A Neural Network Based Testbed for Modelling Sensorimotor Integration in Robotic Applications[1]

Andrew H. Fagg
ahfagg@pollux.usc.edu

Irwin K. King[§]
king@rana.usc.edu

M. Anthony Lewis
mlewis@pollux.usc.edu

Jim-Shih Liaw
liaw@rana.usc.edu

Alfredo Weitzenfeld
alfredo@rana.usc.edu

Center for Neural Engineering
University of Southern California
Los Angeles, CA 90089-2520

To model realistic biological behaviors such as perception and motor interactions, one requires a flexible simulation environment in which goal-oriented action and sensory perception can be integrated to produce intelligent behavior. Many researchers have focused their work primarily on one specific aspect of this loop of action and perception. Our challenge is to demonstrate a feasible environment in which both sensory processing as well as motor actions are integrated simultaneously and attached to the outside world. In this report, we describe the preliminary work implemented in our lab towards the integration of sensorimotor interactions using ***Neural Simulation Language*** (NSL) and the ***Rapid Robotics Application Development*** environment (R^2AD) for computing sensory information and creating motor behavior respectively for the purpose of visuomotor coordination in the real world.

1. Introduction

In order to build intelligent robotic machines it is necessary that there be a close integration between the perception of the external environment and the generation of actions (Arbib, 1989). In the biological domain, this tight coupling (i.e., the accuracy or the behavior and the time taken to generate the behavior) of the perception/action cycle often determines the adaptability of the organism.

Within the neural networks community, the majority of neural control work has been applied to simulated dynamic systems. The problem with relying upon a simulated system as a proof of a control technique is that much of the "reality" is left out of the simulation. It is precisely these elements of reality that make many control problems truly difficult. Although we feel that simulated systems are an important element in the development and testing of models, they must be viewed as only a stepping stone to the more interesting problems.

This need of interfacing with the real world has not been completely ignored by the research community. Some examples include the work of Kawato (1990) and Kuperstein (1988). These control systems implemented for this work, however, are generally very specific to the neural control algorithm of interest, as well as to the system to be controlled. The work of Brooks (1986) is one example of a more general control architecture. This system-building philosophy allows the extension of the control system to include more complex control algorithms, and also allows the expansion of a system to include additional sensors and actuators.

1 The research described in this paper was supported in part by grant no. 1RO1 NS 24926 from the National Institutes of Health (M. A. Arbib, principal investigator).

§ The author to whom all correspondances should be sent.

We find, however, that the system philosophy imposed by the ***Subsumption Architecture*** is too restrictive in the styles of computation that are expressible within its language. In the marriage of NSL, a general neural simulator, and R^2AD, a general robot control environment, our hope is to achieve a system that is flexible, and yet powerful enough to support the experimentation with neural systems as possible robotic controllers.

With these challenges in mind, our goal is then to provide a testbed environment using neural networks for visual information processing and motor behavior generation. We will begin by giving a brief overview of the NSL system. A simple model integrating sensory processing and motor behavior generation will also be provided.

2. Application Testbed

An environment that supports the integration between sensory perception and motion behavior generation must provide the necessary tools to implement the respective models in each domain. For the perception part, we have developed the NSL (Weitzenfeld, 1991) to model classes of neural networks performing different aspects of sensory processing. From the other perspective, the R^2AD system furnishes a flexible robotic application development platform using a graphical user interface to monitor dynamically activated processes during robot program executions (Fagg et al., 1991). Although these two developmental systems were designed and implemented independently, the internal software architecture in each system allows external access and exchange of data and control information.

2.1. Neural Simulation Language (NSL)

NSL is a general purpose neural simulation language and developmental system. The underlying system architecture of the NSL takes advantage of the object-oriented programming methodology. This programming paradigm provides: (1) fast prototyping of new neural object types, (2) consistent object interface across all objects for creating and modifying internal values, and (3) object data encapsulation that adds extra caution of data security. On the top layer which interfaces with the users, the system offers an interactive command interpreter that translates high level descriptional language for specifying neural models. Additionally, an interactive graphical tool allows the user to visualize the processed output in various ways including temporal and spatial displays.

The NSL system is composed of several modules which are arranged into two independent sub-systems, the ***Simulator*** and the ***Window Interface***. The Simulator section, where the actual neural models interact to produce outputs, further houses three sub-modules. These are the ***Processing Module***, the ***Model Language Compiler***, and the ***Command Interpreter***. The first unit contains the actual code to process the low-level computational operations. It consists of routines optimized for matrix operations (i.e. convolution, summation, etc.) and numerical methods (eg. Eüler's method). The second sub-module performs the actual translation of user designed models written in NSL into executable code and links these binary modules with other pertinent library routines before loading the user specified model into the Processing Module. Lastly, the Command Interpreter serves to execute directives from the user, such as changing simulation parameter values and output graphic types. This can be done either via an external text file describing the execution procedure or the user may interactively enter the commands. This interpretive simulation mode optimizes the process of modelling and simulation. Once a model has been implemented and compiled, the interpretive mode helps the users by allowing them the flexibility to modify model parameters at run time. Different initial values for various parameters can be read in from external files and the corresponding outputs can also be saved for later viewing and further analysis.

The Window Interface section is separated from the model and the control of the simulation system since the visualization of the computational result can be interpreted and verified independently. By doing so, the methodology liberates the programmers to isolate a specialized section of programming for maximum optimization of graphical routines. Secondly, it also presents the users a choice to view and review the data during simulation in real-time and/or after batch processing. The interface system contains the ***Graphical Displays***, where all window and graphics interaction takes place and the ***Graphics Libraries*** that stores all display functions and object libraries.

The NSL 2.1 system is written in C++ and runs on Sun workstations under Unix OS using X-windows. The system is implemented utilizing AT&T's C++ (2.0) compiler. For most modelling, the user is only required to learn the high-level NSL language. However, for advanced users a good understanding of the programming language C and C++ is desirable.

2.2. Rapid Robotics Application Development (R^2AD) System

The R^2AD system is designed to support the development of experimental robotic control programs. The system, itself, provides an overall control structure, a set of inter-process communication primitives, and a unified data storage and passing methodology. The communication primitives link processes across a heterogeneous network of processors. These unifying features allows code that is developed under one project to be easily usable for other work. A research programmer can therefore concentrate more fully on the logical structure of the experimental program, rather than on a myriad of low-level details.

An experimental program is implemented in R^2AD by a top-level application program and a set of processes referred to as ***widgets***. A widget is defined as a small body of code that performs a very specific task. It is this encapsulated body of code that acts as the unit of software sharing between projects. Many widgets within R^2AD act as interfaces between the system and the physical hardware of a sensory or actuation device. Examples of widgets that have been implemented within the USC Robotics lab include low-level controllers for the Puma 560 Arm and the VPL Dataglove. Currently, NSL acts as the high-level computational entity of an application program interacting with low level R^2AD processes. Eventually, NSL will also be used in the implementation of the widgets.

3. Application Example

Although frogs, monkeys, and humans utilize many different modalities of sensory information (i.e., auditory, olfactory, visual, etc.), our choice of sensory processing models the visual sensory modality in predator avoidance behavior found in the frogs and toads (Ewert, 1984). We focus mainly on visual motion processing and the corresponding ability to generate interactive behaviors in our demonstration. In Figure 1, a schematic diagram of the overall system architecture is presented.

Typically, these behaviors are found when large objects are looming toward the organism which provide moving expansions of the retinal image cast by the object. In frogs, this visual motion perception is achieved through the R3 and R4 ganglion cells to the T3 neurons which solely responds to the dilation of the stimulus (Liaw & Arbib, 1990). However, not all looming objects reside on the trajectory of the collision course. Hence, a avoidance response is generated if the object is directly moving along the collision path; otherwise, the organism may yield no response since it senses no immediate danger.

The Looming model in NSL extracts the center of the looming object and perceives its looming direction in order to activate the proper behavior procedure. The model consists of physiologically identified neurons each represented by an array of 32×32 leaky integrator units.

3.1. Hardware Setup

For this particular experimental setup, the R^2AD system allows the integration of three distinct systems : the Meno walking machine, the camera mounted on Meno, and the Puma 560 Arm that moves the simulated looming object. The looming detector circuit, implemented in NSL, acts as the high-level controller for the Meno subsystem.

Meno is an autonomous four legged walking machine built by the USC robotics lab to explore issues in neural control of walking. The machine measures 40 cm high, and is approximately 46 cm long. The net weight of the machine is approximately 2 KG.

Each leg of the machine has 3 degrees of motion: two planar rotational joints with axis aligned perpendicular to the ground, and a distal prismatic joint, with its axis also aligned to the ground. Each joint is accuated by a Futaba Servo motor. The actuation of the servo motors is controlled by a 68332 micro-controller on board the machine. R^2AD issues commands to Meno in the form **Turn Left**, **Turn Right**, and move **Forward** or **Backward**. An escape trajectory is executed by specifying an appropriate sequence of these movement primitives.

The widget that acts as the controller for the camera system preprocesses the incoming image by reducing the 512×480 image down to an image size of 32×32. A sequence of these reduced-resolution images are sent to the looming detector circuit, which estimates the trajectory of the predator. From this estimated trajectory, the network triggers the execution of an appropriate escape behavior.

The simulated predator is mounted on the end-effector of a Puma 650. A separate control process interfaces with the Puma control widget to generate a set of straight-line trajectories which pass near Meno.

3.2. Experimental Results

In this coordination example, we have selected four real world time-varying sequences of files representing various objects moving in the space around the visual sensor. The files are : (1) an object directly looming towards the camera, (2) an object moving transversely (frontoparallel) across the visual field, (3) a crossing looming stimulus which passes through the midline of the visual field, and (4) an object which looms from the upper visual field in a downward and leftward direction. Out of these four image segments only the first one poses a direct danger to the robot which must generate a sequence of motor patterns to flee the scene while other perceptual sequences will not induce the same behavior even though all stimuli are looming towards the robot.

We display an instance of typical input sequences in Figure 2. Figures 3 to 6 illustrate the output result from the neural layer detecting looming stimulus, the 180° movement directional-selectivity layer, and the collision layer which responses to direct looming objects. The firing patterns in these figures are topologically organized in retina coordinates with the location of the peak activity corresponding to the center of the looming stimulus.

3.3. Discussion

One of our major obstacles is the lack of sufficient computational power in processing sensory information. Even with the limited environment of 32×32, we were unable to achieve a real-time coordination between the sensory computation and motor action. However, this may be rectified adequately with speedier hardware, i. e. parallel or concurrent object-oriented hardware systems.

4. Future Work

Although our primary simulations have modelled after visual perception mechanisms, we hope to further model other sensory information processing circuitries in the future. If an individualized sensory unit could be simulated successfully, one interesting topic of research would be to find ways for neural networks having diverse modality to fuse together to provide a more realistic perception.

Being an object-oriented neural network simulator, we hope the future NSL will be able to take full advantage of concurrent/parallel hardware. This would mean a potential speed up of several magnitudes during different simulation trials.

5. Conclusion

In this paper, we have demonstrated an integration of visuomotor coordination using NSL and R^{2}AD developmental systems. By combining these environments, we gain greater flexibility in designing neural networks to model various perception and behaviors in biological systems. The cooperation between the two systems presents our first attempt in an integrated environment for the modelling of sensorimotor interactions in the real world.

This system also gives the computational neurobiologist a new avenue of investigation : the ability to see his algorithms, based upon neurophysiological and behavioral data, come to life within an artificial creature. Such a capability can allow a better behavioral comparison between the real system and the artificial algorithm. In addition, the computational neurobiologist is forced to consider all of the important elements of the control algorithm, not just those that are easily modeled and confirmed. It is our hope that these flexible testbeds for action-oriented perception modelling will eventually lead to practical real world applications resulting in more intelligent machines.

6. Acknowledgements

We wish to thank all the members of the Brain Simulation Laboratory and the USC Robotics Laboratory for their contributions to the continuing growth of NSL, R^2AD, and Meno.[2]

Figures

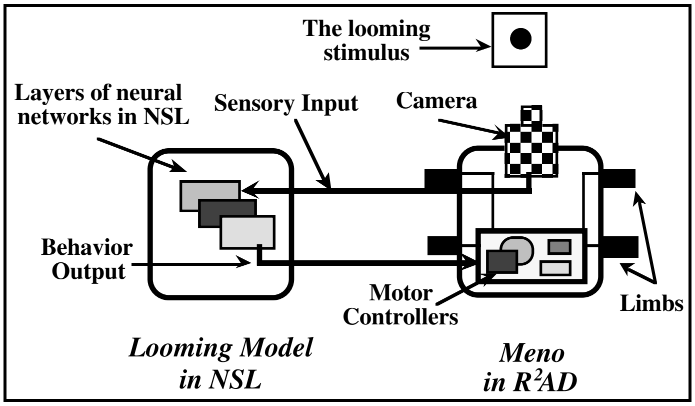
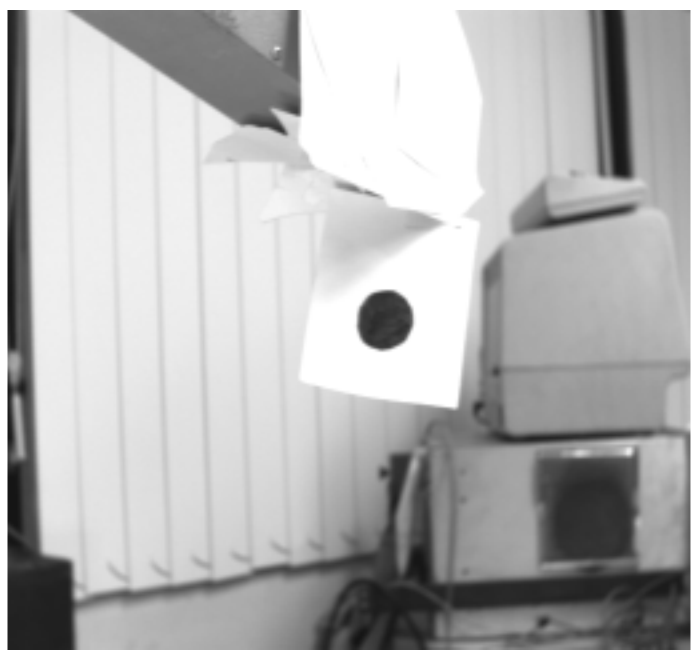


Figure 1. This schematic drawing illustrates the interaction between the perception and action systems. To the left is the sensory processing looming perception model in NSL which receives visual inputs from Meno on the right and produces neural activities in the presence of a looming object. This population firing is then sent to Meno's robot controller which in term generates a command to move the limbs synchronously to avoid the moving object.

Figure 2. Meno's view of the environment. Here shows a typical real world input image to the testbed which is processed by the neural network model in NSL. The predator is represented as a black dot mounted on the end of the Puma Arm. This "predator" is moved in a series of straight-line trajectories towards the camera. Note that the background environment is rather cluttered with a variety of stationary objects. The resolution shown here is 256 × 240 but the actual input to the looming neural network is 32 × 32.



[2] NSL is available through anonymous FTP. Please contact Alfredo Weitzenfeld for more information on obtaining the software environment and the associated manuals.

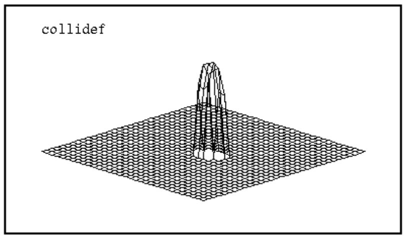
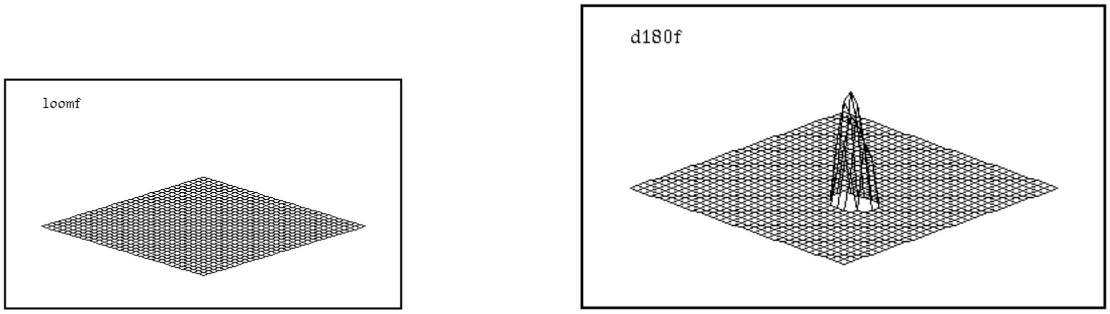
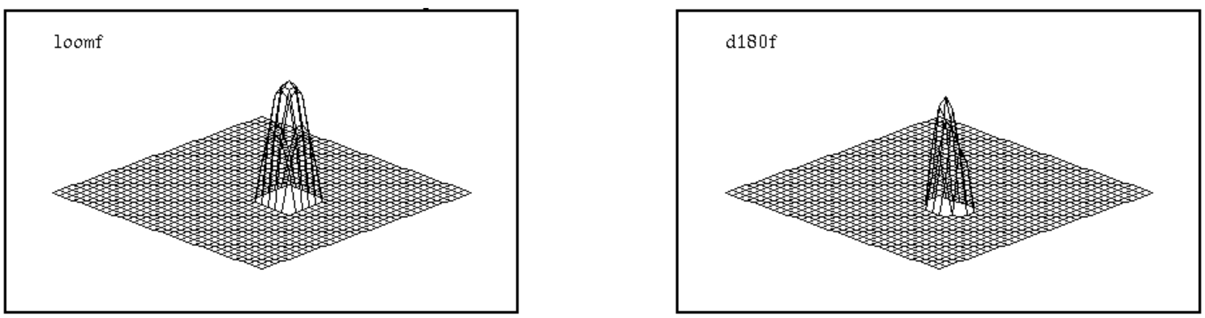


Figure 3. This is the collision layer from experiment #1. It indicates the presence of an object moving directly towards Meno. Similar activities also exist in the **loomf** and **d180f** layers that illustrate the moving object is expanding (d180f) while looming (loomf).



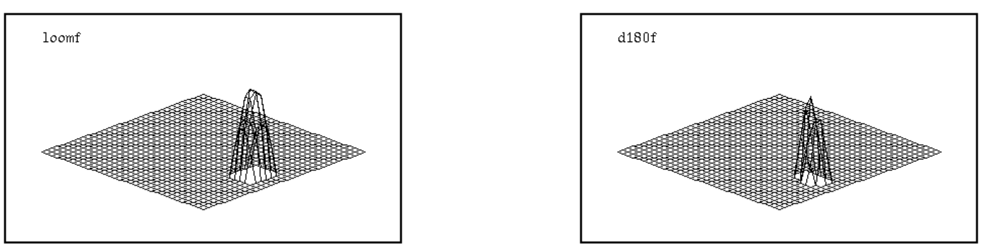
(a) (b)

Figure 4. In experiment #2, we have an object moving across the visual field from right to left. This motion corresponds to the activation of neuron firings in the **d180f** layer that is sensitive to movements in the right-to-left direction. However, we observe that the object does not loom towards the robot which explains the inactivity in the **loomf** layer.



(a) (b)

Figure 5. (a) This is the neural layer indicating the presence of a looming stimulus with an object approaching from the left as described in experimental setup #3. (b) This figure depicts the motion sensitive layer which is directionally selective to motions that is 180˚.



(a) (b)

Figure 6. In experiment #4, the looming stimulus approaching from above which accounts for the position of the peak neural firing towards the edge of these two figures. Nevertheless, these activities does not elicit a response in the collision layer.

References

Arbib, M. A., (1989) The Metaphorical Brain 2: Neural Networks and Beyond, Wiley-Interscience.

Brooks, R., (1986) "A Layered Intelligent Control System for a Mobile Robot," IEEE Journal of Robotics and Automation, **RA-2**:14-23, April.

Ewert, J.-P., (1984) "Tectal Mechanism that Underlies Prey-catching and Avoidance Behavior in Toads," in Comparative Neurology of the Optic Tectum (H. Vanegas, Ed.), Plenum Press.

Fagg, Andrew H., Lewis, M. Anthony, Iberall, Thea, and Bekey, George, (1991) "R^2AD : Rapid Robotics Application Development Environment," In the Proceedings of the IEEE Conference of Robotics and Automation, pp. 1420 - 1426

Kawato, M., (1990) "Computational Schemes and Neural Models for Formation and Control of Multijoint Arm Trajectories," in Neural Networks for Control (Miller, Sutton, and Werbos, Eds.), pp. 198 - 228, MIT.

Kuperstein, M., (1988) "Neural Model of Adaptive Hand-Eye Coordination for Single Postures," Science **239**:1308-1311.

Lewis, M. Anthony, Fagg, Andrew H., Solidum, Alan, (1992) "A Genetic Approach to the Development of a Neural Network for the Control of a Walking Machine," to appear in the Proceedings of the IEEE Conference of Robotics and Automation.

Liaw, J.-S., and Arbib, M.A., (1990) A Neural Network Model for Response to Looming Objects by Frog and Toad, in Visual Structures and Integrated Functions, (M.A. Arbib and J.-P. Ewert, Eds.), *Research Notes in Neural Computing*, Springer-Verlag.

Weitzenfeld, A., (1991), *NSL: Neural Simulation Language, Version 2.1*, CNE-TR 91-05, University of Southern California, Center for Neural Engineering.