Hebbian Learning of Visually Directed Reaching by a Robot Arm

Yiwen Wang[1], Tingfan Wu[2], Garrick Orchard[3], Piotr Dudek[4], Michele Rucci[5] and Bertram E. Shi[1]

[1]Hong Kong Univ. of Science and Tech.
Kowloon, HONG KONG SAR

[2]Univ. of California at San Diego
La Jolla, CA, U.S.A

[3]Johns Hopkins Univ.
Baltimore, MD, USA

[4]Univ. of Manchester
Manchester, UK

[5]Boston University
Boston, MA, USA

Abstract—**We describe a robotic system consisting of an arm and an active vision system learns to align its sensory and motor maps so that it can successfully reach the tip of its arm to touch the point where it is looking. This system uses an unsupervised Hebbian learning algorithm, and learns the alignment by watching its arm waving in front of its eyes. After watching for 25 minutes, the maps are sufficiently well aligned that it can execute the desired behavior.**

I. Introduction

A number of studies have investigated supervised learning of visually guided pointing or reaching by a robotic arm, e.g. [1][2][3]. In this work, the robot might generate an arm command based upon the visual location of the target. After executing the command, it uses the error between the target location and the actual location of the end effector to update the network that generates arm commands from visual signals.

Here, we describe a robot that uses a Hebbian network for unsupervised learning of this behavior. Salinas and Abbot have shown that the required alignment of sensory and motor signals can be achieved using correlation based rules operating while waving the arm randomly and watching it with its eyes [4]. This idea has been further elaborated to incorporate dynamical behavior in [5]. The work we describe here follows this approach, but demonstrates it on a real physical robot, rather than through simulation. It differs from prior unsupervised approaches on physical robots in that we use a Hebbian network, rather than a Kohonen map [6].

II. System Description

The robotic testbed (Figure 1) consists of a robotic arm (the Cyton Alpha 7D-1G arm from Robai, www.robai.com) and a binocular active vision system. The vision system gets input from a Bumblebee stereo camera (Point Grey Research, www.ptgrey.com) mounted on a PTU-D46-17 pan-tilt unit (Directed Perception, www.dperception.com). Visual processing is performed by three HKUST MultiMap Boards [7]. The results from the Multimap boards are transferred to a PC, which implements the learning algorithm and controls the arm and pan-tilt head.

As a baseline visual behavior, the robot maintains binocular fixation on a light emitting diode (LED). The LED provides a convenient way to establish the efficacy of the Hebbian learning rule described here, by reducing uncertainty about where the robot is fixating. During training, this LED is fixed at the tip of the arm, so that the robot can unambiguously locate the arm tip location. During testing, the robot moves the arm tip location to that of the LED.

In order to maintain fixation, the robot adjusts the pan and tilt angles of the pan-tilt head. Because the physical orientations of the two cameras are fixed mechanically, we orient them virtually by adjusting the locations of sub-windows within the image, which are analogous to foveae. The sub-window locations are adjusted anti-symmetrically, so that their movements mimic vergence eye movements.

The visual tracking strategy first identifies the LED location by the point with maximum intensity in the image. If the maximum intensity is greater than a threshold, it updates the vergence angle so that the difference between the horizontal locations of the LED in the left and right sub-windows is constant. It updates the pan and tilt angle so that the average of the horizontal and vertical LED locations of the LED across the two sub-windows are centered in the sub-window. Updates occur every 40ms (25 Hz).

As shown in Figure 2, the network represents the visual coordinates, the arm command signals, the actual arm angles and an externally specified set of desired set of motor coordinates as two dimensional arrays. We move only the shoulder and elbow joints of the arm, which we will refer to as Joint 0 and Joint 1. Thus, the tip of the arm moves along a 2D surface in 3D space. To match the dimensions of the visual and motor coordinates, we extract only the commanded pan and tilt angles as visual coordinates. We use 10x10 arrays of neurons. For consistency with the visual tracking system

This was done while the authors were participants of the 2009 Telluride Neuromorphic Engineering Workshop, as a project in the Vision Chips Workgroup. For inquiries, please contact BES (eebert@ee.ust.hk).

described above, the state of the network is updated every 40ms (25 Hz).

Although the processing is feed-forward, there are feedback loops due to the connections to the robot. In particular, during training, the activity in the arm command neurons affects the activity in visual coordinate neurons, since changes in the arm position cause changes in the visual angles as the vision system tracks the LED at the arm tip. In addition, there is a feedback loop from the arm control to the arm angle neurons, since changes in the commanded angles give rise to changes in the actual angles. This self-excitatory feedback loop serves as a "memory", generating arm commands that keep the arm at its current position in the absence of activity from the other networks.

In the 2D neuron array representing the arm commands, each neuron represents a particular combination of joint angles. The distributed pattern of activity across this array is converted into a pair of commanded joint angles by taking the centroid of the activity. Suppose the number of neurons across each array dimension is N. Denote the activity of neuron $m_a(x_0, x_1)$, where (x_0, x_1) are integer indices running from 0 to $(N$-1). The commanded joint angles (c_0, c_1) are given by

$$\begin{aligned} c_{0/1} &= \frac{1}{M}\sum_{x_0=1}^{N}\sum_{x_1=1}^{N}\tfrac{x_{0/1}-1}{N-1}m_a(x_0,x_1) \\ M &= \sum_{x_0=1}^{N}\sum_{x_1=1}^{N}m_a(x_0,x_1) \end{aligned} \tag{1}$$

where 0/1 can be replaced by either 0 or 1. These normalized coordinates running from 0 to 1 are converted to actual joint angle commands by a linear scaling and offset.

Similarly to the arm command array, each neuron in the arm angle array represents a particular combination of joint angles assumed by the arm. The activity of each neuron in the array is determined by a Gaussian function that decays with the distance between the actual arm angles and the arm angles represented by each neuron. Denoting the activity of the neurons in the array by $p_a(x_0, x_1)$ and the actual arm angles in normalized coordinates by (a_0, a_1),

$$p_a(x_0,x_1) \propto \mathcal{N}\left(\left(\tfrac{x_0}{N-1},\tfrac{x_1}{N-1}\right) \mid (a_0,a_1), \sigma^2\mathbf{I}\right) \tag{2}$$

where $\mathcal{N}(\vec{x} \mid \vec{m}, \mathbf{C})$ is a multi-dimensional Gaussian with mean $\vec{m}$ and covariance matrix $\mathbf{C}$, $\mathbf{I}$ is the identity matrix, σ^2 is a positive constant controlling the width of the Gaussian, and (x_0, x_1) are integer indices running from 0 to $(N$-1).

Because the servomotors used to actuate the arm joints in our current implementation do not provide feedback, we assume that the joint angles assumed by the arm at each time are equal to the commanded joint angles from the previous time step. Thus, these can be considered to be efference copies of the motor command signal. As mentioned above, this self-excitatory feedback loop enables the network to maintain a constant arm position without activity in the other two arrays. It also provides a mechanism for us to incorporate proprioceptive feedback, if available in future implementations.

In the 2D neuron array representing the visual coordinates, each neuron represents a particular combination of pan and tilt angles. We denote the activity in the neurons by $p_v(x_0, x_1)$. The activity in the array is a Gaussian function similar to (2), but with mean equal to the visual coordinates (v_0, v_1). The visual coordinates v_1 and v_2 do not correspond directly to the normalized pan and tilt angles, P and T, but by

$$\begin{bmatrix} v_0 \\ v_1 \end{bmatrix} = \begin{bmatrix} 4 & 4 \\ 0.5 & -0.5 \end{bmatrix}\begin{bmatrix} P \\ T \end{bmatrix} + \begin{bmatrix} 2.95 \\ 0.6 \end{bmatrix} \tag{3}$$

This affine transformation is used to better spread the visual angles observed during training over the range covered by the neurons.

The activity in the 2D external input neuron array is a Gaussian function similar to (2) where the mean is given by (e_0, e_1), an external command signal in normalized coordinates. This external input is provided during training, so that the arm can be commanded to explore the arm joint space.

The neurons in the arm command array are driven by the outputs of the neurons in the other arrays through a set of weights W_a and W_v :.

$$\begin{aligned} m_a(x_0,x_1) &= \sum_{\xi_0=1}^{N}\sum_{\xi_1=1}^{N}W_a(x_0,x_1,\xi_0,\xi_1)p_a(\xi_0,\xi_1) \\ &+ \sum_{\xi_0=1}^{N}\sum_{\xi_1=1}^{N}W_v(x_0,x_1,\xi_0,\xi_1)p_v(\xi_0,\xi_1) + C\cdot p_e(x_0,x_1) \end{aligned} \tag{4}$$

The connections from the external input array and the arm command are topologically organized and one-to-one. The weight C is large so that the external input, when present, overrides the signals from the other arrays.

The weights are learned via a Hebbian learning rule. The weight updates depend upon correlations in the activity at the source and target of the weight:

$$\begin{aligned} \Delta W_a(x_0,x_1,\xi_0,\xi_1) &= \eta\cdot m_a(x_0,x_1)\cdot p_a(\xi_0,\xi_1) \\ \Delta W_v(x_0,x_1,\xi_0,\xi_1) &= \eta\cdot m_a(x_0,x_1)\cdot p_v(\xi_0,\xi_1) \end{aligned} \tag{5}$$

where η is a positive step size parameter. To avoid the weights growing without bound, we normalize so that the weights leaving each neuron sum to one.

III. Experimental Results

A. Training Procedure

During training, the robot waves its arm in front of its eyes. The robot holds an LED in the grasper at the end of the arm. The vision system tracks the LED by updating the pan/tilt/vergence angles 25 times per second. These updates are reflected as new patterns of activity in the visual coordinate array. A video showing the training of the network is available online [8].

The arm motion is established by first defining a 2D trajectory in the joint space, and using this 2D trajectory to control the activity in the external input array. The trajectory slowly sweeps the joint angle of Joint 0 from one end of its range to the other over 100 seconds, while Joint 1 moves back and forth linearly between the two ends of its range a total of 8 times. Over the next 100 seconds, the roles of the two joints are switched, with Joint 1 moving slowly and Joint 0 moving back and forth faster. This pattern is repeated during the training period, and ensures that all areas of the motor space are explored equally.

Because of the strong fixed connections between the external input array and the motor command array, the activity in the external input array is essentially mirrored in the motor command array. Due to boundary effects when computing the centroid, the actual motor command sent to the arm, (a_0, a_1), is shifted towards the center of the range in comparison to the external command, (e_0, e_1).

The Hebbian learning algorithm described in the previous section reinforces connections between the input and the output that are simultaneously active. Thus, as the robot moves its arm and tracks the LED at the tip, the weights W_v evolve so that weight between a neuron in the arm command array and the neurons visual coordinate array that are active when the robot is looking at the tip of the arm are strengthened. Similarly, the weights W_a evolve so that weights between topologically corresponding neurons in the arm command and arm angle arrays are strengthened. Thus, the robot learns the correlations between the sensory and motor spaces, which it can later exploit to exhibit pointing behavior during the testing phase.

At the beginning of training, the weights are initialized to small random positive values that sum to one. We ran the training for 25 minutes (37,500 weight updates).

Figure 3 shows the weight matrices at the start and end of training. The weight images are organized into a 10x10 array of 10x10 pixel blocks. Each 10x10 pixel block shows the projective field from one neuron in the arm or visual position array to the 10x10 array of motor command neurons. The blocks are organized according to the position of the source neuron in the array. Due to normalization, the pixel values in each 10x10 pixel blocks sum to one.

At the start of training, there is no structure in the weight matrices. At the end of training, the projections from each neuron have concentrated in a particular region of the motor command space, as evident by the bumps in the map. For the arm angle neurons, the locations of the bumps progress smoothly from left to right and top to bottom as we traverse the array, reflecting the topographical correspondence between the two maps. For the visual coordinate neurons, bumps are only evident for a subset of neurons. This is because even though the arm has explored the entire range of its space, this corresponds to only a subset of the visual coordinate space. The projective fields for areas of visual coordinate space that have not been explored remain unstructured. An important area for future investigation is to enable the system to tune the selectivity of the neurons automatically so that they best span the observed space of inputs. An online video [9] shows the weight evolution.

B. *Test Results*

In order to enable the system to point where it is looks, we simply disable the external input by zeroing the activity of all neurons in the external input array. In this case, the activity in the motor command array is generated purely by the activity in the arm angle and visual coordinate arrays. The activity in the visual coordinate array depends upon the pan and tint angles, which are determined through by the tracking of the LED position. Through learning, the weights from the visual coordinate array to the arm command array encodes the relationship between the visual coordinates and motor coordinates. Thus, the visual coordinate array sets up a pattern of activity in the motor command arrays that should command the robot arm to move to the location of the LED. On the other hand, the activity in arm angle array sets up a pattern of activity that should keep the arm at the same position. The combination implements a temporal low pass filtering on the arm commands generated by the visual tracking.

We first evaluated the performance of the system by leaving the LED at the tip of the arm, but disconnecting the robot arm from the network. This enabled us to control the arm joints to move to different positions, and compare the arm commands generated by the network with the actual commands.

Figure 4 compares the motor commands sent to the two joint angles as well as the motor commands generated by the network with the weights from the beginning of the training. Consistent with the random and uniform initialization of the weight matrices, the motor command generated by the network remains close to 0.5 for both joint angles even though the arm moves throughout the entire range of its motion.

Figure 5 shows the comparison after training. There is better agreement between the arm command positions and those generated by the network. However, we observe that the magnitude of the arm commands is smaller. We also observe coupling between the motion of the two joints, so that even when one joint is moving slowly, the motor command for that joint generated by the network oscillates approximately in phase with the motion of the other joint. We speculate that these differences arise due to several reasons. First, the range over which the system can track the actual arm position due to boundary effects, which shift the arm commands towards the center of the range as discussed above. Second, differences in the arm joint positions such as we observe do not necessarily imply large differences in the spatial locations of the arm tip.

Indeed, when we connect the arm to the network, we observe that the network can successfully control the arm angles so that the tip of the arm points towards the LED. A video of the robot arm tip tracking the LED is available online [10].

IV. Conclusion

We have demonstrated the development of hand eye coordination in a robotic system. The system uses distributed representation of sensory and motor space, consisting of arrays of neurons representing different combinations of arm joint angles and different combinations of visual coordinates. The system learns the required sensori-motor transformations by watching its arm and modifying weights connecting sensory and motor neurons using the simple and biologically plausible Hebbian learning rule. Future work on this system includes increasing the dimensionality of the system, and incorporating the ability for neurons to change their selectivity to better span the input space covered during training.

REFERENCES

[1] M. Kuperstein, "Neural model of adaptive hand-eye coordination for single postures." *Science*, vol. 239, no. 4845, pp. 1308–1311, 1988.

[2] T. Martinetz, H. Ritter, and K. Schulten. Three-dimensional neural net for learning visuo-motor coordination of a robot arm. *IEEE Transactions on Neural Networks*, vol. 1, pp. 131-136, 1990.

[3] M. Marjanovic, B. Scassellati, and M. Williamson, "Self-Taught Visually-Guided Pointing for a Humanoid Robot," in *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, Sep. 1996.

[4] E. Salinas and L. F. Abbot, "Transfer of Coded Information from Sensory to Motor Networks," *Journal of Neuroscience*, vol. 75, no. 10, pp. 6461-647, Oct. 1995.

[5] D. Caligiore, D. Parisi, and G. Baldassarre,). "Toward an integrated biomimetic model of reaching," in *6th IEEE International Conference on Development and Learning*, 2007

[6] M. Zeller, K. R. Wallace, and K. Schulten, "Biological visuo-motor control of a pneumatic robot arm," in Dagli et al., eds., *Intelligent Engineering Systems Through Artificial Neural Networks*, vol. 5, pp. 645-650, 1995.

[7] Y. Meng, E. K. C. Tsang, S. Y. M. Lam and B. E. Shi, "The HKUST Multimap System for Active Vision," *International Journal of Humanoid Robotics*, vol. 6, no. 3, pp. 505-536, 2009.

[8] "Training the Learn to Point System" (on-line video), http://www.youtube.com/watch?v=fq0fmmimViE

[9] "Weight evolution of the Learn to Point System" (on-lne video), http://www.youtube.com/watch?v=L7kZEp4R8BQ

[10] "Test results of the Learn to Point System" (on-line video), http://www.youtube.com/watch?v=gPfcOvVxzZU

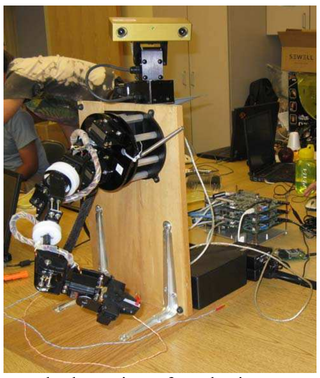
Figure 1: The robotic testbed consists of a robotic arm and a vision system.

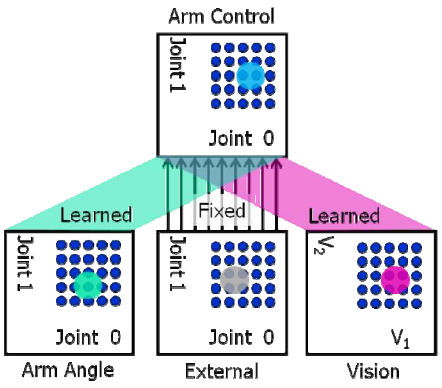

Figure 2: The neural network architecture.

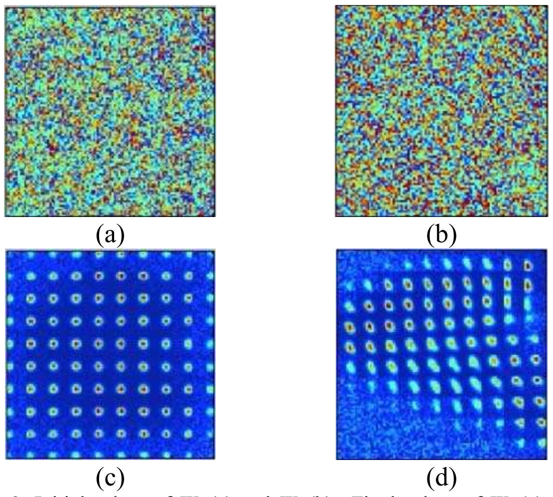

Figure 3: Initial values of W_a (a) and W_v (b). Final values of W_a (c) and W_v (d). Blue indicates small weights and red indicates large weights.

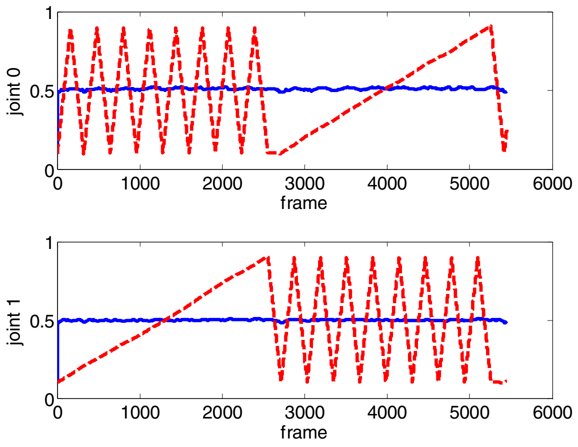

Figure 4: Comparison between actual arm commands (red dotted) and centroid of arm command array activity (blue solid) before training.

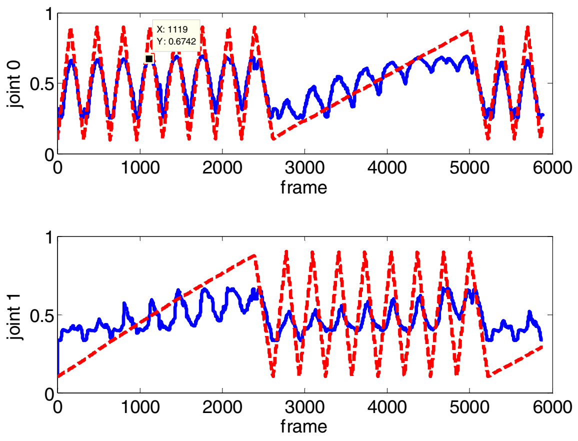

Figure 5: Comparison between actual arm commands (red dotted) and centroid of arm command array activity (blue solid) after training.