a double saccade would mean two consecutive buildup neuron activity. The SCbu1 layer is retinotopically organized so the start of the activity corresponds to the target stimulus for the saccade. However the second target is remapped with anticipation of the first saccade. So the second saccade activity seen in buildup neurons are based on the remapped location.

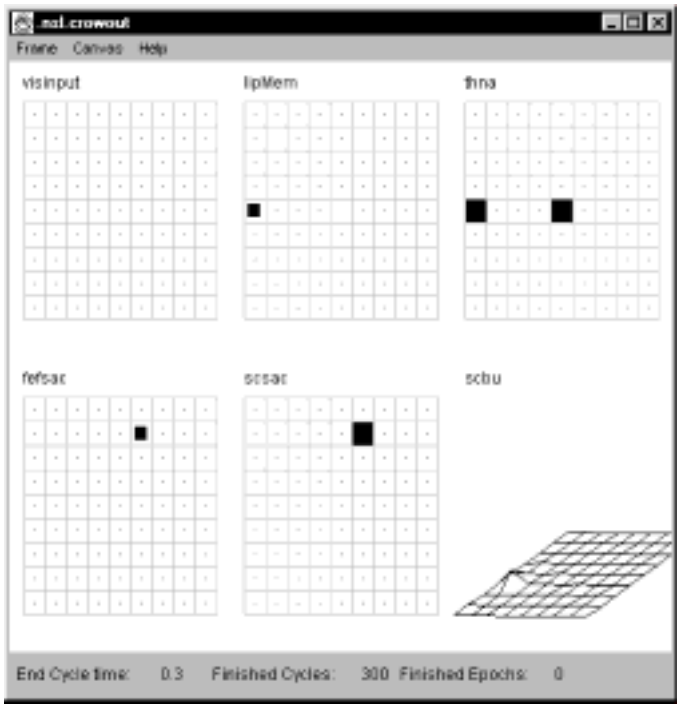

Figure 15.12
First saccade in progress

The buildup neuron activity is shown using a 3d graph (figures 15.12 and 15.13). The targets in this simulation run were horizontally aligned and above the fixation point. The first target were also aligned with the fixation point. The system was expected to make a vertical (upward) saccade then a horizontal (rightward) saccade to the second target. If there were no remapping the second saccade would not be horizontal but it would be an up-right one. The prediction for the buildup neuron activity was to have two perpendicular activities decaying at the center. The following two figures demonstrate the expected result. First figure shows the simulator display window during first saccade. The

second figure shows the expected second saccade (Note that the direction of the saccades are towards the center so they are perpendicular).

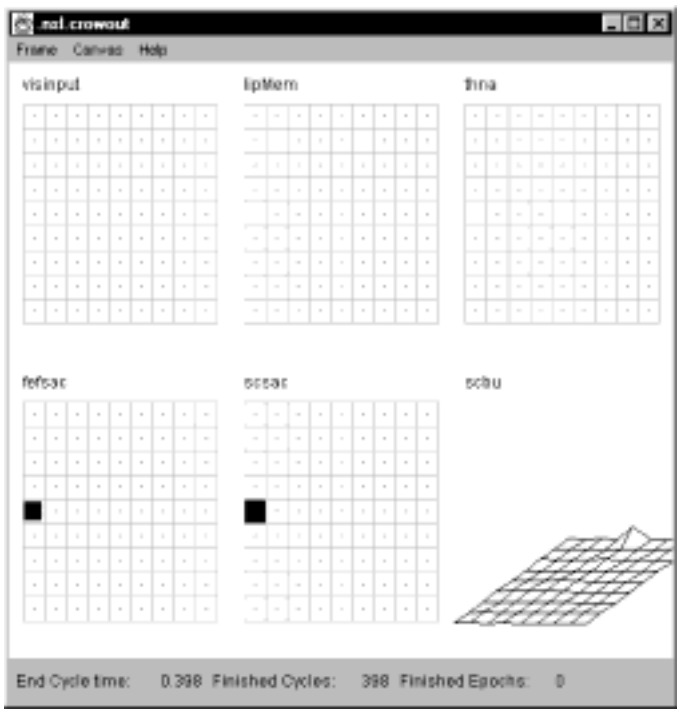

Figure 15.13
Second saccade in progress

15.5 Summary

We have developed a neural network model of the saccadic motor control system that includes a number of cortical and subcortical systems known to be involved in saccadic eye movements. One primary thesis of our model is that the basal ganglia has two primary roles in saccadic motor control: (1) inhibition of voluntary saccadic eye movements until cortical centers provide a go signal and (2) provide for the sensory remapping of potential saccade targets based upon an impeding saccade command. Additionally, we have implemented a mechanism simulating the effects of dopamine deficit in saccadic eye movements. Lastly, we have implemented a model that places the superior colliculus within a feedback control loop responsible for terminating saccades.

We have made use of the NSLM language functionality showing how buffered ports can be utilized to make the order in which modules are executed unimportant. We have explained the creation of standard and custom user interfaces. We reviewed how protocols are declared to simulate different experiments for the same model. We provided an example of how to extend NSLM to obtain additional functionality not directly available in the language. We ended showing how the NSL simulation environment can be used to run the implemented model for the double saccade experiment.

Notes

1. The Saccade model was implemented and tested under NSLJ.