Teach a Robot to Fish... A Thesis Proposal

Matthew J. Marjanović

November 30, 2000

Abstract

This proposal outlines the design of an extensible adaptive control system which will enable a humanoid robot to learn models of causality/causation through interaction with its environment. Using models of sensory-motor interactions as primitives, the system is able to learn progressively more abstract concepts. Unlike prior projects with similar philosophies and goals, this work is a component of a real-time control system implemented on a physical robot. Indeed, one crucial feature is that the system is driven by the effects of real physics on the robot, not just arbitrary rules in a physical simulation.

1 A Philosophy of Interaction

This thesis is an outgrowth of the idea that *perception is meaningless without action*. The value of what we see and hear and feel comes from what we can do about it. The semantic content of a sensory experience is grounded in an organism's ability to affect its environment and in its need to decide what effect to produce.

A disembodied perceptual system cannot assign any intrinsic value to the information it processes. When a face detection algorithm draws boxes around faces in a scene displayed on a computer screen, the entirety of the *meaning* of those boxes arises out of their observation by a human, to whom a "face" has meaning because it is attached to a large repertoire of social cues, interactions, desires, and memories.

In very simple terms, the brain is an intricate, stateful mechanism designed to "figure out what to do next".[1] The layers upon layers of interwoven concepts

[1] Coincidentally, Rod Brooks once wrote a working paper titled "Planning is just a way to avoid figuring out what to do next"[1]. Could we *both* be wrong? Very doubtful...

comprising intelligence are rooted in primitives that correspond to simple, direct interactions between motor systems and sensory systems, coupled either internally or through the environment. It is the interactions between these systems, and the patterns discovered among these interactions, which are the basis of thought.

Experience and Metaphor These ideas were inspired by (and espoused in) the works of Mark Johnson [2] and George Lakoff [3]. Their philosophical discourses argue that one cannot take for granted an "objective truth" which is independent of the physicality of the observer or thinker. They also show cross-cultural psycholinguistic evidence which demonstrates that our thoughts are rooted in physical interaction with the world.

We think in metaphors borrowed from physical processes. When we discuss increasing quantities, they inevitably move "up", even if the quantity has no physical realization.

In Our Own Image... These were some of the original philosophies behind the Cog Project. We have been designing a robot with human-like form, predicated on the notion that achieving human-like intelligence ultimately requires human-like experience.

My goal for this thesis is to design a system, running on a real anthropomorphic robot, that can generate/discover/invent those primitives which form the substrate upon which more complex reasoning can be built. In addition to creating primitives, the system must have methods for composing primitives — i.e., abstracting complex concepts from simple ones. These will give the robot a way of discovering and representing knowledge about itself and the world in terms of the sensory input it receives and the motor output it can produce.

I do not expect to have a complete walking, talking, thinking machine at the end of the next 12 months. However, I do intend to produce a system which has discovered some basic sensory-motor relationships and utilizes them to accomplish a complex task (Section 4). I plan to devise a solid methodology and functioning software architecture, which could then be extended and elaborated.

The remainder of this proposal delves into the nitty-gritty of the tasks required to approach these goals. Section 2 gives the details of the proposed implementation. Section 3 compares this project to other works with similar implementation or similar goals. Section 4 describes a complete proof-of-concept task, "Reach Out and Grab Objects" (ROGO), which shall be implemented to demonstrate that this system actually works. Finally, Section 5 outlines the schedule by which all this will come together over the next year.

2 Implementation

This project is divided into three parts.

The first part is a data-driven modular programming architecture. This is sok, which will be the bricks and mortar of the rest of the work. sok is already operational and mostly complete.

The second part is a motor control scheme in the form of a simulated musculature, called meso. This is work in progress (being developed jointly with another graduate student, Bryan Adams).

The third part is pamet, a self-sustaining system for modeling, arbitration, and encapsulation. pamet will be composed of processes which model sensory and motor interactions, and other processes which figure out how to use that knowledge to create behavior. This is currently theory on the verge of implementation.

2.1 sok: an Architecture for Behavior-Based Control

sok is a simple yet rich API for designing behavior-based control systems. sok shares many of the same goals as Brooks' original *Behavior Language* (BL) [4], and it is the evolutionary successor to InterProcess Socks (IPS) [5] and MARS [6], which had been used in earlier work on Cog. Unlike its Lisp-based ancestors, sok is implemented as a C library and API, and it allows computation to be distributed throughout a multiprocessing, multiprocessor network running the QNX operating system.[2] sok provides a real-time, dynamic environment for data-driven programming, which is essential to realizing the goals of this project.

The fundamental unit in the sok paradigm is a *sok process* (Figure 1), which can be considered a typical Unix-like process executing in its own memory space, with some additional built-in communication and control features. A sok process exchanges data with peer sok processes via *inports* and *outports*. Typically, a sok process responds to received data on its inports, performs some calculation, and then sends messages with the results via its outports. The intent is that

[2] sok is built on top of the message-passing features of the QNX operating system, however it could probably be ported to another OS given the appropriate communication layers.

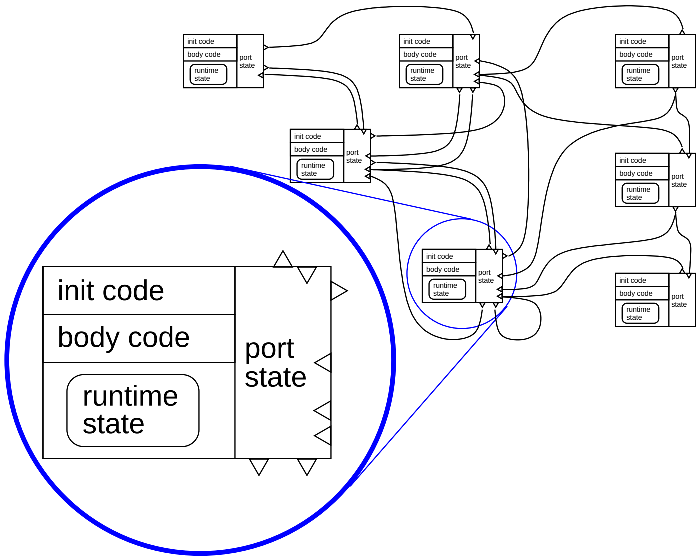

Figure 1: A network of *sok processes,* connected via *inports* and *outports.* The inset highlights the structure of a sok process with a number of ports. Multiple incoming and outgoing connections are allowed. The body code runs asynchronous to and independent of the message passing.

each sok process encapsulates some behavioral primitive such as “visual motion detection” or “arm motor interface”. Such primitives are coded independently of each other; yet by connecting ports, they are glued together to form a complete control system.

The connections of a sok process are independent of the process execution; messages are sent and received asynchronously. Furthermore, a sok process can be suspended, or even killed and restarted, without affecting the state of its connections. This is useful for graceful crash recovery, preserving system state, and testing by “lesioning” the system.

sok incorporates many essential features not found in its predecessors. Unlike BL, connections between modules are not statically compiled into the system. Connections are created or broken at run-time, by scripts, by the programmer, or by the modules themselves. Although its catalog of ports is fixed once registered as a sok process, an executable can gracefully remove itself from sok space and reconfigure itself — even spawning clones of itself. Thus one seed process can give rise to an entire interconnected network of processes.

Each sok process has a unique name in a hierarchical namespace managed by a locator daemon. This namespace is a tree structure which can be searched by any process. The connections between sok processes can also be queried, allowing the communication network to be traversed as well. Due to the underlying QNX OS, sok modules can be distributed over a multiprocessor network. Although processes must be explicitly invoked on a particular node, communication with other modules across the network is completely transparent.

A crucial feature for doing research is that a properly written sok process can be suspended and have its essential state dumped to disk (connections included). The entire universe of sok processes can in fact be suspended and dumped at once, and then restored to a saved state later on.

2.2 meso: a Simulated Musculature

Cog is an anthropomorphic robot: one fundamental principle of its mechanical design is that it should have enough degrees of freedom and articulation to enable it to recognizably emulate human motor behavior. This is a hardware requirement: Cog can't pretend to move an arm it doesn't have. The basic requirements for the control system are not so stringent, though. A very elaborate animatronic motor controller could produce very life-like canned motion, albeit with a very unanthropomorphic control system.

Cog was not built to produce canned animations; the goal of the project is to explore mechanisms for generating and learning social behavior. If Cog emulates the human form at the right level of granularity, these mechanisms should have a very natural form. With that in mind, Cog's actuators should incorporate human-like control as well as kinematics.

Given the mechanics of the existing robot, the best way to do this is to implement the thinnest possible abstraction layer which provides an effective *simulation* of human musculature. This is meso, which will simulate a motor system at the spinal level, incorporating essential features for human-like behavior and response: reflex stiffness, polyarticulate coupling, and pain and fatigue models. This minimal set should allow production of human-like movement which the control system can tune and optimize by human-like feedback.

Cog's actuators are already force-controlled [7], which is muscle-like in itself. meso will add a layer on top of this which will simulate "muscle groups" which approximate the function of antagonistic pairs of muscles in the human body. Figure 2 illustrates one proposal for allocating muscle groups to one of Cog's arms.

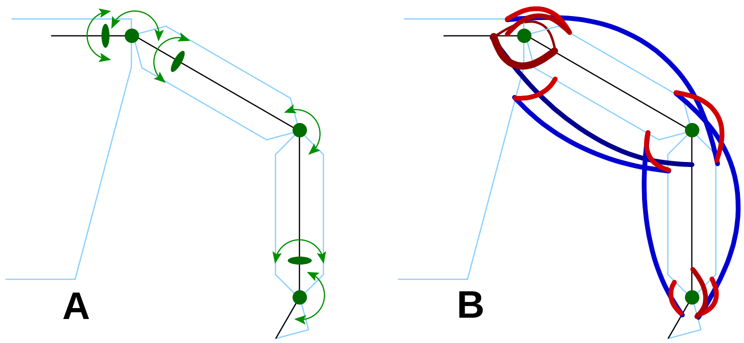

Figure 2: Cog's arms have six single-axis actuators (A); however, they will be controlled as if they were actuated by antagonistic pairs of real muscles (B). On Cog, most muscles will be polyarticulate. For example, any muscle spanning a shoulder must actually span three revolute joints (versus a single ball-and-socket joint in the human shoulder).

Spinal Reflexes Human muscle tissue is mechanically very different from electric motors, even force-controlled motors. However, an accurate simulation of muscle tissue is not necessarily useful. The brain's motor cortex does not directly activate muscles; the cortex connects to spinal motor neurons which control the muscles in conjunction with spinal reflex loops. Using input from stress and strain sensors in the muscles and tendons, these reflexes make antagonistic pairs of muscles act like simple damped, linear springs over a wide range of motion.

meso will incorporate this subcortical machinery into its simulation, creating abstract linearized "muscle groups" instead of individual muscles. Simulating the mechanics of raw muscle tissue is a waste of computation, since the body itself seems designed to try to hide such intricacies from the brain.

Polyarticulate Muscles Many muscles in the human body span more than one joint. For example, the biceps and triceps each span both the elbow and shoulder joints. Such muscles are *kinematically* redundant, because the same movements could be produced with muscles spanning only one joint. However, polyarticulate muscles do have at least two important effects on limb *dynamics*.

First, a multijoint arm actuated by single-joint linear springs will not have isotropic stiffness in endpoint coordinates [8]. The hand will react with varying stiffness when pushed in different directions, and the stiffness profile will be a function of limb configuration. Polyarticulate muscles add a tunable interjoint

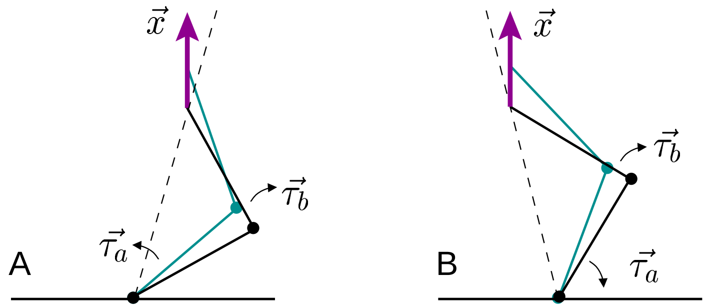

Figure 3: The illustrations depict a simple two segment arm with monoarticulate muscles at each joint. The initial position of the arm is the black configuration. (a) To do work along $\vec{x}$ — that is, to apply a force along that vector — the arm must move into the blue configuration. In doing so, the two joints apply torques $\vec{\tau}_a$ and $\vec{\tau}_b$ in the same directions as they are displaced, thus both contributing to the output work. (b) In a different configuration which does the same overall work, the lower joint must apply a torque in opposition to its displacement. This joint is *absorbing* energy (to be dissipated as heat), and that energy must ultimately be provided by the other joint.

coupling, which allows the endpoint stiffness to be controlled and stabilized over a wide-range of the workspace, independent of the limb configuration.

Second, polyarticulate muscles can make the musculoskeletal system more efficient [9, pp.298–303]. Applying force in certain configurations of a multijoint limb results in some muscles undergoing a "lengthening contraction" (Figure 3). That is, the muscle applies a force while being stretched, doing negative work. This energy is wasted as heat, and other muscles must provide that work, which is never seen at the output of the limb. In these cases, a stiffened biarticulate muscle can provide a mechanical linkage which produces the same motion, but does not waste the work.

Cog has no real polyarticulate actuators (each joint is driven by a single motor), and a simulation of such won't make the robot any more energy efficient. But, the simulation should make human-like motion appear more optimal to the controller than motion which looks (for lack of a better term) "robotic".

Fatigue and Pain Cog's motors have completely alien fatigue characteristics compared to human muscle. Given an unquenched power supply from the national power grid, the motors can apply a wide range of forces indefinitely, as long as they don't overheat. Human muscles get tired much more quickly, and this affects not only how much they are used, but also *how* they are used.

Constraining the motors to operate under a muscular fatigue model should encourage the development of both human-like movement and behavior. The model should account for basic metabolic processes in muscle [10, ch. 6], including a reservoir of energy for penalty-free short-term activity, and the build-up of slow-to-dissipate byproducts from unsustainably intense activity. The "fatigue level" should affect motor performance, and should also be data accessible directly by the control system. The model should be tunable, to simulate different stages of growth and ability. These aspects of fatigue have been recently implemented on Cog by Bryan Adams as a system based on models of human biochemistry [11].

Pain is another component of this model. Fatigue causes physical discomfort, as does over-exertion or joint over-extension. Such pain is a cue to perform an activity differently (if at all), and thus is an important component of an anthropomorphic robot which learns by doing.

2.3 pamet: Arbitration, Modeling, and Encapsulation

The focus of this project is pamet, a self-supervised system which discovers and models causal relations between sensory and motor systems on the robot. Such relations are practical representations of knowledge about the robot, its interactions with the world, and the world at large. pamet needs to do three things: learn models of correlated events, determine how to arbitrate between conflicting sources of information or intent, and aggregate and encapsulate knowledge into a manipulatable form. pamet will be a recursive system, capable of discovering relations between its own modules as well as between the sensory and motor primitives which are hard-wired into the robot.

Modules and Messages pamet is built on top of sok; its universe consists entirely of control modules (sok processes) which communicate by sending or receiving messages along unilateral connections. Some modules are considered "hard-wired" or static — these are the modules which interface directly to sensory or motor hardware. They comprise the *primitives* in the system, and are coded and tuned to achieve a particular function. The rest of the modules are "soft-wired" or dynamic. They are not wired into the system by the hand of the programmer, but rather they connect themselves via programmatic con-

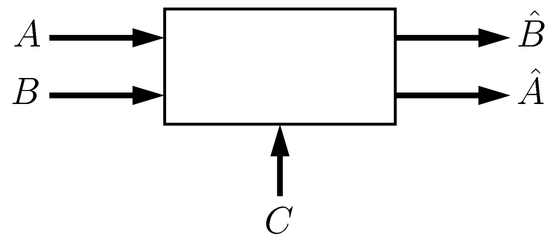

Figure 4: The basic modeller template. A, B, and C are inputs; $\hat{A}$ and $\hat{B}$ are outputs. The module learns a relation between A and B, which is reliable under certain inputs on C. In response to input to A, the trained modeller produces an output on $\hat{B}$ which predicts the expected input to B. Likewise, input to B will produce an output from $\hat{A}$ which should correlate with A. The model is symmetric in that sense.

straints. Dynamic modules can replicate, mutate, or vanish. The complexity of the system ultimately rests in the networks developed by the dynamic modules.

The messages passed between pamet modules have three primary characteristics. The first and most obvious is *value*, i.e., the data payload, which may be a scalar, vector, image array, bit string, etc. The second is *reliability*, which is a literal real value (e.g., $r \in [0.0, 1.0]$) transmitted with the data which provides a measure of how reliable the data is, as determined by the process which produced it. The third characteristic is the *rate*, which is simply the number of messages per second sent along a connection; this is not a literal value but rather a measurable quantity. The rate reflects the level of activation of the module producing the messages.

Three Families Dynamic modules are generated by pamet, and can be divided into three basic types: modellers, arbitrators, and encapsulators. Modellers discover and model correlations of activity between other modules, both static and dynamic. Arbitrators act as complex switchboxes, directing data streams and controlling access to motor and sensory resources. Encapsulators serve to dereference connections so that a subnetwork of modules can be assessed as a single module. In short, modellers store knowledge, arbitrators direct control, and encapsulators produce abstraction.

To date, only the modellers have been thought out well enough to begin implementing them. I plan to focus my efforts on these modules while leaving room in the system for the others to be developed later.

Modellers Every modelling module has the same basic interface, shown in Figure 4. A and B are two sets of inputs, representing two parameter spaces to be correlated. C is another set of inputs which provide additional context under which the module is activated. $\hat{B}$ and $\hat{A}$ are outputs, which give the predictions of the model in response to inputs to A or B, respectively. Aside from this basic template, a modeller is a black box.

Different species of modeller module could learn different types of models; I plan to focus on a zeroth-order relation $f : A \to B$ with a time delay Δt. That is, B takes on the value $f(A)$ after a delay Δt, with no other time dependencies. Both f and Δt are learned. In a loose sense, such a model represents the knowledge that "A causes B". Specific examples of such a model are given in Section 4.

The modeller template is symmetric by design. A modeller observes its inputs and tries to learn some correlation of B with A. Once trained, an input to A will produce an output prediction $\hat{B}$. However, an input to B should also produce a prediction $\hat{A}$. In the case of the simple causal model above, $\hat{A}$ represents "the activity which is expected to yield B". This symmetry requires that a model be implemented so as to simultaneously learn its inverse.

The models will likely be implemented as piecewise-linear function approximations. [12] and [13] outline potentially fruitful ways to learn such models in high-dimensional parameter spaces. This could be combined with one of two well-known techniques for simultaneously learning a forward and inverse model, described by [14] and [15].

The type of model determines what type of knowledge a modeller can encode. The simple predictive model above learns that activity in one module elicits activity in another. A more sophisticated modeller would include dynamics and account for a more complex interaction. Yet another model could be designed to represent persistent objects.

Growth and Networking Learning by a modelling module is on-line and automatic. It has no distinct training phase; it adapts continuously as it receives input. Instantiation of a modelling module must also be automatic — a key goal of pamet is that features and correlations within the system are discovered by the system itself, not enumerated by an engineer's hand. This is the only way to achieve learning which transcends simple adaptation.

As a sok process, a single instance of a modeller can be designed to replicate and spawn copies which attach to other modules in a search for correlations. It can search through the sok process space in two different ways: by looking at the namespace for modules tagged with similar (or even disparate) functionality by the programmer; or, by traversing connections and using the network topology as a guide.

I envision this as a dynamic process: new modellers of various species are spawned continuously to try to find new relations in the system; modules fork to try alternative hypotheses; unreliable, unsuccessful modules give up and die after some amount of time. Obviously this is not going to be a simple or quick process — it will require a lot of data online, a lot of time, and a fair bit of tuning. For this reason, I plan to initially focus on the simple predictive model outlined above, to see what it can accomplish and to see how the system must be tuned to make it work. A sample task utilizing such models is described in Section 4.

Using the Knowledge Once pamet acquires a reliable model, it still needs to *use* it. Models can be used in two modes: forward, to predict an outcome, or reverse, to generate an outcome. A predictive model of a feedback circuit can be used for feedforward control. A reverse model of a sensory-motor circuit can be used to generate motor behavior.

The job of connecting models back to action-producing systems falls to the arbitrator modules, which follow strategies to experiment with different connection schemes. Like models, the utility of a connection scheme needs to be evaluated as the system develops.

Modellers observe activity within Cog, to produce models, and the models are interconnected via arbitrators to yield more complex forms of activity. Conceptually this is a spiral, expanding outward as it grows in breadth and complexity (Figure 5). But, what drives the initial activity? What drives the interconnection?

Cog needs a source of intention to provoke action. At its root, this should be provided by primal drives engineered into the system. Early activity should be driven by *reflexes*, which are inhibited as more elaborate model-driven controllers develop. Early arbitration is driven by *emotion,* which is also inhibited as more elaborate evaluation schemes develop. Reflexes are simple motor responses elucidated by simple sensory inputs. Emotions are simple decisions based on hard-wired performance cues. For example, a vision module which detects smiles could provide a global reinforcement signal — when a smile is detected, the robot is assured that *some* behavior is being performed correctly.

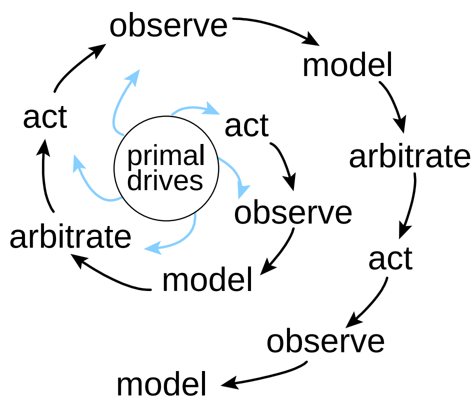

Figure 5: Behavior and learning form a spiral of action and observation which continually grows in complexity. The spiral is rooted and driven by *primal drives* — reflexes and emotions which are hard-wired into a system. As the system develops, the primal drives become more distant, however they remain active at the core.

Some fairly elaborate emotional models have already been developed by coworkers [16]. A simpler adaptation of one of these should suffice to achieve my goals.

Encapsulation and Abstraction I haven't though about these too much yet. The overall idea is to have a mechanism which allows groups of pamet modules to be treated as a single module, with regards to interconnection and replication. This would allow the functionality produced by the group to be used atomically elsewhere. This is the essence of abstraction. Unfortunately, it is difficult to develop the idea further until the modules themselves have a more tangible form.

3 Other Work in the Field

The basic ideas of connecting perception and action, of learning by exploring the environment, and of learning transforms between different parameter spaces are not brand-new by any means. But, when my scheme actually makes a physically-realized, life-sized robot do something, I think we will all be pretty impressed.

This proposal picks up on the themes of my master's thesis [17] and of some of my earlier work on Cog [18], significantly extending these works. It shares many high-level goals with Drescher's schema system [19], although the mechanisms are very different. Finally, it also bears similar intent to a number of other adaptive control systems, such as DRAMA [20], which I will briefly discuss.

3.1 Earlier Endeavors

This entire endeavor originated with my master's thesis [17]. Quoting from Chapter 1:

> From a more pragmatic perspective, as Cog becomes more developed, it will have more and more specialized computational systems which will need to cooperate with each other. Models which translate the actions of one system into the native parameters of another will facilitate such cooperation. Once the number of subsystems on Cog exceeds a handful, an infrastructure which automatically generates such adaptive models will make the entire programming experience much more manageable.

My master's thesis project was a system that learned the relationship between head movement and consequent motion in Cog's visual field. The motion detector, trying to run at real-time frame rates on a slow processor, had a very limited dynamic range. By calculating the effect of head movement on perceived motion, and feeding that back into the motion detector, the range of the detector could be extended.

The system is outlined in Figure 6. The modeller, labelled as "compensation network" would observe neck motor and motion detector activity, and try to produce a signal which reduced detected motion to zero. In lieu of any other behaviors to drive activity, the head was programmed to move randomly.

This project was a simple first step toward a system which learned models between sensory and motor systems. It contained only one model, however, and that model was completely hard-wired.

Another project which continued along these lines was "visually-guided pointing", in which Cog learned to reach out and point at visual targets [18]. The overall network for this task is illustrated in Figure 7. This project actually coupled several independent systems: motor-only reach generator, motion detection, saccade learning and generation, object segmentation, and reach learning.

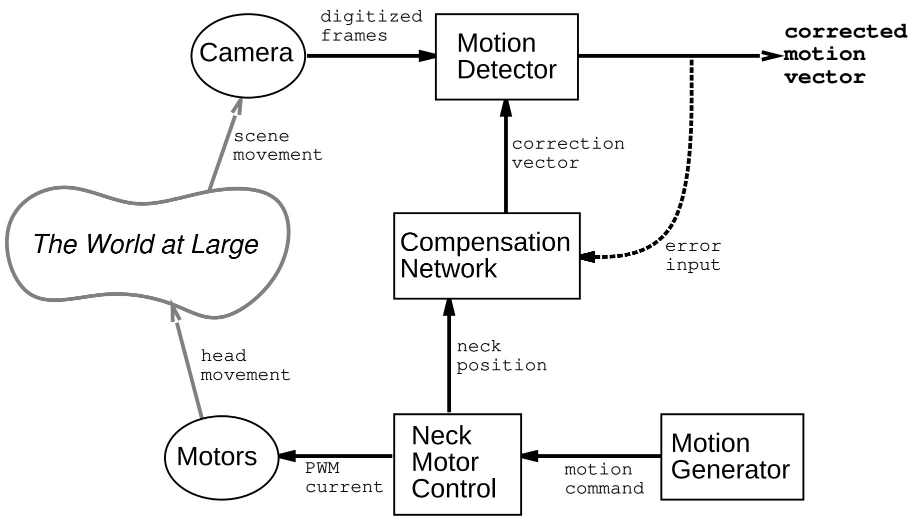

Figure 6: This compensation network [17] learned a model of head movement in terms of its effect on perceived visual motion. This knowledge was used to make the motion detector more robust.

For the pointing task, learning proceeded in two stages, with the second bootstrapped by the first. The robot learned how to generate saccades — that is, how to move the eye to foveate upon a stimulus somewhere in the visual field. This required learning a map between the retinal coordinate frame and the eye-motor frame. In the second stage, Cog learned to extend its arm to point at what it was looking at. This required learning a map from the eye-motor frame to an arm-motor coordinate frame. However, since the feedback was *visual*, training required the previously acquired saccade map to perform a retinal-to-eye-motor translation.

This pointing task is similar to the proposed ROGO task which I will discuss in Section 4. However, like my master's thesis, the pointing task network was completely hard-wired. Furthermore, the pointing operated over a very limited range of the arm workspace, without any head motion (eyes only). The grand task proposed for this thesis will overcome these limitations.

3.2 Drescher's Schema System

Drescher [19] presents a learning system which attempts to avoid the grounding problems of symbolic AI by learning in terms of relations between sensory and motor primitives in a simulated "robot" which lives in a grid-world. His *schema*

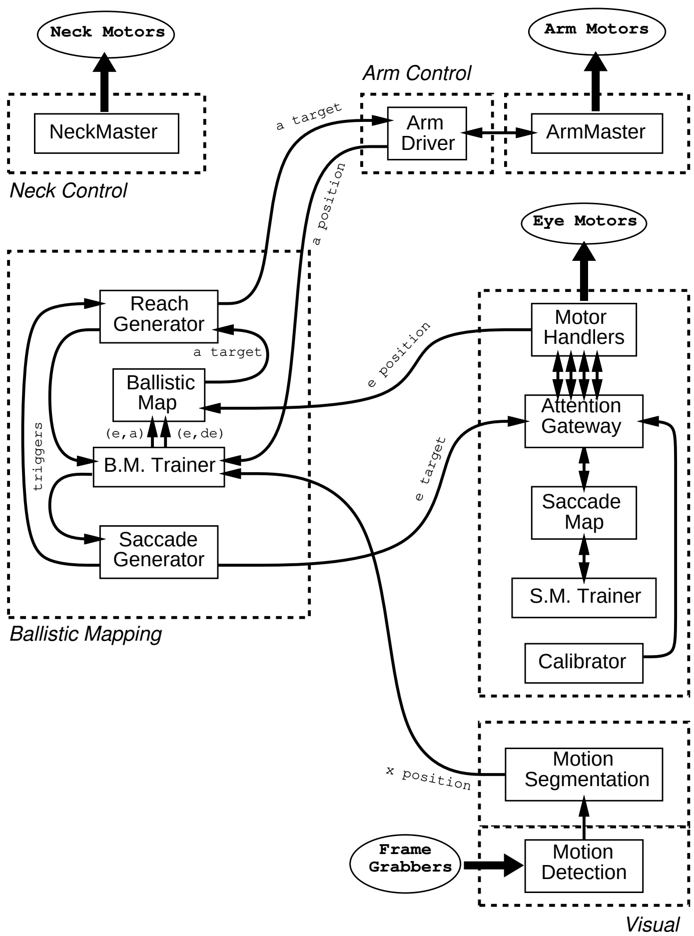

Figure 7: The visually-guided pointing task [18] involved several interconnected subsystems. Two models were learned, saccade map and ballistic map, and the first was used to facilitate training the second.

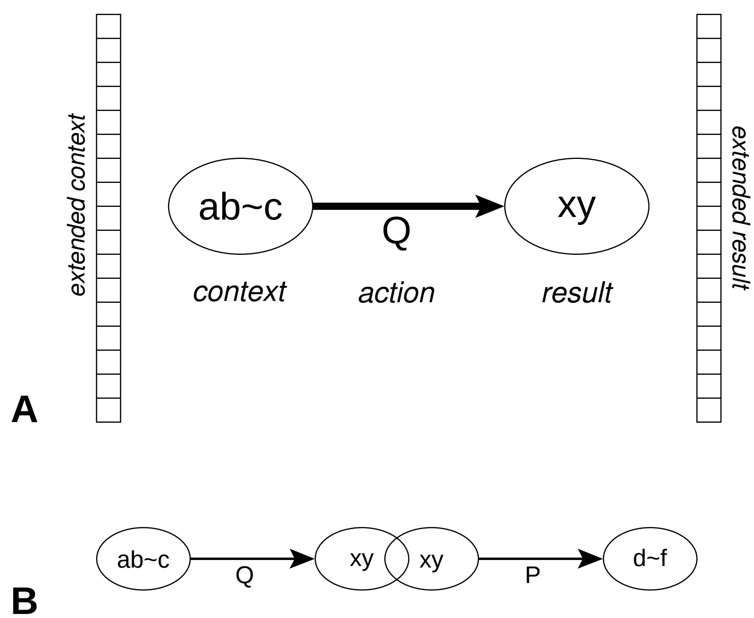

Figure 8: Drescher's schema system [19]: A schema (A) is a rule which specifies the result state of some number of binary items ("x" and "y") if an action ("Q") is performed while the system's initial state satisfies some context ("a", "b", and "not c"). A schema maintains extended context and extended result vectors as well, which track all state items and are used to decided to "spin-off" a more specific schema. Composite actions are instantiated as chains of schemas (B) with compatible result and context clauses.

system comprises three basic entities: items, actions, and schemas (Figure 8). These entities have primitive forms, which are built into the system at the outset, as well as synthetic or composite forms, which are generated by learning. Items are binary (on or off) state elements — primitive items are tied directly to sensory input. Primitive actions represent basic motor activity. Schemas are rules which specify what happens to certain items (the result) if an action is carried out while other items are in a prescribed state (the context). They are not plans, but descriptions of cause and effect.

The system learns by experimenting: an action is taken, and the new item states are checked to see what effect the action had. The system maintains statistics on the results, and new schemas with revised contexts or results are periodically "spun-off". If a new schema is deemed more reliable than its predecessor, it is favored during action selection.

Composite actions can be created, which refer to chains of schemas (in which the result of the first satisfies the context of the next, and so on). Also, synthetic items can be created. Each synthetic item is tied to a base schema; a synthetic item represents the conditions that make its basic schema reliable. That is to say, when the item is true, the basis schema's action will reliably lead to its result.

These two methods for abstraction give the schema system a way to represent concepts beyond raw sensor and motor activity. Drescher gives the example that a schema that says "moving to X,Y results in a touch sensation" effectively defines the item of state "tactile object at position X,Y". For that schema to be reliable, that bit of knowledge must be true — so in the schema system, that schema *is* that knowledge.

Limitations Drescher's system incorporates many important ideas. In the end, however, it is just a symbolic AI engine and thus suffers from a couple of fundamental flaws.

First and foremost, the system operates in a toy grid-world with a small number of discrete features. Sensor and motor activity is represented by a small repertoire of symbolic binary primitives. Furthermore, state is perfect; there is no noise and no errors in determining the values of primitive items. Real robots and real physical processes are not so simple and rarely so forgiving — at best, real sensors report streams of values which are ambiguous in isolation.

The schema system also has no notion of time. Actions are completely serialized — carried out one-at-a-time, never overlapping or simultaneous. State updates instantly, an affect of the quantized grid world. The lack of time and the lack of any material physics in the grid world reduces the system to a purely symbolic exercise.

State and actions are distinct entities, at least at the beginning. Symbolic items are actually directly linked to actions (via the basis schema), which is in fact necessary if the system is to develop percepts beyond its hand-wired primitives. pamet dispenses with the labels "state" and "action" altogether. There are merely messages being passed around, and the only concrete action occurs at the physical outputs of the actuators.

3.3 DRAMA and other control systems

Billard's DRAMA [20] is representative of connectionist control systems which attempt to achieve similar goal as pamet and share similar limitations as the schema system. DRAMA contains a fully-connected network, including self-connections. Sensor input and actuator feedback are connected to the network via "event detectors" which only propagate $0 \rightarrow 1$ transitions.

The network acts like an associative memory with a Hebbian learning rule. The self-connections provide an element of short-term memory, so the network can learn to correlate inputs with a time delay.

Limitations DRAMA is used in real-time to control some simple wheeled robots. One of its selling points is that it is simple enough to run quickly on low-power processors. And it does have an inherent sense of time.

DRAMA relies on binary states, and that states need to be disjunctive. If a compass sensor can report one of eight directions, it must be represented by eight bits.

The number of correlations that can be learned by DRAMA is proportional to the number of units in the network, which is limited by the number of sensor/actuator bits. There is no mechanism form abstraction, nor a way to learn anything more complicated than simple sequences.

4 A Concrete Task

The only way that we can see if the ideas behind pamet hold water is to use them to produce a visible, active behavior in the robot. The demonstrable goal of this project is for Cog to learn to "Reach Out and Grab Objects" (ROGO). This is easier said than done. The task is simple to state and recognize, however it decomposes into several non-trivial subtasks. pamet will be crucial to implement the subtasks as well as to connect them together to achieve the overall task. ROGO is a behavior more than a task, in the sense that it is something which the robot will learn to do in response to environmental cues, as prompted by a simple motivational system.

I have broken ROGO down into four sub-behaviors which I explain in further detail in the following sections. The behaviors stand alone to a degree, although they reinforce each other. Each subtask (and the complete ROGO as well) could be hard-wired, as opposed to discovered by pamet. This provides a way of assessing the performance of pamet, by comparing to the hard-wired task.

Note that hard-wiring does not forego adaptation and learning — it means that the module connections are wired by hand to achieve the behavior, rather than being discovered after some amount of machine exploration. The modellers still do their modelling. Keep in mind that the goal of pamet is to allow these tasks and behaviors to emerge and develop as a result of the dynamics of the robot and its interaction with the people and objects around it.

Table 1 summarizes the goals and prerequisites of each subtask. Each requires several built-in faculties — modules which are hard-wired into the robot. These consist of basic perceptual modules, reflexes, and emotional drives. The subtasks are mostly independent and could be developed in any order, except that "hand segmentation" relies on transforms learned by "head-eye coordination".

4.1 Grasping

The grasping subtask consists of the rudimentary action of closing the hand upon an object at an appropriate time. This is a deceptively simple operation: even if the mechanical act of grasping were as simple as flipping a bit in a control register, that bit would have no *a priori* "meaning" to the robot. The meaning is as essential as the mechanical act itself, as it is the meaning which will eventually drive the robot to *try* to grasp something.

To bootstrap the mechanics, Cog will have a built-in grasp reflex: similar to a young infant [21], a reflex module will cause the hand to close whenever tactile sensors detect pressure on the inside surface. A pamet modeller observing this process could correlate the hand motor commands with the continuing and increased tactile stimulation.

Subtask	Goal	Prerequisite
Grasping	simple grasp of an object already in hand	clench reflex muscle fatigue tactile sense tactile reinforcer inhibition mechanism
Visual Targeting	direct gaze towards people bearing gifts	saccades face/person detection motion detection social reinforcer smooth pursuit
Head-Eye Coordination	turn head in concert with shifting gaze, learn eye$\leftrightarrow$head map	VOR/OKR eye muscle fatigue stereopsis saccades
Hand Segmentation	detect motion of hand/arm in visual field, learn correspondence of motor command with visual position	muscle model motion detection stereopsis Head-Eye Coord.

Table 1: A breakdown of ROGO into subtasks: each subtask builds on top of a number of primitive motor or perceptual systems already built into the robot.

This model needs to be coupled with a reinforcer which responds to tactile stimulation; the robot must be hard-wired to some degree "to like to touch things". This desire for novel tactile stimulation will be one primal drive behind the entire ROGO behavior. Another key ingredient is an inhibition mechanism. As the grasp concept takes form, the modeller which instantiates it must be able to inhibit the grasp reflex. Once the reflex is completely subsumed, one could say that grasping is under "conscious" control.

Figure 9 outlines the role of a modeller in the grasping subtask. The grasp model will become further refined once reaching occurs — grasping will be triggered at the right moment at the end of a reach.

4.2 Visual Targeting

ROGO is a motor behavior which is triggered by visual events: the sight of a person proffering a toy should instigate reaching. The visual targeting subtask is the precursor to reaching: recognizing these visual cues and directing gaze towards a target object.

Visual targeting is constructed on top of a number of built-in faculties: mo-

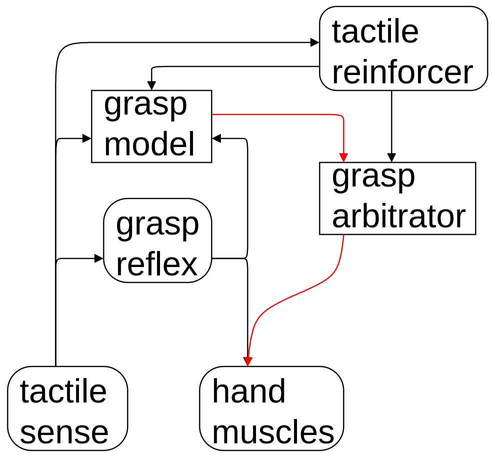

Figure 9: The basic network for the grasping task: the model learns that a grasping motion can create a desirable tactile response; the arbitrator eventually allows the model to override the activity of the reflex. The red pathway is active after learning.

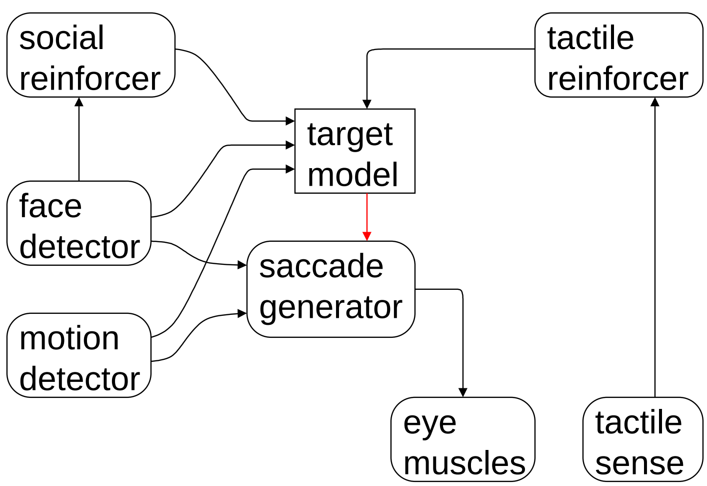

Figure 10: Stimuli from the face and motion detectors reflexively drive Cog to look around, via the saccade generator. The target model learns to discriminate stimuli that lead to favorable tactile stimulation, and eventually the model overrides the random choices of the saccade generator.

tion detection, person/face detection, and saccades (Figure 10). The reflexive part of the behavior is to randomly saccade to foveate upon people or moving things. As people play with Cog and wave objects around in front of it, Cog should learn that moving objects associated with people yield a better chance of tactile sensation. Cog should then preferentially focus upon such objects.

Note that this subtask requires Cog to live in a visually "busy" environment; without visual stimulation, the robot will not learn anything.

4.3 Head-Eye Coordination

Head-eye coordination involves learning the relation between head movement and eye movement with regards to the visual scene, and turning the head in concert with the eyes during large gaze shifts. This subtask relies on the built-in vestibular-ocular reflex (VOR) and optokinetic response (OKR), as well as a sense of eye muscle fatigue.

When the head moves, the scene perceived by the eyes shifts. VOR and OKR are adaptive biological stabilization mechanisms which counter-rotate the eyes when the head moves, so that gaze remains fixed upon a target. This stabilization involves implicit knowledge of the eye-head kinematics and the transform between the two coordinate frames.

The subtask here is to learn this transform by observing the VOR and OKR so that the robot can perform the converse behavior: when gaze shifts by a large amount, the head should also turn in advance of the eyes, such that the eyes are not driven to the extremes of their range of motion.

Figure 11 illustrates this system. In addition to built-in VOR, OKR, and saccade generation, there is also a random head movement generator and two pamet modellers. The first modeller learns the correlations between head motor commands and the compensatory eye motor commands generated by stabilization. Note that this modeller is not simply learning what the VOR and OKR modules already know. These reflexes generate eye motor commands in response to vestibular and optical input; the modeller is learning a secondary relation between those eye motor commands and head motor drives.

Since moving the head can reduce (or worsen) eye fatigue, the second modeller discovers the correlation between eye position, eye muscle fatigue, and head movement. It learns to perform feedforward control of the head — turning in advance of the eyes when a long-distance saccade is activated, so that the eyes don't actually get driven to their limits of motion.

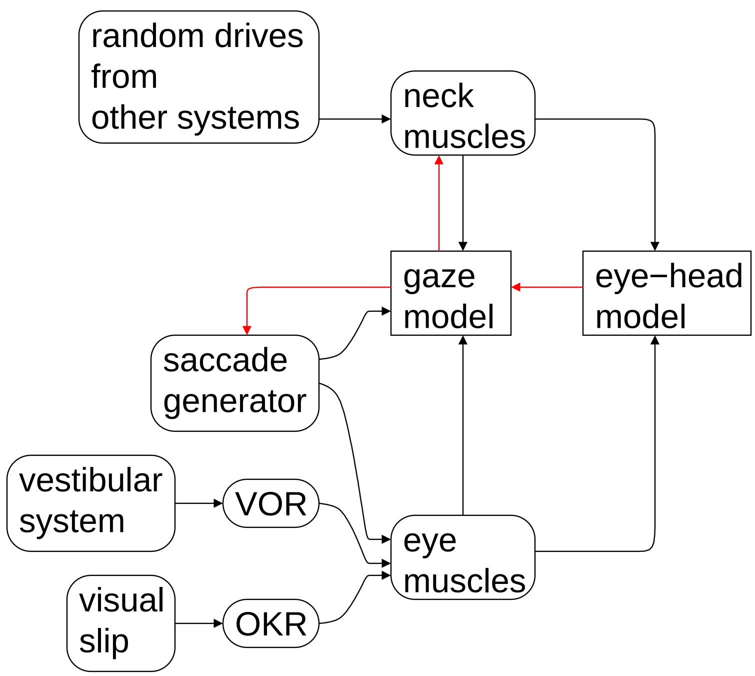

Figure 11: The eye-head model learns the relation between eye and head coordinate frames, by observing the action of the VOR and OKR during head movements (generated by other systems, perhaps even just “random flailing”). The gaze model observes saccade generation, head movement, and eye fatigue, and learns to turn the head in concert with a long-distance saccade.

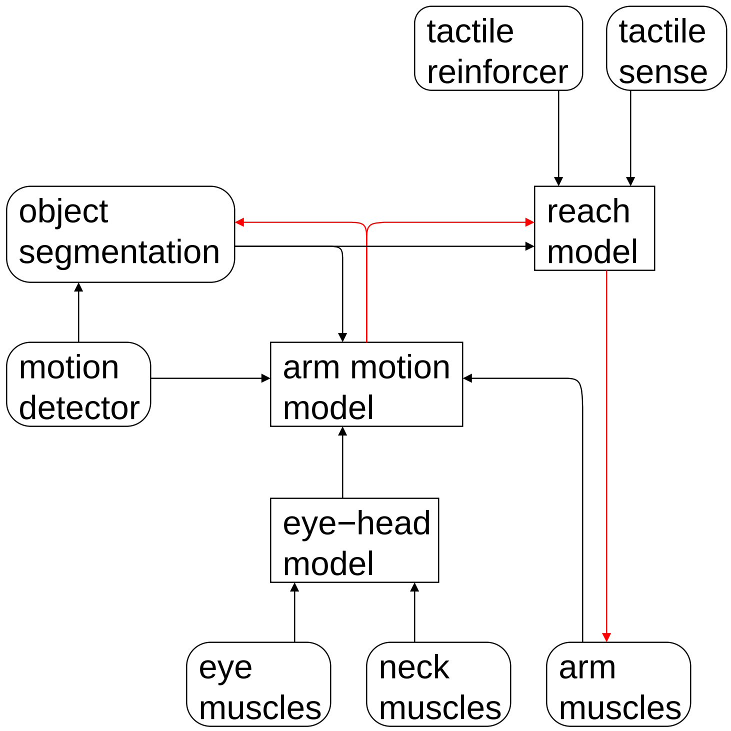

Figure 12: The arm motion model correlates motion detection and object cues with motor commands to the arm, acquiring the knowledge that the arm commands cause motion. This knowledge can be fed back into the object segmentation module to help pull out the arm. The reach model correlates arm motion near other objects with pleasing tactile sensation. The model can be inverted to generate a reach towards an object.

4.4 Hand Segmentation

Even though sophisticated motion detection and object segmentation will allow Cog to "see" its hands (as moving objects), and a simulated musculature allows Cog to move its hands, those two faculties alone do not give Cog the knowledge that it "has" hands. The goal of this subtask is to model the correlation between arm/hand motor commands and the resulting visual motion. This model encapsulates knowledge of what the hands are in terms of what they can do and how they can be made to do it.

Figure 12 outlines this subsystem; the modeller observes visual stimuli and motor commands and finds the correlation. This actually involves the head-eye coordination maps as well, which correct for different head orientations with respect to the arms.

A second goal is to learn that seeing the hand touch an object results in a (good) tactile sensation. This leads to the reaching behavior: upon seeing an object, use of the inverse visuomotor model will move the hand to touch that object.

5 Timetable

Table 2 outlines the schedule I plan to follow to get accomplish these goals. sok is mostly finished already; it is functional, but the remaining features will make it more effective. meso — the muscle model — will be implemented and functional by the mid-autumn, 2000 (probably before the physical arms are operational). In the meantime, I will start toying with and simulating modelling techniques, so that pamet and its tasks can be implemented once motor control is in place.

References

[1] Rodney A. Brooks. Planning is just a way of avoiding figuring out what to do next. MIT AI Lab Working Paper 303, September 1987.

[2] Mark Johnson. *The Body in the Mind.* University of Chicago Press, 1987.

[3] George Lakoff. *Women, Fire, and Dangerous Things.* University of Chicago Press, 1987.

[4] Rodney A. Brooks. The behavior language user's guide. A.I. Technical Report 1227, Massachusetts Institute of Technology AI Lab, Cambridge, MA, April 1990.

[5] Rodney Brooks, Johanna Bryson, Matthew Marjanović, Lynn Stein, and Mike Wessler. Humanoid software manual. MIT AI Lab internal document, 1996.

[6] Rodney Brooks. L. IS Robotics Internal Technical Report, 1996.

[7] Matthew M. Williamson. Series elastic actuators. A.I. Technical Report 1524, Massachusetts Institute of Technology AI Lab, Cambridge, MA, January 1995.

Dates	sok	meso	pamet	thesis
September, 2000	basic communication complete implement dump/restore	arms move spring law		
October, 2000	finish up simple type manager	pseudo-muscle model torso moves		
November, 2000	implement input arbitration	fatigue model	implement modellers	
December, 2000			"visual targeting"	
January, 2001			"head-eye coordination"	
February, 2001			"hand segmentation"	
March, 2001			"grasping"	start writing
April, 2001			complete "ROGO"	first draft done
May, 2001				defend turn-in thesis
June, 2001				graduate!

Table 2: Schedule for implementation of the components of this thesis, including ROGO subtasks and thesis preparation.

[8] Neville Hogan. The mechanics of multi-joint posture and movement control. *Biological Cybernetics*, 52:315–331, 1985.

[9] Stan Gielen. Muscle activation patterns and joint-angle coordination in multijoint movements. In Alain Berthoz, editor, *Multisensory Control of Movement*, chapter 19, pages 293–313. Oxford University Press, 1993.

[10] Thomas A. McMahon. *Muscles, Reflexes, and Locomotion*. Princeton University Press, 1984.

[11] Bryan P. Adams. Meso: A virtual musculature for humanoid motor control. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, September 2000.

[12] Stefan Schaal, Christopher G. Atkeson, and Sethu Vijayakumar. Real-time robot learning with locally weighted statistical learning. In *International Conference on Robotics and Automation*, San Francisco, April 2000.

[13] Sethu Vijayakumar and Stefan Schaal. Fast and efficient incremental learning for high-dimensional movement systems. In *International Conference on Robotics and Automation*, San Francisco, April 2000.

[14] M. I. Jordan and D. E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16:307–354, 1992.

[15] M. Kawato. Feedback-error-learning neural network for supervised learning. In R. Eckmiller, editor, *Advanced Neural Computers*, pages 365–372. North-Holland, Amsterdam, 1990.

[16] Cynthia Breazeal and Juan Velasquez. Toward teaching a robot 'infant' using emotive communication acts. In *Proceedings of 1998 Simulation of Adaptive Behavior, Workshop on Socially Situated Intelligence*, pages 25–40, Zurich, Switzerland, 1998.

[17] Matthew J. Marjanović. Learning functional maps between sensorimotor systems on a humanoid robot. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1995.

[18] Matthew Marjanović, Brian Scassellati, and Matthew Williamson. Self-taught visually-guided pointing for a humanoid robot. In Pattie Maes, Maja J Matarić, et al., editors, *From animals to animats*, number 4, pages

35–44, North Falmouth, MA, September 1996. Fourth International Conference on Simulation of Adaptive Behavior (SAB96), MIT Press, Cambridge, MA.

[19] Gary L. Drescher. *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. MIT Press, Cambridge, MA, 1991.

[20] Aude Billard and Gillian Hayes. DRAMA, a connectionist architecture for control and learning in autonomous robots. *Adaptive Behavior*, 7(1):35–63, 1999.

[21] Adele Diamond. Developmental time course in human infants and infant monkeys, and the neural bases of inhibitory control in reaching. In *Development and Neural Bases of Higher Cognitive Functions*, volume 608, pages 637–676. New York Academy of Sciences, 1990.