Developmental Robots: Theory, Method and Experimental Results

Juyang Weng, Wey S. Hwang, Yilu Zhang and Colin H. Evans

Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824, USA
E-mail: {weng,hwangwey,zhangyil,evansco1}@cse.msu.edu

Abstract

It is very challenging for humans to program a humanoid robot to act properly in human environment. Humans have a fundamental limitation in constructing an adequate model for the world or an adequate behavior model for the robot, because of the complexity of such models and the unpredictable unknown environments that the models must apply. This article introduces a new approach to intelligent robots, the developmental approach, which is different from other existing major approaches: knowledge-based, behavior-based, learning-based, and evolutionary approaches. The developmental approach is motivated by human mental development from infancy to adulthood, during which each human individual develops his cognitive and behavioral capabilities through interactions with the environment. This approach results in a new kind of robots, developmental robots — robots that can develop automatically. These robots require a new kind of algorithm — developmental algorithm — which enables the robot to learn new tasks without a need of reprogramming, including those tasks that the human programmer does not understand or cannot predict. This paper introduces the basic theory of developmental robots, the SAIL robot, the SAIL developmental architecture, the SAIL-2 developmental algorithm, and some experimental results from our SAIL project.

1 Introduction

The conventional mode of developmental process for a robot is not automated — the human designer is in the loop. Given a robotic task, it is the human designer to understand the task. Based on his understanding, he comes up with a representation, chooses a computational method, and writes a program that implements the method for the robot. During this developmental process, some machine learning may be used, during which some parameters are adjusted according to the collected data. However, these parameters are defined by the human designer for the task given. The resulting program is for this task only, not for any other tasks. This manual development paradigm has met tremendous difficulties for tasks that require complex cognitive capabilities, including tasks that autonomous robots must perform.

Since 1996, we have been working on a robotic project called SAIL (short for Self-organizing, Autonomous, Incremental Learner) and SHOSLIF is its predecessor [16]. The goal of the SAIL project represents a fundamental departure from traditional engineering paradigm — to *automate* the *developmental* process of robots. In this paper, we describe the theory, method and experimental results of the SAIL-2 developmental algorithm for the SAIL robot.

Our developmental algorithm is motivated by the human development. At the birth time of the SAIL robot, its developmental algorithm starts to run. This developmental algorithm runs in real time, through the entire "life span" of the robot. The robot learns while performing simultaneously. The human trainers train the SAIL robot by interacting with it, very much like the way human parents interact with their infant. The SAIL developmental algorithm updates the robot memory in real-time according to what is sensed by the sensors and what is sent to the effectors as control signals.

Automated development requires a capability of learning but it requires something more fundamental. A developmental algorithm for automated development must be able to learn new tasks and new skills without requiring reprogramming, including those tasks and skills that the programmer cannot predict. This basic capability of human developmental algorithm enables humans to learn more and more skills without a need to change his brain.

2 The Developmental Approach

Automated development requires a drastic departure from the current task-specific nature of all the existing approaches. Table 1 outlines the major characteristics of existing approaches to constructing artificial systems and the

Table 1. Comparison of Approaches

Approach	Species architecture	World knowledge	System behavior	Task specific
Knowledge-based	programming	manual modeling	manual modeling	Yes
Behavior-based	programming	avoid modeling	manual modeling	Yes
Learning-based	programming	treatment varies	special-purpose learning	Yes
Evolutionary	genetic search	treatment varies	genetic search	Yes
Developmental	programming	avoid modeling	general-purpose learning	No

new developmental approach. The developmental approach relieves humans from explicit design of (a) any task-specific representation and knowledge and (b) system behavior representation, behavior modules and their interactions.

2.1 AA-learning

A robot agent *M* may have several sensors. By definition, the *extroceptive*, *proprioceptive* and *interoceptive* sensors are, respectively, those that sense stimuli from external environment (e.g., visual), relative position of internal control (e.g., arm position), and internal events (e.g., internal clock).

The operational mode of automated development can be termed AA-learning (named after *a*utomated, *a*nimal-like learning *without claiming to be complete*) for a robot agent.

Definition 1 *A robot agent M conducts AA-learning at discrete time instances, $t = 0, 1, 2, ...$, if the following conditions are met: (I) M has a number of sensors, whose signal at time t is collectively denoted by $x(t)$. (II) M has a number of effectors, whose control signal at time t is collectively denoted by $a(t)$. (III) M has a "brain" denoted by $b(t)$ at time t. (IV) At each time t, the time-varying state-update function f_t updates the "brain" based on sensory input $x(t)$ and the current "brain" $b(t)$:*

$$b(t+1) = f_t(x(t), b(t)) \quad (1)$$

and the action-generation function g_t generates the effector control signal based on the updated "brain" $b(t+1)$:

$$a(t+1) = g_t(b(t+1)) \quad (2)$$

where $a(t+1)$ can be a part of the next sensory input $x(t+1)$. (V) The "brain" of M is closed in that after the birth (the first operation), $b(t)$ cannot be altered directly by human teachers for teaching purposes. It can only be updated according to Eq. (1).

As can be seen, AA-learning requires that a system cannot have two separate phases for learning and performance. An AA-learning agent learns while performing.

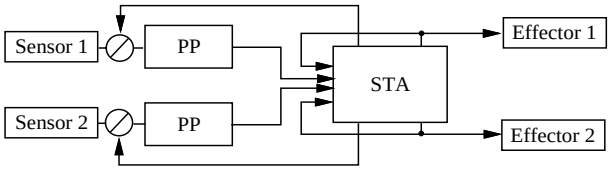


Figure 1. A schematic illustration of the coarse architecture of the proposed learning mechanism. A circle represents an attention selector. It is also an effector. PP: preprocessor. STA: spatiotemporal associator.

2.2 Outline of developmental architecture

Fig. 1 gives a schematic illustration of the coarse architecture of the proposed AA-learning mechanism SAIL. The preprocessor in Fig. 1 performs some sensor-specific transformation, such as intensity normalization, automatic gain (contrast) control, filtering, etc. The attention selector in Fig. 1 is an intero-effector, which selects, e.g., a subpart of the image frame for later processing. As shown in the figure, the input to the STA includes not just information from the sensors, but also the current control signal of the effectors.

2.3 Level: length of temporal context

As a basic point of developmental approach summarized in Table 1, we do not define architecture levels in terms of either domain knowledge hierarchy or system behavior hierarchy, as is common in existing works (e.g., see a recent survey article about machine learning by Langley [10]). Instead, our proposed level-based architecture corresponds to the length of temporal context.

In the AA-learning mechanism, the global state s of the "brain" $b(t)$ at any time t is represented distributedly by states at different levels: $s = (s_0, s_1, s_2, ..., s_L)$, where s_i, $i = 0, 1, 2, ..., L$, represents the state at level i. The current number of levels is determined automatically based on the maturation schedule of the system which depends on the

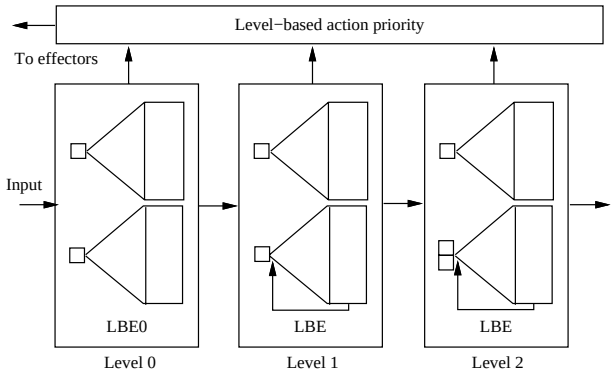


Figure 2. The levels that are built automatically in the STA. Each level corresponds to a level-building element (LBE).

experience as well as the virtual age [1] of the system. Level 0 is context free, to model S-R (stimulus-response) reflex. Starting from level 1, temporal context is incorporated. The higher the level i, the more temporal context each state at level i represents. Fig. 2 illustrates the level building in STA. The basic mechanism of the level-building elements for each level is basically the same. The differences between levels will be explained later. From Fig. 2, one may immediately see the similarity between this scheme and the level arrangement with Rodney Brooks' well-known subsumption architecture [4]. The major differences are: (1) Levels in SAIL are not defined in the sense of behavior as in the subsumption architecture, but rather in the extent of temporal context that is recorded. Each level in SAIL can incorporate many behaviors as long as each behavior has a similar amount of temporal context. (2) Mediation among many behaviors both within each level and among different levels are automatically learned in SAIL, instead of being programmed in. Such a mediation is extremely difficult to hand craft and program when the number of behaviors is large. In the following discussion, we will concentrate on a single level. Integrating different levels is beyond the scope of this paper.

2.4 State

In behavior-based learning approaches, the states of a robot agent are manually bound to a set of predefined task concepts before training. A developmental algorithm must automatically generate states without being given any task.

Let us first consider level 1 in Fig. 2. The part of the "brain" state at this level is denoted by a state vector $s(t)$. If $s(t)$ is considered a random process, Eqs. (1) and (2) are closely related to the formulations for Markov decision processes (MDP) [12], or HMMs (hidden Markov models) if the action part is omitted [8]. Indeed, the state transition function f and the decision function g can be based on probability distributions shown below to take into account the uncertainty in states, observations and actions:

$$\mathrm{P}(s(t+1) = s' \mid x(t), s(t) = s)$$

and

$$\mathrm{P}(a(t+1) = a \mid s(t+1) = s')$$

where $\mathrm{P}(\cdot)$ denotes the probability. However, the states in MDPs have been typically defined as a set of symbols and there is no distance metric defined to measure the similarity between any two symbols (see, however, various MDP generalization techniques surveyed by Kaelbling, Littman & Moore [9]).

In contrast to existing MDP methods, we define the state space at level 1 to be $\mathcal{S} = \mathcal{X} \times \mathrm{R}(\mathcal{S})$, where $\times$ denotes Cartesian product and $\mathrm{R}(\cdot)$ denotes a re-sampling operator.

We define a state s to be a vector in a high dimensional space $\mathcal{S}$. Thus, our state has an explicit representation. $\mathcal{S}$ must contain all the possible sensory input $x \in \mathcal{X}$. In contrast to existing MDP methods, we require that the state records temporal context. Thus, we define the state space at level 1 recursively to be $\mathcal{S} = \mathcal{X} \times \mathrm{R}(\mathcal{S})$, where $\times$ denotes the Cartesian product and $\mathrm{R}(\cdot)$ denotes a re-sampling operator. The design of the re-sampling operator needs to take into account (a) the nature of the signal, (b) the desired temporal span in the state vector, and (c) the recursive relation $\mathcal{S} = \mathcal{X} \times \mathrm{R}(\mathcal{S})$. For example, for 1-D signal, $R()$ can be 2:1 uniform resampling. That is, if $s = (s_1, s_2, ..., s_{2m})$, $\mathrm{R}(s) = (s'_1, s'_2, ..., s'_m)$ is computed by average resampling: $s'_i = (s_{2i-1} + s_{2i})/2$, $i = 1, 2, ..., m$.

Thus, the state transition function in Eq. (1) represents a simplified mapping $f : \mathcal{X} \times \mathrm{R}(\mathcal{S}) \mapsto \mathcal{S}$ at level 1. First, since the state space cannot be manually designed, we let f map $(x(t), \mathrm{R}(s(t)))$ directly to itself:

$$s(t+1) = (\mathrm{R}(s(t), x(t)). \tag{3}$$

In other words, the next state $s(t+1)$ keeps all the information of sensory input $x(t)$ and the re-sampled version of the current state $s(t)$. Given sensory inputs $x(0), x(1), ...$, this simplified f defines a trajectory of states $s(1) = (0, x(0))$, $s(2) = (\mathrm{R}(s(1), x(1))$ and so on. Fig. 3 gives an illustration of a state at time $t = 3$, which uses 1-D, 2:1 uniform resampling. The 2:1 resampling rate reduces the resolution by a factor of 2 through time. If $x(t)$ of a particular sensing modality has low dimensionality but a longer history is necessary in the state representation, a slower resolution reduction rate is necessary. For example, 3:2 or 5:3 ratios can be used.

[1] The virtual age is the time of operation since the birth of the system.

Figure 3. The state representation at $t = 3$ for 1-D signal with 2:1 uniform resampling in space, from sensory inputs $x(0)$, $x(1)$, $x(2)$, ... For this illustration, $\mathcal{S}$ is an 8-dimensional space. $x'(0)$ is the average of $x(0)$ and the initial zeros in the state vector.

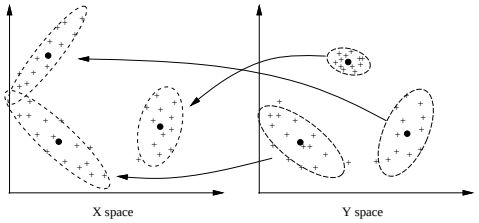


Figure 4. Y-clusters in space $\mathcal{Y}$ and the corresponding x-clusters in space $\mathcal{X}$. The first and the second order statistics are updated for each cluster.

3 The Mapping Engine

The two functions (1) and (2) are realized by the same mapping engine. In the work reported here, teaching is done by supplying desired action in real time. When action is not supplied, the system provides the action using the estimated mapping. In other words, we use supervised learning in the work reported here.

The developmental algorithm needs to map high dimensional inputs to numerical outputs. We introduce a new technique to incrementally achieve this mapping. Let us consider a general regression problem: approximating a mapping $h : \mathcal{X} \mapsto \mathcal{Y}$ from a set of training samples $\{(x_i, y_i) \mid x_i \in \mathcal{X},\ y_i \in \mathcal{Y},\ i = 1, 2, \ldots, n\}$. If y_i was a class label, we could use linear discriminant analysis (LDA) [6] since the within-class scatter and between-class scatter matrices are all defined. However, if y_i is a numerical output, which can take any value for each input component, it is a challenge to figure out an effective discriminant analysis procedure that can disregard input components that are either irrelevant to output or contribute little to the output. We introduce a new hierarchical statistical modeling method. Consider the mapping $h : \mathcal{X} \mapsto \mathcal{Y}$, which is to be approximated by a regression tree, called incremental hierarchical discriminating regression (IHDR) tree, for the high dimensional space $\mathcal{X}$. Our goal is to automatically derive discriminating features although no class label is available (other than the numerical vectors in space $\mathcal{Y}$). In addition, for real-time requirement, we must process each sample (x_i, y_i) to update the IHDR tree using only a minimal amount of computation.

Two types of clusters are incrementally updated at each node of the IHDR tree — y-clusters and x-clusters, as shown in Fig. 4. The y-clusters are clusters in the output space $\mathcal{Y}$ and x-clusters are those in the input space $\mathcal{X}$. There are a maximum of q (e.g., $q = 6$) clusters of each type at each node. The q y-clusters determine the virtual class label of each arriving sample (x, y) based on its y part. Each x-cluster approximates the sample population in $\mathcal{X}$ space for the samples that belong to it. It may spawn a child node from the current node if a finer approximation is required. At each node, y in (x, y) finds the nearest y-cluster in Euclidean distance and updates (pulling) the center of the y-cluster. This y-cluster indicates which corresponding x-cluster the input (x, y) belongs to. Then, the x part of (x, y) is used to update the statistics of the x-cluster (the mean vector and the covariance matrix). These statistics of every x-cluster are used to estimate the probability for the current sample (x, y) to belong to the x-cluster, whose probability distribution is modeled as a multidimensional Gaussian at this level. In other words, each node models a region of the input space $\mathcal{X}$ using q Gaussians. Each Gaussian will be modeled by more small Gaussians in the next tree level if the current node is not a leaf node.

Moreover, the center of these x-clusters provide essential information for discriminating subspace, since these x-clusters are formed according to virtual labels in $\mathcal{Y}$ space. We define a discriminating subspace as the linear space that passes through the centers of these x-clusters. A total of q centers of the q x-clusters give $q-1$ discriminating features which span $(q-1)$-dimensional discriminating space. A probability-based distance called size-dependent negative-log-likelihood (SNLL) [7] is computed from x to each of the q x-clusters to determine which x-cluster should be further searched. If the probability is high enough, the sample (x, y) should further search the corresponding child (maybe more than one but with an upper bound k) recursively, until the corresponding terminal nodes are found.

The algorithm incrementally builds an IHDR tree from a sequence of training samples. The deeper a node is in the tree, the smaller the variances of its x-clusters are. When the number of samples in a node is too small to give a good estimate of the statistics of q x-clusters, this node is a leaf node.

The above incrementally constructed tree gives a coarse-to-fine probability model. If we use Gaussian distribution to model each x-cluster, this is a *hierarchical version* of the well-known mixture-of-Gaussian distribution models: the deeper the tree is, the more Gaussians are used and the finer are these Gaussians. At shallow levels, the sample distribution is approximated by a mixture of large Gaussians (with large variances). At deep levels, the sample distribution is approximated by a mixture of many small Gaussians

(with small variances). The multiple search paths guided by probability allow a sample x that falls in-between two or more Gaussians at each shallow level to explore the tree branches that contain its neighboring x-clusters. Those x-clusters to which the sample (x, y) has little chance to belong are excluded for further exploration. This results in the well-known logarithmic time complex for tree retrieval: $O(\log m)$ where m is the number of leaf nodes in the tree, assuming that the number of samples in each leaf node is bounded above by a constant.

4 Experiments

4.1 SAIL robot

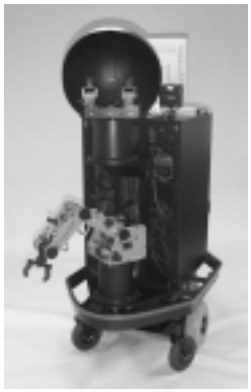


Figure 5. The SAIL robot built at the Pattern Recognition and Image Processing Laboratory at Michigan State University.

A real robot called SAIL was assembled at MSU, as shown in Fig. 5. SAIL robot's "neck" can turn. Each of its two "eyes" is controlled by a fast pan-tilt head. Its torso has 4 pressure sensors to sense push actions and force. It has 28 touch sensors on its arm, neck, head, and bumper to allow human to teach how to act by direct touch. Its drive-base is adapted from a wheelchair and thus the SAIL robot can operate both indoor and outdoor. Its main computer is a high-end dual-processor dual-bus PC workstation with 512MB RAM and an internal 27GB three-drive disk array for real-time sensory information processing, real-time memory recall and update as well as real-time effector controls. This platform is being used to test the architecture and the developmental algorithm outlined here.

4.2 Experimental results for the IHDR tree algorithm

Before real-system online tests, we conducted careful experiments on the performance characteristics of the proposed architecture and the algorithm. Here we present some experimental results for our mapping engine – IHDR tree algorithm.

Since the input to our developmental algorithm contains very high dimensional data, we first show the performance of the IHDR tree algorithm applied to the face recognition problem. We treat each face image of m rows and n columns as a mn-dimensional vector, where each component of the vector corresponds to the intensity of each pixel. The experimental data set used here is face images from the Weizmann Institute at Israel. The image database were constructed from 28 human subjects, each having thirty images all combinations of two different expressions under three different lighting conditions with five different orientations. The preprocessed images have a resolution of 88×64, resulting an input space of 5632. The task here is to classify images into person's ID as class label. We used the mean of all training images of each person as the corresponding y vector.

The data set was divided into two groups: training set and testing set. The training set contains 504 face images. Each subject contributed 18 face images in the training set. The 18 images include three different poses, three different lightings, and two different expressions. The remaining 336 images were used for the testing set. Each subject had 12 images for testing, which include two different poses, three different lightings, and two expressions. In order to present enough training samples for the IHDR algorithm to build a stable tree, we artificially increase the samples by presenting training samples to the program 20 times (20 epochs). Table 2 compares different appearance-based methods. We used 95% sample variance in determining the number of basis vectors (eigenvectors) in the principal component analysis (PCA). PCA is faster than nearest neighbor (NN) and shares a similar accuracy. However, the 95% of variance results in about 98 eigenvectors which are much less than that of NN (5632-D!). PCA organized with a binary tree was faster than straight NN as shown in the Table. It is the fastest algorithm among all the methods we tested but the performance is worse than those of PCA and NN. The accuracy of LDA is the third best. Our new IHDR method is faster than LDA and resulted in the lowest error rate.

We also applied support vector machines (SVM) [5] to this image set to compare the performance. Support vector machines utilizes the structural risk minimization principle [15]. It results in a maximum separation margin and the solution depends only on the training samples (support vectors) which are located on the supporting planes. SVM has been applied on both classification and regression problems. We used the SVM software obtained from Royal Holloway, University of London [14] for this experiment. We used the PCA of the face images as the input features for

the SVM[2] The best result we obtained by tuning the parameters of the SVM software is reported in Table 2. The recognition rate of the SVM with PCA is similar to that of PCA alone. However, SVM with PCA is faster than PCA. This is because SVM has more compact representation and PCA alone needs to conduct linear search for every training sample.

We compared the error rate of the proposed IHDR algorithm with some major tree classifiers. CART [3] and C5.0 [13] are among the best known classification trees[3] . However, like most other decision trees, they are univariate trees in that each internal node used only one input component to partition the samples. This means that the partition of samples is done using hyperplanes that are orthogonal to one axis. We do not expect this type of tree can work well in a high dimensional space. Thus, we also tested a more recent multivariate tree OC1 [11]. We realize that these trees were not designed for high dimensional spaces like those from images We also tested the corresponding versions by performing PCA before using CART, C5.0, and OC1 and call them CART with PCA, C5.0 with PCA, and OC1 with PCA, respectively.

We have also compared the batch version of this algorithm. The batch version, named hierarchical discriminating regression (HDR) tree, computes statistics of training samples in a batch fashion. We expect the batch method out-perform the incremental one. However, the error rate of IHDR tree is lower than that of HDR tree for this set of data. The reason is that the same training samples might distribute in different leaf nodes for the IHDR tree because we run several iterations during training. For batch version, each training sample will only be allocated in one leaf node.

Table 2. The performance for Weizmann face data set

Method	Error rate	Avg. testing time (msec)
PCA	12.8%	115
PCA tree	14.58%	34
LDA	2.68%	105
NN	12.8%	164
SVM with PCA	12.5%	90
C5.0 with PCA	45.8%	95
OC1 with PCA	44.94%	98
HDR tree	1.19%	78
IHDR tree	0.6%	74

[2]The software failed when we used the original image input with dimensionality 5362.

[3]We have experimented the same data set using CART implemented by OC1. The performance is significantly worse than those reported in the Table 2.

4.3 Experiments with autonomous navigation problem

A vision-based navigation system accepts an input image X and outputs the control signal C to update the heading direction of the vehicle. The navigator can be denoted by an function f that maps the input image space $\mathcal{X}$ to control signal space $\mathcal{C}$. The learning process of the autonomous navigation problem then can be realized as a function approximation. This is a very challenging task since the function to be approximated is for a very high dimensional input space and the real application requires the navigator to perform in real time.

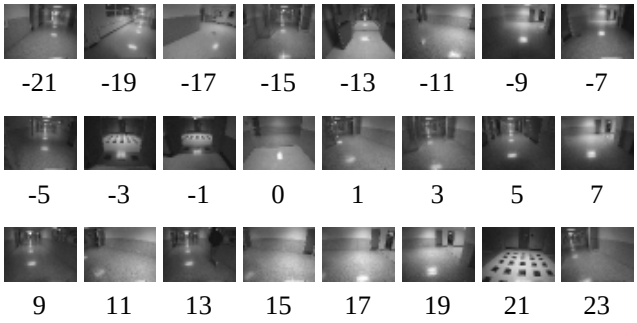
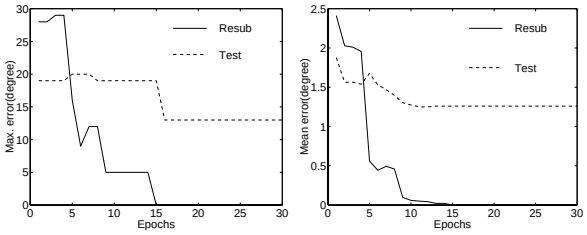


Figure 6. A subset of images used in autonomous navigation problem. The number right below the image shows the needed heading direction (in degrees) associated with that image.

We applied our IHDR algorithm to this challenging problem. Some of the example input images are shown in Fig 6. Totally 318 images with the corresponding heading directions were used for training. The resolution of each image is 30 by 40. We used the other 204 images to test the performance of the trained system. Fig 7 shows the maximum error rates and the mean error rates versus the number of training epochs. Both maximum error and mean error converge around the 15th epoch. Fig 8 gives plots of the histograms of the error rates at different epochs. As shown even after the first epoch, the performance of the IHDR tree is already reasonably good. With the increase of the epochs, we observed the improvement of the maximum error and mean error. The improvement stopped at the 15th epoch because we did not use any new training samples in each epoch and the system has perfectly fit the existing training data set. our test on real mobile robot has shown that a system of such an error level can navigate the robot very reliably for hours until the batteries are exhausted.

We also compare our experimental results with two artificial neural networks (ANN). with a consideration that the pattern-by-pattern training mode of artificial neural networks is also an incremental learning method. A two-



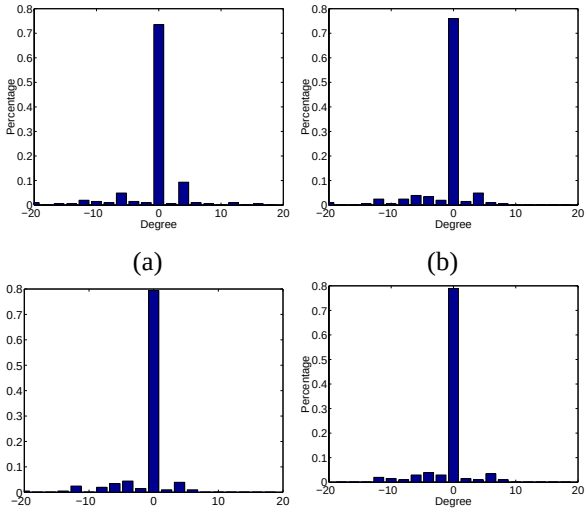
(a) (b)

Figure 7. The performance of the autonomous navigation. (a) The plot for maximum error rates vs. epochs. (b) The plot for mean error rates vs. epochs. The solid line represents the error rates for resubstitution test. The dash line represents the error rates for the testing set.

Table 3. The performance for vision-based navigation

Algorithm	Mean error (degree)		Max. error (degree)	
	Resub.	Testing set	Resub.	Testing set
FF	1.02	2.00	10	12
RBF	1.53	1.84	12	12
IHDR tree	0.00	1.25	0	13

layer feed-forward (FF) network and a radial basis function (RBF) network were used to train and test for the mapping from the image space to control signal space using the same data set as used in our IHDR tree algorithm. Both resubstitution (resub.) method and disjoint testing were applied among those three algorithms. The results are listed in Table 3 which shows that our algorithm outperforms these two ANN methods.



(a) (b) (c) (d)

Figure 8. The histograms of the error rates. Plot (a), (b), (c), and (d) correspond to the histograms at epoch 1, 6, 11, 20, respectively.

Figure 9. Real-time training and testing for developmental SAIL robot: finding the nearby ball and then reaching for it.

4.4 Test for developmental algorithm

We run the developmental algorithm on the SAIL robot. Since tracking objects and reaching objects are sensorimotor behaviors first developed in early infants, we trained our SAIL robot for two tasks. In the first task, called finding-ball task, we trained the SAIL robot to find a nearby ball and then turn eyes to it so that the ball is located on the center of sensed image. In the second task, called reaching task, we trained the SAIL robot to reach for the object once it has been located and the eyes fixate on it.

Existing studies on visual attention selection are typically based on low-level saliency measures, such as edges and texture [1]. In Birnbaum's work [2], the visual attention is based on the need to explore geometrical structure in the scene. In our case, the visual attention selection is a result of past learning experience. Thus, we do not need to define any task-specific saliency features. It is the SAIL robot that automatically derive most discriminating features for the tasks being learned. At the time of learning, the ball was presented in the region of interest (ROI) inside the stereo images. The human trainer interactively pulls the robot's eyes toward the ball (through the touch sensors for the pan-tilt heads) so that the ball is located on the center of the region of ROI (fixating the eyes on the ball). The inputs to the developmental algorithm are the continuous sequence of stereo images and the sequence of the pan-tilt head control signal. Three actions are defined for the pan-tilt head in pan direction: 0 (stop), 1 (move to the left), or -1 (move to the right). The size of ROI we chose for this experiment is defined as 120×320. In the mind of trainer, the ROI is divided into five regions so that each region is of size 120×64. The

Table 4. The performance for the finding ball task

	Row	Column	2D
Average error (pixels)	12.9	6.8	14.58

Table 5. The performance for the arm reaching task

	Row	Column	2D
Average error (pixels)	16.2	9.0	18.53

goal of the finding-ball task, is to turn the pan-tilt head so that the ball is at the center region. Five training sessions were conducted on line in real time. In each session, the ball were presented at different five different regions. During testing, four test sessions were conducted to evaluate the performance of the training sessions. The evaluation results are listed in the Table 4.

The task of reaching is in fact to establish the coordination between eye and the arm in any eye direction. The robot arm either turn left or right according to the pan-tilt positions. This training session was also achieved by interactively turning the robot arm by pressing its touching sensors. The numbers of training and testing sessions for the reaching task are the the same as the corresponding ones for the finding-ball task. The evaluation result is shown in the Table 5.

5 Conclusions

We introduced here a new kind of robots: robots that can develop automatically through real-time interactions with the environment. A technical challenge for the developmental algorithm is the mapping engine that scalable — keeping real-time speed and a stable performance for a very large number of high dimensional vector data. With our powerful mapping engine, the developmental algorithm is able to operate in real time. The SAIL-2 developmental algorithm has successfully run on the SAIL robot for real-time interactive training and testing for two sensori-motor tasks: finding ball and reaching the centered ball, two early tasks that infants often learn to perform. Since the developmental algorithm is not task specific, we plan to train the SAIL robot for other more tasks to study the limitation of the current SAIL-2 developmental algorithm as well as the SAIL robot design.

References

[1] Martin Bichsel. *Strategies of Robust Object Recognition for the Automatic Identification of Human Faces*. Swiss Federal Institute of Technology, Zurich, Switzerland, 1991.

[2] Lawernce Birnbaum, Matthew Brand, and Paul Cooper. Looking for Trouble: Using Causal Semantics to Direct Focus of Attention. In *Proc of the IEEE Int'l Conf on Computer Vision*, pages 49–56, Berlin, Germany, May 1993. IEEE Computer Press.

[3] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1993.

[4] R. Brooks. Intelligence without reason. In *Proc. Int'l Joint Conf. on Artificial Intelligence*, pages 569–595, Sydney, Australia, August 1991.

[5] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[6] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, New York, NY, second edition, 1990.

[7] W. Hwang, J. Weng, M. Fang, and J. Qian. A fast image retrieval algorithm with automatically extracted discriminant features. In *Proc. IEEE Workshop on Content-based Access of Image and Video Libraries*, pages 8–15, Fort Collins, Colorado, June 1999.

[8] Jr. J. R. Deller, Jone G. Proakis, and John H. L. Hansen. *Discrete-Time Processing of Speech Signals*. Macmillan, New York, NY, 1993.

[9] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[10] P. Langley. Machine learning for intelligent systems. In *Proc. 14th National Conf. on Artificial Intelligence*, pages 763–769, Providence, RI, July 1997.

[11] S. K. Murthy. Automatic construction of decision trees from data: A multidisciplinary survey. *Data Mining and Knowledge Discovery*, 1998.

[12] M. L. Puterman. *Markov Decision Processes*. Wiley, New York, NY, 1994.

[13] J. Quinlan. Introduction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[14] C. Saunders, M. O. Stitson, J. Weston, L. Bottou, B. Schlkopf, and A. Smola. Support vector machine reference manual. Technical Report CSD-TR-98-03, Royal Holloway, University of London, Egham, UK, March. 1998.

[15] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[16] J. Weng and S. Chen. Vision-guided navigation using SHOSLIF. *Neural Networks*, 11:1511–1529, 1998.