Self-Taught Visually-Guided Pointing for a Humanoid Robot

Matthew Marjanović, Brian Scassellati, Matthew Williamson*
545 Technology Square
Room NE43-920
Cambridge, MA 02139
maddog@ai.mit.edu, scaz@ai.mit.edu, matt@ai.mit.edu

Abstract

The authors implemented a system which performs a fundamental visuomotor coordination task on the humanoid robot Cog. Cog's task was to saccade its pair of two degree-of-freedom eyes to foveate on a target, and then to maneuver its six degree-of-freedom compliant arm to point at that target. This task requires systems for learning to saccade to visual targets, generating smooth arm trajectories, locating the arm in the visual field, and learning the map between gaze direction and correct pointing configuration of the arm. All learning was self-supervised solely by visual feedback. The task was accomplished by many parallel processes running on a seven processor, extensible architecture, MIMD computer.

1 Introduction

This paper is one of a series of developmental snapshots from the Cog Project at the MIT Artificial Intelligence Laboratory. Cog is a humanoid robot designed to explore a wide variety of problems in artificial intelligence and cognitive science (Brooks & Stein 1994). To date our hardware systems include a ten degree-of-freedom upper-torso robot, a multiprocessor MIMD computer, a video capture/display system, a six degree-of-freedom series-elastic actuated arm, and a host of programming language and support tools (Brooks 1996, Brooks, Bryson, Marjanovic, Stein, & Wessler 1996). This paper focuses on a behavioral system that learns to coordinate visual information with motor commands in order to learn to point the arm toward a visual target. Related work on Cog is also being presented at this conference, see (Ferrell 1996, Williamson 1996). Additional information on the project background can be found in (Brooks & Stein 1994, Irie 1995, Marjanovic 1995, Matsuoka 1995, Pratt & Williamson 1995, Scassellati 1995).

Given the location of an interesting visual stimulus in the image plane, the task is to move the eyes to foveate on that stimulus and then move the arm to point to that visual location. We chose this task for four reasons: First, the task is a fundamental component of more complex tasks, such as grasping an object, shaking hands, or playing "hide-and-seek" with small toys. Second, reaching to a visually stimulating object is a skill that children develop at a very early age (before the 5th month), and the development of this skill is itself an active area of research (Diamond 1990). Third, the task specification can be reformulated as a variety of behavioral responses. The task can be viewed as a pointing behavior (to show the location of a desired object), a reaching behavior (to move the arm to a position where the hand can begin to grasp an object), a protective reflex (to move the arm to intercept a dangerous object), or even as an occlusion task (to move the arm to block out bright lights or to hide an object from sight like the children's game "peek-a-boo"). Finally, the task requires integration at multiple levels in our robotic system.

To achieve visually-guided pointing, we construct a system that first learns the mapping from camera image coordinates $\vec{x} = (x, y)$ to the head-centered coordinates of the eye motors $\vec{e} = (pan, tilt)$ and then to the coordinates of the arm motors $\vec{\alpha} = (\alpha_0 ... \alpha_5)$. An image correlation algorithm constructs

*The authors receive support from a National Science Foundation Graduate Fellowship, a National Defense Science and Engineering Graduate Fellowship, and JPL Contract # 959333, respectively. Support for the Cog project is provided by an ONR/ARPA Vision MURI Grant (No. N00014-95-1-0600), a National Science Foundation Young Investigator Award (No. IRI–9357761) to Professor Lynn Andrea Stein, and by the J.H. and E.V. Wade Fund. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

a saccade map $\vec{S} : \vec{x} \rightarrow \vec{e}$, which relates positions in the camera image with the motor commands necessary to foveate the eye at that location. Our task then becomes to learn the ballistic movement mapping from head-centered coordinates $\vec{e}$ to arm-centered coordinates $\vec{\alpha}$. To simplify the dimensionality problems involved in controlling a six degree-of-freedom arm, arm positions are specified as a linear combination of basis posture primitives. The ballistic mapping $\vec{B} : \vec{e} \rightarrow \vec{\alpha}$ is constructed by an on-line learning algorithm that compares motor command signals with visual motion feedback clues to localize the arm in visual space.

The next section describes the hardware of Cog's visual system, the physical design of the arm, and the computational capabilities of Cog. Section 3 gives a functional overview of the parallel processes that cooperate to achieve the pointing task. Section 4 describes details of the visual system: how the saccade map is learned and how the end of the arm is located in the visual field. Section 5 details the decomposition of arm movements into a set of linearly separable basic postures, and the learning algorithms for the ballistic map are explained in Section 6. Preliminary results of this learning algorithm and continuing research efforts can be found in Section 7.

2 Robot Platform

This section gives a brief specification of the physical subsystems of Cog (see Figures 1 and 2) that are directly relevant to our pointing task. We will describe the visual inputs that are available, the design and physical characteristics of the arm, and the processing capabilities of Cog's "brain". We have compressed much detail on the Cog architecture into this section for those readers interested in observing the progress of the project as a whole. Readers interested only in the pointing task presented here may omit many of these details.

2.1 Camera System

To approximate human eye movements, the camera system has four degrees-of-freedom consisting of two active "eyes" (Ballard 1989). Each eye can rotate about a vertical axis (pan) and a horizontal axis (tilt). Each eye consists of two black and white CCD cameras, one with a wide peripheral field of view $(88.6°(V) \times 115.8°(H))$ and the other with a narrow foveal view $(18.4°(V) \times 24.4°(H))$. Our initial experiments with the pointing task have used only the wide-angle cameras.

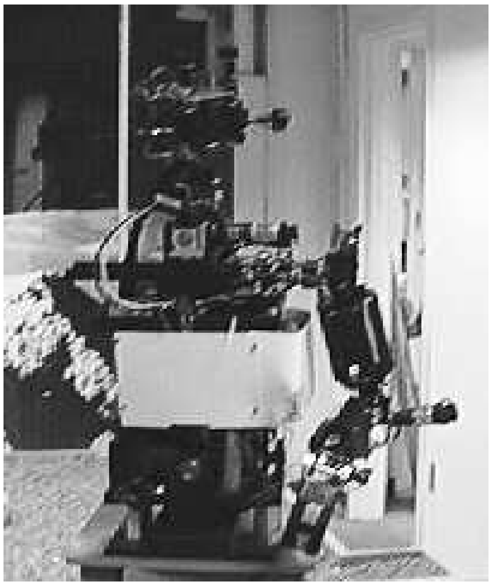

Figure 1: Cog, an upper-torso humanoid robot. Cog has two degrees-of-freedom in the waist, one in the shoulder, three in the neck, six on the arm, and two for each eye.

The analog NTSC output from each camera is digitized by a custom frame grabber designed by one of the authors. The frame grabbers subsample and filter the camera signals to produce 120×120 images in 8-bit grayscale, which are written at a frame rate of 30 frames per second to up to six dual-ported RAM (DPRAM) connections. Each DPRAM connection can be linked to a processor node or to a custom video display board. The video display board reads images simultaneously from three DPRAM slots and produces standard NTSC output, which can then be routed to one of twenty video displays.

2.2 Arm Design

The arm is loosely based on the dimensions of a human arm, and is illustrated in Figure 1. It has 6 degrees-of-freedom, each powered by a DC electric motor through a series spring (a series elastic actuator, see (Pratt & Williamson 1995)). The spring provides accurate torque feedback at each joint, and protects the motor gearbox from shock loads. A low gain position control loop is implemented so that each joint acts as if it were a virtual spring with variable stiffness, damping and equilibrium position. These spring parameters can be changed, both to move the arm and to alter its dynamic behavior. Motion of the arm is achieved by changing the equilibrium positions of the joints, not by commanding the

Figure 2: Supporting structure for Cog. The "brain" of the robot is a MIMD computer which occupies the racks in the center of this image. Video from the cameras or from the brain is displayed on a bank of twenty displays shown on the left. User interface and file storage are provided by a Macintosh Quadra. Cog itself is on the far right.

joint angles directly. There is considerable biological evidence for this spring-like property of arms (Zajac 1989, Cannon & Zahalak 1982, MacKay, Crammond, Kwan & Murphy 1986).

The spring-like property gives the arm a sensible "natural" behavior: if it is disturbed, or hits an obstacle, the arm simply deflects out of the way. The disturbance is absorbed by the compliant characteristics of the system, and needs no explicit sensing or computation. The system also has a low frequency characteristic (large masses and soft springs) which allows for smooth arm motion at a slower command rate. This allows more time for computation, and makes possible the use of control systems with substantial delay (a condition akin to biological systems). The spring-like behavior also guarantees a stable system if the joint set-points are fed-forward to the arm.

2.3 Computational System

The computational control for Cog is split into two levels: an on-board local motor controller for each motor, and a scalable MIMD computer that serves as Cog's "brain." This division of labor allows for an extensible and modular computer while still providing for rapid, local motor control.

Each motor has its own dedicated local motor controller, a special purpose board with a Motorola 6811HC11E2 microcontroller, which reads the encoder, performs servo calculations, and drives the motor with a 32KHz pulse-width modulated signal. For the eyes, the microcontroller implements a PID control law for position and velocity control, which is optimized for saccadic movement. For the arms, the microcontroller generates a virtual spring behavior at 1kHz. Similar motor control boards, with device-specific control programs, are used for body and neck motors.

Cog's "brain" is a scalable MIMD computer consisting of up to 239 processor nodes (although only eight are in use so far). During operation, the brain is a fixed topology network. However, the topology can be changed and scaled by adding additional nodes and connections. All components of the processing system communicate through 8K by 16 bit DPRAM connections, so altering the topology is relatively simple. Each node uses a standard Motorola serial peripheral interface (SPI) to communicate sensory information and control loop parameters with up to eight motor control boards at 50Hz. Each processor node contains its own 16MHz Motorola 68332 microprocessor mounted on a custom-built carrier board that provides support for the SPI communications and eight DPRAM connections. A Macintosh Quadra is used as the front-end processor for the user interface and file service (but *not* for any computation). Communication between the Quadra and the nodes of the MIMD computer is handled by a custom-built packet multiplexer box.

Each processor runs its own image of L, a compact, downwardly compatible version of Common Lisp that supports multi-tasking and multi-processing (Brooks 1996); and each uses IPS, a front end to L that supports communication between multiple processes (Brooks et al. 1996).

3 Task Overview

Figure 3 shows a schematic representation of the system architecture, at the process and processor level. The system can be decomposed into three major pieces, each developed semi-independently: visual, arm motor, and a ballistic map. The visual system is responsible for moving the eyes, detecting motion, and finding the end of the arm. The arm motor system maintains the variable-compliance arm and generates smooth trajectories between endpoints specified in a space of basis arm postures. The ballistic mapping system learns a feed-forward map from gaze position to arm position and generates reaching commands. Each of these subsystems is described in greater detail below.

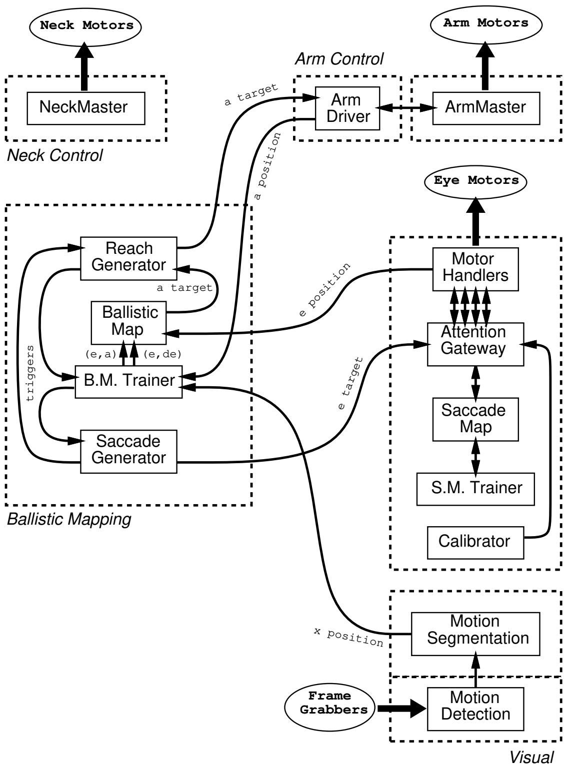

Figure 3: Schematic representation of the system architecture. Solid boxes are processes, dashed boxes indicate processor nodes. Messages pass between processors via dual-ported RAM connections. Image coordinates are represented by $\vec{x}$ positions, head-centered coordinates are represented by pan and tilt encoder readings $\vec{e}$, and arm positions are represented as linear combinations of the basis postures $\vec{\alpha}$.

For this first large-scale integration task implemented on Cog, we strove to meet a number of constraints, some self-imposed and some imposed by the hardware capabilities. The software architecture had to be distributed at both the processor and the process level. No single processor node had enough power to handle all the computation, nor enough peripheral control ports to handle the eleven motors involved. Within each processor, the system was implemented as collections of functionally independent but interacting processes. In the future we hope to implement more refined and elaborate behaviors by adding new processes to the existing network.

Although the basic activity for this particular task is sequential — foveate, reach, train, repeat — there is no centralized scheduler process. Rather, the action is driven by a set of triggers passed from one process to another. This is not a very important design consideration with the single task in mind; however as we add more processes, which act in parallel and compete for motor and sensor resources, a distributed system of activation and arbitration will become a necessity.

4 Visual System

The components of the visual system used in this task can be grouped into four functional units: basic eye-motor control, a saccade map trainer, a motion detection module, and a motion segmentation module. The eye-motor control processes maintain communication with the local motor control boards, initiate calibration routines, and arbitrate between requests for eye movement. The saccade trainer incrementally learns the mapping between the location of salient stimuli in the visual image with the eye motor commands necessary to foveate on that object. The motion detection system uses local area differences between successive camera images to identify areas where motion has occured. The output from the motion detection system is then grouped, segmented, and rated to determine the largest contiguous moving object. This segmented output is then combined with arm motor feedback by the ballistic map trainer (see Section 6) to locate the endpoint of the moving arm.

4.1 Eye Motor Control

The basic eye-motor control software is organized into a two-layer structure. In the lower layer, there is one process, called a handler, which maintains a continuous communication between the processor node and the local motor control board. In the upper layer is a single attentional gateway process which ensures that only one external process has control over the eyes at any given time. Currently, as soon as calibration has finished, the attentional gateway cedes control of the eye-motors to the ballistic map trainer. As more procedures begin to rely on eye movement, the attentional gateway will arbitrate between requests. Similar structures are used for the neck and arm motors, but do not appear in the Figure 3.

4.2 Learning the Saccade Map

In order to use visual information as an error signal for arm movements, it is necessary to learn the mapping between coordinates in the image plane and coordinates based on the body position of the robot. With the neck in a fixed position, this task simplifies to learning the mapping between image coordinates and the pan/tilt encoder coordinates of the eye motors. The behavioral correlate of this simplified task is to learn the pan and tilt positions necessary to saccade to a visual target. Initial experimentation revealed that for the wide-angle cameras, this saccade map is linear near the image center but rapidly diverged near the edges. An on-line learning algorithm was implemented to incrementally update an initial estimate of the saccade map by comparing image correlations in a local field. This learning process, the saccade map trainer, optimized a look-up table that contained the pan and tilt encoder offsets needed to saccade to a given image coordinate.

Saccade map training began with a linear estimate based on the range of the encoder limits (determined during calibration). For each learning trial, the saccade map trainer generated a random visual target location (x_t, y_t) and recorded the normalized image intensities $\bar{I}_t$ in a 16×16 patch around that point. The process then issued a saccade motor command using the current map entries. After the saccade, a new image $\bar{I}_n$ is acquired. The normalized 16×16 center of the new image is then correlated against the target image. Thus, for offsets x_0 and y_0, we sought to maximize the dot-product of the image vectors:

$$\max_{x_0, y_0} \left(\sum_i \sum_j \bar{I}_t(i,j) \cdot \bar{I}_n(x_0 + i, y_0 + j) \right) \qquad (1)$$

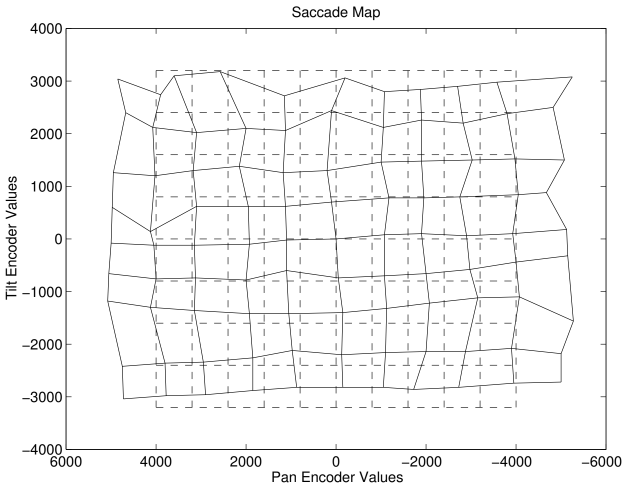

Figure 4: Saccade Map after 0 (dashed lines) and 2000 (solid lines) learning trials. The figure shows the pan and tilt encoder values for every tenth position in the image array within the ranges x=[10,110] (pan) and y=[20,100] (tilt).

Since each image was normalized, maximizing the dot product of the image vectors is identical to minimizing the angle between the two vectors. This normalization also gives the algorithm a better resistance to changes in background luminance as the camera moves. In our experiments, the offsets x_0 and y_0 had a range of $[-2, 2]$. The offset pair that maximized the expression in Equation 1, scaled by a constant factor, was used as the error vector for training the saccade map.

Note that a single learning step of this hill-climbing algorithm does not find the optimal correlations across the entire image. The limited search radius vastly increases the speed of each learning trial at the expense of producing difficulties with local maxima. However, in the laboratory space that makes up Cog's visual world, there are many large objects that are constant over relatively large pixel areas. The hill-climbing algorithm effectively exploited this property of the environment to avoid local maxima.

To simplify the learning process, we initially trained the map with random visual positions (x_t, y_t) that were multiples of ten in the ranges $[10, 110]$ for x_t (the pan dimension) and $[20, 100]$ for y_t (tilt). By examining only a subset of the image points, we could quickly train a limited set of points which would bootstrap additional points. Examining image points closer to the periphery was also unnecessary since the field of view of the camera was greater

Figure 6: Expanded example of the visual learning of the saccade map. The center collage is the pre-saccade target images $\bar{I}_t$ for a subset of the entire saccade map. The left collage shows the post-saccade image centers with no learning. The right collage shows the post-saccade image centers after 2000 learning trials. The post-learning collage shows a much better match to the target than the pre-learning collage.

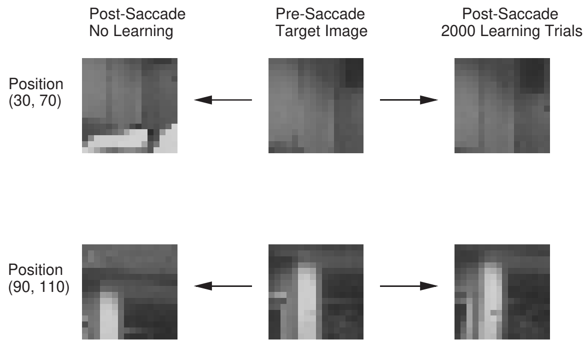

Figure 5: Two examples of the effects of the saccade map learning. The center set of images is the pre-saccade target image $\bar{I}_t$. The left image is the post-saccade image centers with no learning. The right image is the post-saccade image centers after 2000 learning trials. The post-learning images match the target more closely than the pre-learning images.

than the range of the motors; thus there were points on the edges of the image that could be seen but could not be foveated regardless of the current eye position. Figure 4 shows the data points in their initial linear approximation (dashed lines) and the resulting map after 2000 learning trials (solid lines). The saccade map after 2000 trials clearly indicates a slight counter-clockwise rotation of the mounting of the camera, which was verified by examination of the hardware. The training quickly reached a level of 1 pixel-error or less per trial within 2000 trials (approximately 20 trials per image location). Perhaps as a result of lens distortion effects, this error level remained constant regardless of continued learning.

Two examples of the visual effect of the learning procedure are shown in Figure 5. The center two images are the expected target images $\bar{I}_t$ recorded before the saccade for the image positions (30,70) and (90,110). Using the initial linear approximation with no learning, the post-saccade image $\bar{I}_n$ (shown at left) does not provide a good match to the target image (center). After 2000 learning trials, the difference in results is dramatic; the post-saccade image (shown to the right of the target) closely matches the pre-saccade target image. If the mapping had learned exactly the correct function, we would expect the pre-saccade and post-saccade images to be identical (modulo lens distortion). Visual comparison of the target images before saccade and the new images after saccade showed good match for all training image locations after 2000 trials. A larger set of examples from the collected data is shown in Figure 6.

4.3 Motion Detection and Segmentation

The motion detection and motion segmentation systems are used to provide visual feedback to the ballistic map trainer by locating the endpoint of the moving arm. The motion detection module computes the difference between consecutive wide-angle images within a local field. The motion segmenter then uses a region-growing technique to identify contiguous blocks of motion within the difference image. The bounding box of the largest motion block is then passed to the ballistic map trainer as a visual feedback signal for the location of the moving arm. In order to operate at speeds close to frame rate, the motion detection and segmentation routines were divided between two processors.

The motion detection process receives a digitized 120×120 image from the left wide-angle camera. Incoming images are stored in a ring of three frame buffers; one buffer holds the current image I_0, one buffer holds the previous image I_1, and a third buffer receives new input. The absolute value of the difference between the grayscale values in each image is thresholded to provide a raw motion image ($I_{raw} = \mathcal{T}(|I_0 - I_1|)$). The raw motion image is then used to produce a motion receptive field map, a 40×40 array in which each cell corresponds to the number of cells in a 3×3 receptive field of the raw motion image that are above threshold. This reduction in size allows for greater noise tolerance and increased processing speed.

The motion segmentation module takes the receptive field map from the motion detection processor and produces a bounding box for the largest contiguous motion group. The process scans the receptive field map marking all locations which pass threshold with an identifying tag. Locations inherit tags from adjacent locations through a region grow-and-merge procedure. Once all locations above threshold have been tagged, the tag that has been assigned to the most locations is declared the "winner". The bounding box of the winning tag is computed and sent to the ballistic map trainer.

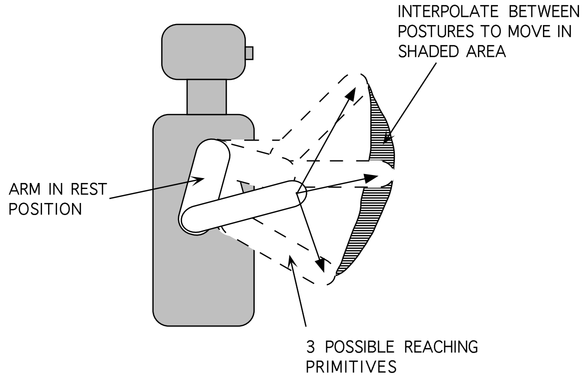

Figure 7: Primitives for the reaching task. There are four primitives: a rest position, and three in front of the robot. Linear interpolation is used to reach to points in the shaded area. See also Figure 8.

5 Arm Motion Control

5.1 Postural Primitives

The method used to control the arm takes inspiration from work on organization of movement in the spinal cord of frogs (Bizzi, Mussa-Ivaldi & Giszter 1991, Giszter, Mussa-Ivaldi & Bizzi 1993, Mussa-Ivaldi, Giszter & Bizzi 1994). These researchers electrically stimulated the spinal cord, and measured the forces at the foot, mapping out a force field in leg-motion space. They found that the force fields were convergent (the leg would move to fixed posture under the field's influence), and that there were only a small number of fields (4 in total). This lead to the suggestion that these postures were primitives that could be combined in different ways to generate movement (Mussa-Ivaldi & Giszter 1992). Details on the application of this research to robotic arms can be found in (Williamson 1996).

In Cog's arm the primitives are implemented as a set of equilibrium angles for each of the arm joints, as shown in Figure 7. Each primitive corresponds to a different posture of the arm. Four primitives are used: a rest position, and three on the extremes of the workspace in front of the robot. These are illustrated in Figure 8. Positions in space can be reached by interpolating between the primitives, giving a new set of equilibrium angles for the arm, and so a new end-point position. The interpolation is linear in primitive and joint space, but due to the non-linearity of the forward kinematics (end-point position in terms of joint angles), the motion in Cartesian space is not linear. However since only 4 primitives are used to move the 6 DOF arm, there is a large reduction in the dimensionality of the problem, with a consequent reduction in complexity.

There are some other advantages to using this primitive scheme. There is a reduction in communication bandwidth as the commands to the arm need only set the rest positions of the springs, and do not deal with the torques directly. In addition the motion is bounded by the convex hull of the primitives, which is useful if there are known obstacles to avoid (like the body of the robot!).

5.2 Reaching motion

The reaching behavior takes inspiration from studies of child development (Diamond 1990). Children always begin a reach from a rest position in front of their bodies. If they miss the target, they return to the rest position and try again. This reaching sequence is implemented in Cog's arm. Infants also have strong grasping and withdrawal reflexes, which help them interact with their environment at a young age. These reflexes have also been implemented on Cog (Williamson 1996).

The actual motion takes inspiration from observations of the smooth nature of human arm motions (Flash & Hogan 1985). To produce a movement, the joints of the arm are moved using a smooth, minimum jerk profile (Nelson 1983).

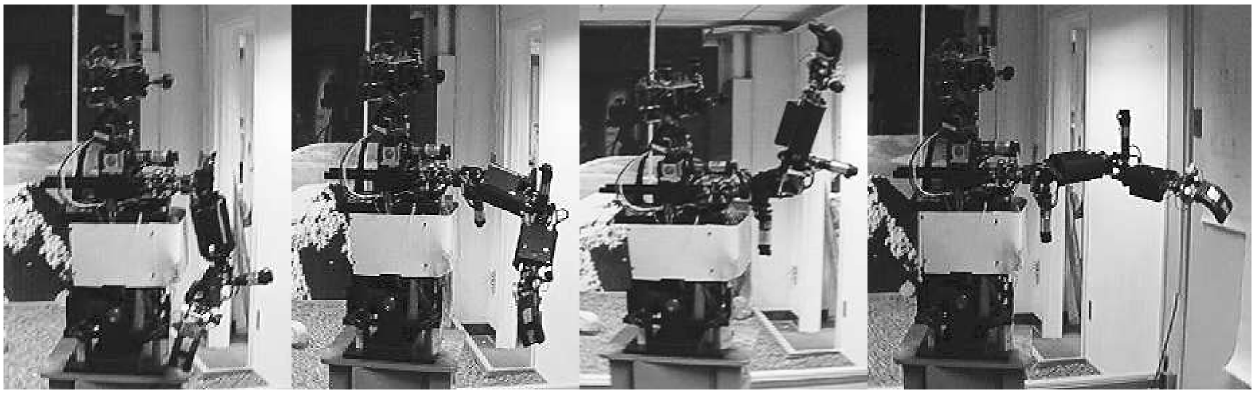

Figure 8: The basic arm postures. From left, "rest", "front", "up", and "side."

6 Ballistic Map

The ballistic map is a learned function $\vec{B}$ mapping eye position $\vec{e}$ into arm position $\vec{\alpha}$, such that the resulting arm configuration puts the end of the arm in the center of the visual field. Arm position is specified as a vector in a space of three basic 6-dimensional joint position vectors — the *reach* primitives (shown in Figure 8). There is also a fourth "rest" posture to which the arm returns between reaches.

The reach primitive coefficents are interpreted as percentages, and thus are required to sum to unity. This constrains the reach vectors to lie on a plane, and the arm endpoint to lie on a two-dimensional manifold. Thus, the ballistic map $\vec{B}$ is essentially a function $\mathcal{R}^2 \to \mathcal{R}^2$.

We attempted to select reach primitives such that the locus of arm endpoints was smooth and 1-to-1 when mapped onto the visual field. The kinematics of the arm and eye specify a function $\vec{E} : \vec{\alpha} \mapsto \vec{e}$ which maps primitive-specified arm positions into the eye positions which stare directly at the end of the arm. The ballistic map $\vec{B}$ is essentially the inverse of $\vec{E}$: we desire $\vec{E}(\vec{B}(\vec{e})) = \vec{e}$. If $\vec{E}$ is 1-to-1, then $\vec{B}$ is single-valued and we need not worry about learning discontinuous or multiple output ranges.

The learning techniques used here closely parallels the distal supervised learning approach (Jordan & Rumelhart 1992). We actually learned the forward map $\vec{E}$ as well as $\vec{B}$; this was necessitated by our training scheme. However, $\vec{E}$ is useful in that it gives an expectation of where to look to find the arm. This can be used to generate a window of attention to filter out distractions in the motion detection.

6.1 Map Implementation

The maps $\vec{B}$ and $\vec{E}$ are both implemented using a simple radial basis function approach. Each map consists of 64 Gaussian nodes distributed evenly over the input space. The nodes have identical variance, but are associated with different output vectors. The output of such a network ($\vec{y}$) for some input vector $\vec{i}$ is given by:

$$\vec{y} = \sum_k \vec{w}_k g_k(\vec{i}),$$

where

$$g_k(\vec{i}) = \exp(-\frac{1}{\sigma^2}\|\vec{i} - \vec{u}_k\|^2).$$

and $\vec{w}_k$ is a set of weights.

The ballistic map is initialized to point the arm to the center of the workspace for all gaze directions. The forward map is initialized to yield a centered gaze for all arm positions.

6.2 Learning the Ballistic Map

After the arm has reached out and its endpoint has been detected in the visual field, the ballistic map $\vec{B}$ is updated. However, since the error signal is a position in the image plane, the training cannot be done directly. We need to use the forward map $\vec{E}$ and the saccade map $\vec{S}$.

The current gaze direction $\vec{e}_0$ is fed through $\vec{B}$ to yield a reach vector $\vec{\beta}$ (β-space is a two dimensional parameterization of the α reach-primitive space). This $\vec{\beta}$ is sent to the arm to generate a reaching motion. It is also fed through the forward map $\vec{E}$ to generate an estimate $\vec{e}_p$ of where the arm will be in gaze-space after the reach. In an ideal world, $\vec{e}_p$ would equal $\vec{e}_0$.

After the arm has reached out, the motion detection determines the position $\vec{x}$ of the arm in pixel coordinates. If the reach were perfect, this would be the center of the image. Using the saccade map $\vec{S}$, we can map the difference in image (pixel) offsets between the end of the arm and the image center into gaze (eye position) offsets. So, we can use $\vec{S}$ to convert the visual position of the arm $\vec{x}$ into a gaze direction error $\Delta\vec{e}$.

We still cannot train $\vec{B}$ directly, since we have an e-space error but a β-space output. However, we can backpropagate $\Delta\vec{e}$ through the forward map $\vec{E}$ to yield a useful error term.

After all is said and done, we are performing basic least-mean-squares (LMS) gradient descent learning on the gaze error $\Delta\vec{e}$. For $\vec{B}$ defined by:

$$\vec{\beta} = \vec{B}(\vec{e}) = \sum_k \vec{w}_k g_k(\vec{e})$$

the update rule for the weights $\vec{w}_k$ is:

$$\Delta w_{ik} = -\eta \left(\Delta\vec{e} \cdot \frac{\partial \vec{F}}{\partial \beta_i} \right) g_k(\vec{e}).$$

for some learning rate η.

The forward map $\vec{F}$ is learned simultaneously with the ballistic map. Since $\vec{e} = \vec{e}_0 + \Delta\vec{e}$ is the gaze position of the arm after the reach, and $\vec{e}_p$ is the position predicted by $\vec{F}$, $\vec{F}$ can be trained directly via gradient descent using the error $(\vec{e}_p - \vec{e})$.

7 Results, Future Work, and Conclusions

At the immediate time of this writing, the complete system has been implemented and debugged, but has not been operational long enough to fully train the ballistic map. Initial results on small subsets of the visual input space show promising results. However, it will take some more extended training sessions before Cog has fully explored the space of reaches.

In addition to completing Cog's basic ballistic pointing training, our plans for upcoming endeavors include:

- incorporating additional degrees of freedom, such as neck and shoulder motion, into the model
- refining the arm finding process to track the arm during reaching
- expanding the number of primitive arm postures to cover a full three-dimensional workspace
- extracting depth information from camera vergence and stereopsis, and using that to implement reaching to and touching of objects.
- adding reflexive motions such as arm withdrawal and a looming response, including raising the arm to protect eyes and head
- making better use of the inverse ballistic map in reducing the amount of computation necessary to visually locate the arm.

This pointing task, albeit simple when viewed alongside the myriad complex motor skills of humans, is a milestone for Cog. This is the first task implemented on Cog which integrates major sensory and motor systems using a cohesive distributed network of processes on multiple processors. To the authors, this is a long-awaited proof of concept for the hardware and software which have been under development for the past two and a half years. Hopefully, this task will be a continuing part of the effort towards an artificial machine capable of human-like interaction with the world.

8 Acknowledgments

The authors wish to thank the members of the Cog group (past and present) for their continual support: Mike Binnard, Rod Brooks, Cynthia Ferrell, Robert Irie, Yoky Matsuoka, Nick Shectman, and Lynn Stein.

References

Ballard, D. (1989), 'Behavioral Constraints on Animate Vision', *Image and Vision Computing* **7:1**, 3–9.

Bizzi, E., Mussa-Ivaldi, F. A. & Giszter, S. F. (1991), 'Computations underlying the execution of movement: A biological perspective', *Science* **253**, 287–291.

Brooks, R. (1996), L, Technical report, IS Robotics Internal Document.

Brooks, R. & Stein, L. A. (1994), 'Building Brains for Bodies', *Autonomous Robots* **1:1**, 7–25.

Brooks, R., Bryson, J., Marjanovic, M., Stein, L. A., & Wessler, M. (1996), Humanoid Software, Technical report, MIT Artificial Intelligence Lab Internal Document.

Cannon, S. & Zahalak, G. I. (1982), 'The mechanical behavior of active human skeletal muscle in small oscillations', *Journal of Biomechanics* **15**, 111–121.

Diamond, A. (1990), *Development and Neural Bases of Higher Cognitive Functions*, Vol. 608, New York Academy of Sciences, chapter Developmental Time Course in Human Infants and Infant Monkeys, and the Neural Bases, of Inhibitory Control in Reaching, pp. 637–676.

Ferrell, C. (1996), Orientation Behavior Using Registered Topographic Maps, Society of Adaptive Behavior. In these proceedings.

Flash, T. & Hogan, N. (1985), 'The Coordination of Arm Movements:An Experimentally Confirmed Mathematical Model', *Journal of Neuroscience* **5**(7), 1688–1703.

Giszter, S. F., Mussa-Ivaldi, F. A. & Bizzi, E. (1993), 'Convergent Force Fields Organized in the Frog's Spinal Cord', *Journal of Neuroscience* **13**(2), 467–491.

Irie, R. (1995), Robust Sound Localization: An Application of an Auditory Perception System for a Humanoid Robot, Master's thesis, MIT Department of Electrical Engineering and Computer Science.

Jordan, M. I. & Rumelhart, D. E. (1992), 'Forward Models: supervised learning with a distal teacher', *Cognitive Science* **16**, 307–354.

MacKay, W. A., Crammond, D. J., Kwan, H. C. & Murphy, J. T. (1986), 'Measurements of human forearm posture viscoelasticity', *Journal of Biomechanics* **19**, 231–238.

Marjanovic, M. (1995), Learning Functional Maps Between Sensorimotor Systems on a Humanoid Robot, Master's thesis, MIT Department of Electrical Engineering and Computer Science.

Matsuoka, Y. (1995), Embodiment and Manipulation Learning Process for a Humanoid Hand, Master's thesis, MIT Department of Electrical Engineering and Computer Science.

Mussa-Ivaldi, F. A. & Giszter, S. F. (1992), 'Vector field approximation: a computational paradigm for motor control and learning', *Biological Cybernetics* **67**, 491–500.

Mussa-Ivaldi, F. A., Giszter, S. F. & Bizzi, E. (1994), 'Linear combinations of primitives in vertebrate motor control', *Proceedings of the National Academy of Sciences* **91**, 7534–7538.

Nelson, W. L. (1983), 'Physical Principles for Economies of Skilled Movements', *Biological Cybernetics* **46**, 135–147.

Pratt, G. A. & Williamson, M. M. (1995), Series Elastic Actuators, *in* 'Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-95)', Vol. 1, Pittsburg, PA, pp. 399–406.

Scassellati, B. (1995), High Level Perceptual Contours from a Variety of Low Level Features, Master's thesis, MIT Department of Electrical Engineering and Computer Science.

Williamson, M. M. (1996), Postural primitives: interactive behavior for a humanoid robot arm, Society of Adaptive Behavior. In these proceedings.

Zajac, F. E. (1989), 'Muscle and tendon:Properties, models, scaling, and application to biomechanics and motor control', *CRC Critical Reviews of Biomedical Engineering* **17**(4), 359–411.