Cañamero, L., Oudeyer, P.-Y., and Balkenius, C. (Eds) Proceedings of the Ninth International Conference on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems. Lund University Cognitive Studies, 146.

Reward-free Learning using Sparsely-connected Hidden Markov Models and Local Controllers

Kohtaro Sabe, Kenta Kawamoto, Hirotaka Suzuki, Katsuki Minamino, and Kenichi Hidai
Corporate R&D System Technologies Laboratories, Sony Corporation
Gotenyama TEC, 5-1-12, Kitashinagawa Shinagawa-ku, Tokyo, 141-0001 Japan
Kohtaro.Sabe@jp.sony.com

Abstract

A novel framework for behavior learning for an autonomous agent without any reward functions is presented. It is focused on how to build and control internal representations of various environments from incomplete data without any a priori knowledge. Hidden Markov Model (HMM) has the potential to represent hidden structures of environments from partial observations, yet learning of HMM parameters without any assumptions remains very difficult. Without loss of generalization, we bring constraints to the parameters of HMM to reduce learning difficulty. The only motive given to the agent is to get the control of learned internal states so that local controllers assigned to HMM nodes are learned to achieve this goal. The framework is tested on two different types of environments: a mobile robot and a robot manipulator. In the mobile robot environment, only the range finder sensors are given to the agent while the robot randomly moves through the maze-like room. As a result, a place representation is captured in the hidden states of HMM. We show that by learning to control these internal states to arbitrary states, the robot can exhibit the skill of navigation. The same model is applied to a single link robot manipulator environment and the robot acquires the control of its dynamical properties.

1. Introduction

The objective of our work is to build an autonomous agent with open-ended learning capability, which is driven by the ultimate goal to acquire the abilities to predict and control everything that the agent has experienced (Sabe et al., 2005) (Fujita, 2009). For such a system, the ability to cope with multiple tasks without a priori knowledge or even without objectives about each task is necessary. We think that the requirements of such open-ended learning agents can be defined as follows:

1. The agent self-organizes internal representations of the unknown environments from partial observation sequence.
2. The agent recognizes past and current states as well as predicts future states.
3. The agent controls the self-organized states to arbitrary target states if possible.
4. The agent has intrinsic motivations to improve above three abilities.
5. The agent has extrinsic motivations to sustain itself within the environments.

Reinforcement learning is a machine learning method that acquires the optimal behaviors based on actual experiences and rewards in unknown environments. However, this framework itself does not provide the solution to the first requirement. For the second and the third requirement, prediction and control are tightly coupled with task specific objective functions so that learned knowledge about the environments cannot be reused efficiently.

In the field of robotics, the framework of Partially Observed Markov Decision Process (POMDP) has been adopted in robot navigation tasks called Simultaneous Localization and Mapping (SLAM) (Leonard and Durrant-Whyte, 1991) in which robot develops a map of unknown environments and at the same time localizes and controls itself anywhere on the learned map. Most of the successful systems adopt the task dependent representation of the environment such as Occupancy Grids (Elfes, 1989). Motion models and observation models are usually known or parameterized from the configuration of the robots.

SLAM is derived from EM algorithm implemented for Hidden Markov Model (HMM) known as Baum-Welch algorithm (Baum, 1970). A HMM has the potential to represent hidden structures of environments from observations, yet learning of HMM parameters without any assumptions remains very difficult. Successful applications such as a speech recognition task usually define the structure and the observation model as the left-to-right HMM with a Gaussian Mixture Model.

In theory, fully connected HMM (ergodic HMM) can be used to estimate the structure of a learning subject because a set of parameters expressing the true structure of the subject is the global minimum in HMM learning. However when learning a fully connected large scale HMM, we can not expect it to converge to the global minimum.

We put an assumption that almost all of the real world phenomenon can be expressed with a sparse structure, e.g. Small World Network. From the analogy of neural connections in the cortex, the structure is at most 3 dimensionally constrained. Since the cortex is a sheet of neurons folded in 3D space, a 2D constrained structure may be sufficient.

Our scenario of the developmental learning is as follows.

1. The agent acts with randomly generated actions or innate actions (motor babbling). Sparsely-connected HMM is used to learn the sensor readings for certain periods of time to build the HMM structure of the environment.
2. The agent learns the local controllers assigned to each HMM node using the observations and actions whenever node transitions occurred during motor babbling.
3. The agent select one of its learned HMM nodes as a target node. It makes a plan to reach the target node from currently recognized node. It activates local controllers along the planned path to emit actions.
4. The agent repeats step 3 to refine HMM and controllers while the performance of controlling to desired nodes improves.

We will show how this general framework of development works by two different classes of simple simulated environments: One is a mobile robot and robot and the other is a robot manipulator.

In section 2, the overall method is described in which learning of sparsely-connected HMM and local controllers, and planning and execution procedures are explained. In section 3, the experiments and results using a mobile robot are described. In section 4, the experiments and results using a pendulum are described. The conclusions and the future works are discussed in the last section.

2 Methods

2.1 Framework of Reward-free Learning

The system diagram of proposed method is shown on Fig.1. It consists of the predictor module, the controller module, the planner module, and the innate controller.

In the learning phase, observed sensor signals are fed to the predictor module where Hidden Markov Model (HMM) is used to learn the internal models of observed sequences. After learning the model, the same sequences are used to recognize the corresponding state at each time step. The results of recognitions are fed to the controller module. Whenever state transitions occur, relations from sensor observations to actions are learned in local controllers assigned to each node of the HMM.

In the execution phase, the target state in the HMM is selected from the learned states and given to the execution module. A plan of HMM state transitions from current state to the target state is generated by

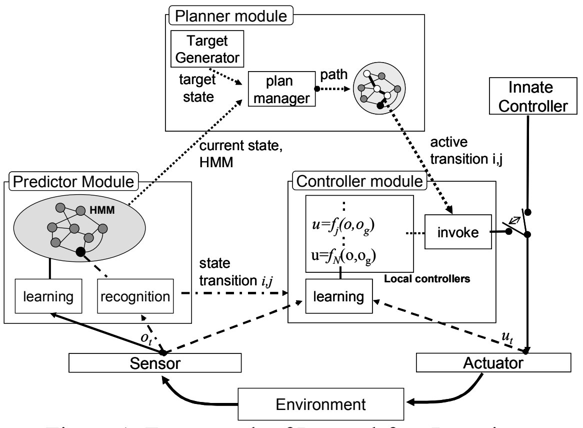

Figure 1: Framework of Reward-free Learning

means of graph search. This plan is executed by activating the corresponding local controllers along the path of the generated plan.

2.2 Sparsely-connected HMM

In the proposed method, we take the strategy to learn the internal models without its own actions and to learn the associations of actions and internal states later. Because in open-ended learning, everything observed and predicted may not be controllable and the causalities of actions to sensors should be discovered by the agent. In this paper, we assume that given actions have direct influences on the observations, but the learning of the predictors and controllers are separated for future extensions.

Hidden Markov Model (HMM) is a useful tool to model the observation sequences which are parameterized with initial state probabilities π_i, transition probability matrix a_{ij}, and observation likelihood at each state, modeled by mean μ_i and variance σ_i^2 of a Gaussian distribution.(Fig.2)

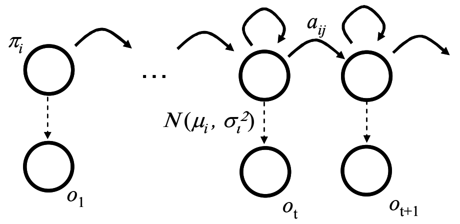

Figure 2 : Graphical model of HMM

To estimate the HMM parameters $\{\pi_i, a_{ij}, \mu_i, \sigma_i^2\}$ from given observation sequence o_t, Baum-Welch algorithm (Baum 1970), (Rabiner and Juang, 1993) is used. In the E step, the Forward Backward algorithm is used to estimate state probabilities at each time step and likelihood of given sequences. In the M step, HMM parameters are updated to maximize this likelihood. These steps are iteratively applied until parameters converge.

Standard Baum-Welch algorithm

```
Initialize HMM parameters ($a_{ij}$, $\pi_i$, $\mu_i$, $\sigma_i$)
for k=1:max_iteration
    E-step : Calculate $\alpha_i(t)$, $\beta_i(t)$
        $b_j(o_t) = N(x, \mu_j, \sigma_j^2)$    (1)
        $\alpha_1(i) = \pi_i$    (2)
        $\alpha_{t+1}(j) = \left[\sum_i^N \alpha_t(i) a_{ij}\right] b_j(o_{t+1})$    (3)
        $\beta_T(j) = 1$    (4)
        $\beta_t(i) = \sum_j^N a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$    (5)
    Calculate log likelihood $L_k$ of observation sequence $o_{1:T}$
        $L(o_{1:T}) = \log\left(\sum_i^N \alpha_T(i)\right)$    (6)
    Abort if $L_k - L_{k-1} < \varepsilon$
    M-step : Estimate HMM parameters
        $\overline{\pi}_i = \alpha_1(i)\beta_1(i)$    (7)
        $\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_t^{T-1} \alpha_t(i)\beta_t(i)}$    (8)
        $\overline{\mu}_j = \frac{\sum_{t=1}^{T} \alpha_t(j)\beta_t(j) o_t}{\sum_{t=1}^{T} \alpha_t(j)\beta_t(j)}$    (9)
        $\overline{\sigma}_j^2 = \frac{\sum_{t=1}^{T} \alpha_t(j)\beta_t(j) o_t^2}{\sum_{t=1}^{T} \alpha_t(j)\beta_t(j)} - \overline{\mu}_j$    (10)
    Update HMM parameters with estimated parameters.
```

In the theoretical aspect, Baum-Welch algorithm provides means to estimate a structure (=transition matrix a_{ij}) of an unknown environment. (Chrisman, 1992) showed the experiment in a small discrete POMDP setting, but it is usually very difficult to train such a HMM without any a priori knowledge.

In most of the practical applications handling time-series data such as speech recognition and gesture recognition, left-to-right HMMs are used. Reducing the complexity of HMM structure makes training easier, but it becomes too task-specific. It is also a problem because we need to segment the data to train left-to-right HMMs. For autonomous agents, training data are neither labeled nor segmented. (Clarkson and Pentland, 1999) used segmental K-means algorithm to segment automatically the data of lifetime log, but this approach only copies the segmented data and does not capture the true structure of the entire environments.

In this paper, we took the approach to put all the data into one big HMM to self-organize the structure of the environment.

In order to overcome the difficulties of training large scale HMM, several concepts are introduced. One is to restrict the topological configuration but not as tight as left-to-right. It may be natural to assume that networks with a large number of local connections in low dimensionality can describe most of the physical systems. Therefore, we allocate the nodes of HMM in 2D square grid or 3D cubic lattice and connect each other with nodes closer than a threshold, θ_d. The distance between the adjacent nodes of the grid is defined as 1. For examples, if θ_d is set to 1, 4 adjacent nodes are connected and if θ_d is set to $\sqrt{2}$, 8 adjacent nodes are connected.

The second is to reduce the redundancy in parameters by simplifying the observation model to a single Gaussian model rather than using a mixture model.

The third is an annealing effect. Just like standard clustering problems, initial observation models have stronger influences on the result of training than transition probabilities. We set the same initial values to all μ_j and σ_j^2 so that training of transition probabilities proceeds before observation models are fixed. We also allowed μ_j and σ_j^2 in equation 9 and 10 to change gradually by introducing trace rule (Eq.11) between the iteration steps.

$$\overline{\mu}_j^{(k)} = (1-\tau)\overline{\mu}_j + \tau \overline{\mu}_j^{(k-1)} \qquad (11)$$

where τ in kth step is $\tau = k/\text{max\_iteration}$.

The last concept is time hierarchy. The long sequence of observations is sub-sampled. Coarse transitions are trained at the earlier stage and the finer transitions are trained later.

In addition to the concepts mentioned above, the scaling technique introduced in (Rabiner and Juang, 1992) to avoid underflow when calculating likelihood and state probabilities of very long sequences becomes also important.

Learning procedure

1. Initialize HMM parameters with following grid constraints and observation likelihood.

   $$a_{ij} = \begin{cases} \text{distance}(i,j) \le \theta_d : 1/n' \\ \text{distance}(i,j) > \theta_d : 0 \end{cases}$$

   n' is the normalize factor, which is number of nonzero connections in *j* th column.
   θ_d is a threshold distance in the node topology

   μ_i=0.5, σ_i = 0.05
   π_i = 1/*N*
2. Subsample every *m* th data of original observation sequence. *m*=1/*r*, where *r* is sub-sampling rate.
3. Train HMM with Baum-Welch algorithm.
4. If sub-sampling rate is 1, finish learning.
5. Reduce the sub-sampling rate. Add small values, ε to all transitions probabilities and normalize the probabilities in each column. Go to step 2 without changing other parameters.

2.3 Recognizing HMM states

After training HMM, the current state can be estimated using Viterbi algorithm (Forney, 1973). It provides the optimal estimate of hidden state sequence for a given observation sequence.

Recognition procedure

```
1. Set uniform probabilities at $t = 1$
       $\delta_1(i) = 1/N$
2. for $t$=1:$T$-1
       $\delta_{t+1}(j) = \max(b_i(o_{t+1})a_{ij}+\delta_t(i))$ for $i$
       $\psi_{t+1}(j) = \text{argmax}(b_i(o_{t+1})\, a_{ij}+\delta_t(i))$ for $i$
       Rescale at each time step
           $\delta_{t+1}(j) = \delta_{t+1}(j) / \Sigma_k(\delta_{t+1}(k))$
3. Backtrack $\delta$
       $s(T) = \text{argmax}(\delta_T(i))$ for $i$
       for $t = T-1:-1:1$
           $s(t) = \psi_{t+1}(s(t+1))$
4. Output $s(t)$ as winner node and $\delta_t(i)$ as
   probabilities of state $i$ at time $t$
```

2.4 Learning local controllers

Once the state space is known, various methods of reinforcement learning can be applied by assigning a reward on a target state. However, we do not want to acquire a single task actor.

In order to achieve multiple targets, a target state should also be a part of the state representation, which would require NxN states combinations and is not feasible. Since the state space is divided with the assumption that their transitions are sparse and local, local controllers that handle only the transitions local to a state may be easy to learn. Figure 4 shows the conceptual schema of transitions on continuous state space. All the observation data in neighboring nodes i which later make a transition to node j are used for learning local controller in node j. Transitions are expressed with the triplets of data (o_t, o_{t+1}, u_t).

For each time step, HMM recognizes current winning node i and buffers a triplet. Once the winning node turns into node j, all the buffered triplets are used to train the local controller in node j, which means that all of the data are used for training any one of the controllers. The local controllers are trained in the following functional form to output directly the action u_t from the observation.

$$u_t = f_j(o_t, o_{t+1}) \tag{12}$$

In the execution phase at time t, instead of passing o_t and o_{t+1}, we give o_t and μ_j (mean observation of node j) as inputs to the controller. As a result, a local attractor which flows into the centroid of node j is formed as figure 5. The agent can control the transition from node i to node j by calling this attractor repeatedly until

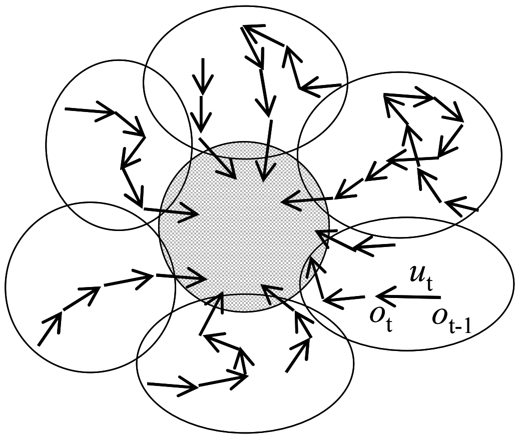

Figure 4: Transitions on state space around node j
Each arrow shows a transition from time t to t+1 with action u_t. Circle means the coverage of states state space by a certain node i. The gray circle is a destination node j.

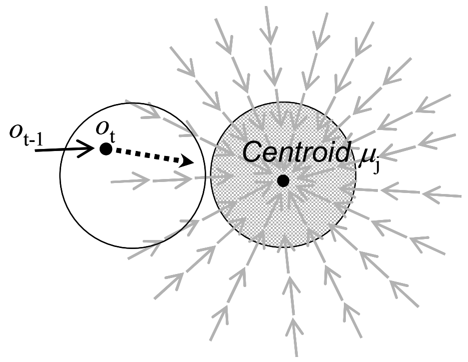

Figure 5: Local attractor formed around node j

entering to node j.

For the function estimator in equation 12, we used Accurate Online Support Regression (Ma et al., 2003) given by following equation,

$$y = \sum_i \alpha_i K(x_i, x) + b \tag{13}$$

where the following RBF kernel is used.

$$K(x_i, x) = \exp\left(-\frac{|x_i - x|^2}{2\sigma^2}\right) \tag{14}$$

2.5 Planning and execution for achieving target states

For autonomous agents, the objectives are given from the motivational system derived from intrinsic or extrinsic motivations. It somehow associates different objectives to different internal states.

Once the desired states are provided, a simple path planning is carried out using transition matrix a_{ij} to generate path from the current state to the desired state. We use Viterbi algorithm again with some modifications for use in planning. Since we do not have observation sequence for future, calculations of the observation likelihood $b_j(o_t)$ are omitted (considered them as 1.0). The values of probability in a_{ij} are also ignored and set to 1.0 for all the nonzero transitions because the innate controller might have the bias to the action and we only care whether the transitions are enabled or not. The planning procedure is described below.

Planning procedure

1. Index of the target node is d.
2. Set the probability $\delta_1(i)$ of current state to 1.0 and others to 0.0.
3. Set transition probabilities a_{ij} above ε (=0.01) to 1.0.
4. Apply probability propagation in step 2 of Recognition Procedure in section 2.3 until $\delta_t(d)$ becomes nonzero or reaching to maximum steps of propagations.
5. If $\delta_t(d)$ is nonzero, apply backtrack in step 3 of Recognition Procedure to fill the path of states. If $\delta_t(d)$ is zero, fill null path.

To execute this plan, simply local controllers on the path are invoked according to the current state recognition. The detail of the execution procedure is described as follows.

Execution procedure

1. Calculate current probabilities of all the state by Viterbi algorithm using past τ steps of observations.
2. Find maximum probability of the state along the path between a previous state and a target state and set it as a current state.
$$s(t) = \underset{s \in \text{path}(s(t-1) \rightarrow \text{target})}{\arg\max}(P(s))$$
3. If probability of $s(t)$ is below threshold, abort plan.
4. Use a controller f_j of the next state j along the path to calculate action u_t using observation o_t and state observation mean μ_j.
$$u_t = f_j(o_t, \mu_j)$$
5. Go to step 1 until reaching a target state.

If the execution is aborted, then it re-plans the path and executes again until it reaches maximum limit of repetitions. In this case, the agent aborts the target and gets a new target.

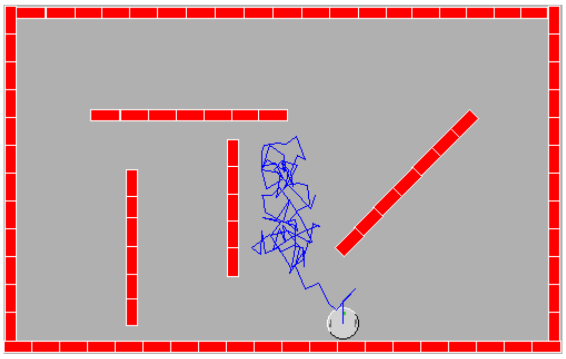
Figure 5: Khepera simulator

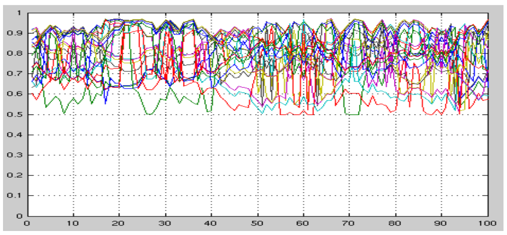

Figure 6 : observation sequence of distance sensor

3. Mobile Robot Environment

3.1 Experimental settings

For the first experiment, we used a mobile robot environment using Khepera simulator (Michel, 1996) for a two-wheel robot in maze-like room.

There are several modifications to the original simulator. The distance sensors measure the relative distance from the robot to the nearest wall. The number of distance sensors is extended to 24 from original of 8. Sensors surround a robot 15 degrees apart from each other. The range of them is extended to 500 from the original of 60. The action to the robot becomes δX, δY instead of a wheel command, δL and δR. δ$X_{\max}$=10, δ$Y_{\max}$=10 for action commands. The size of the room is about 600x1000. The light sensors are not used in this task.

The agent is connected to this simulated environment through 24 sensor inputs and 2 action outputs whose ranges are normalized to fit between 0.0 and 1.0, and no a priori knowledge about this environment is given.

In the standard robot navigation task, state representations such as robot positions (X,Y) and an environmental map (Occupancy grids), an observation model (how walls are observed given a map), and a motion model (how positions changes by action

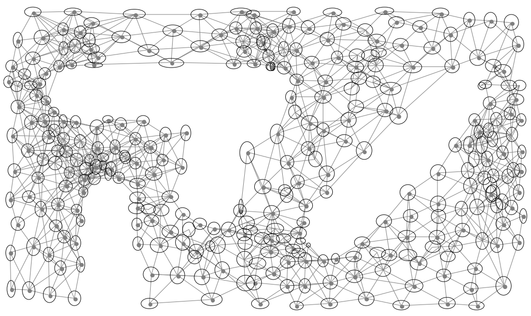
Figure 7: HMM nodes and their transitions
A circles represents a HMM node. They are plotted on average of ground truth robot positions while the node is active. Lines connecting the nodes are indicating the transitions with nonzero probabilities.

commands) are given. The objective of the task is to build a map and control positions.

In our framework, the agent knows nothing mentioned above, instead, it has general modules namely the predictor, the controller, and the planner. It has to find out everything through the responses of the sensors from its actions. We are going to show that even with these challenging settings, a navigation skill emerges as a consequence of learning the environmental structure.

The random innate controller is used to generate actions in training (babbling) phase, in which action δX, δY are generated from uniform distribution in the range of $[-\delta X_{\max}, \delta X_{\max}]$ and $[-\delta Y_{\max}, \delta Y_{\max}]$ at every time step. An example of the trajectory from such actions is shown in the figure 5. An example of 24 dimensional distance sensor measurements is shown in the figure 6.

3.2 *Result of learning HMM*

HMM is trained with the observation sequence generated by the innate controller with the following parameters.

Parameters
Length of the data: 10,000 steps
Number of nodes (2D Grid): 20x20=400
Number of neighbor connections: 12 (θ_d=2.0)
Max. EM iterations: 200
Subsample rate: 1/10, 1/5, 1/2, 1
Min. observation variance σ^2 : 0.01^2

As a result, a topological configuration in Fig. 7 is learned. When compared with figure 5, it is clear that each state in HMM represents a location in the simulated environment. The global structure of the room was captured very well even from the partially observed distance data, which are similar in different places of the environments.

The nodes, which have never been used throughout the learned sequence, are removed. Out of 400 nodes initially prepared 309 nodes are remained. Out of 10,000 transitions initially assigned to nonzero, 2243 transitions remained nonzero.

3.3 *Result of learning local controllers*

After learning the HMM, same sequence is used to recognize state transitions. Using this state transitions, every triplets (o_t, o_{t+1}, u_t) in the sequence can be assigned to one of the local controllers for learning.

Figure 8 shows the result of attractors formed at two different nodes. The zoomed figure corresponds to particular positions on the simulated environment and center of the figure is the centroid of the node. It seems that in both cases all the movement of the robot is heading towards the centroid of the node.

3.4 *Testing the acquired behaviors*

Using learned HMM and local controllers, the tests are carried out to examine what kinds of behaviors are acquired in the agent. The agent is randomly given a target state which is one of the learned states. The planning and execution phase described in the previous section is carried out for that given target state.

Figure 10 shows the snapshot of execution. The target node pointed by the arrow was given to the agent. The node near the upper right corner which was the other end of the connected line, was the state that agent recognized when it made the plan to the target. The connected line is the planned path. The agent invoked the controller along path and reached the half way of the path. The simulated robot moved from right to left corresponding to the change of the node recognition.

To show the performance quantitatively, we measured the success rate of the tasks. A task is given by selecting one of the states randomly. A single trial finishes either the agent gives up or reach a target. The next task starts from the last state in the previous trial.

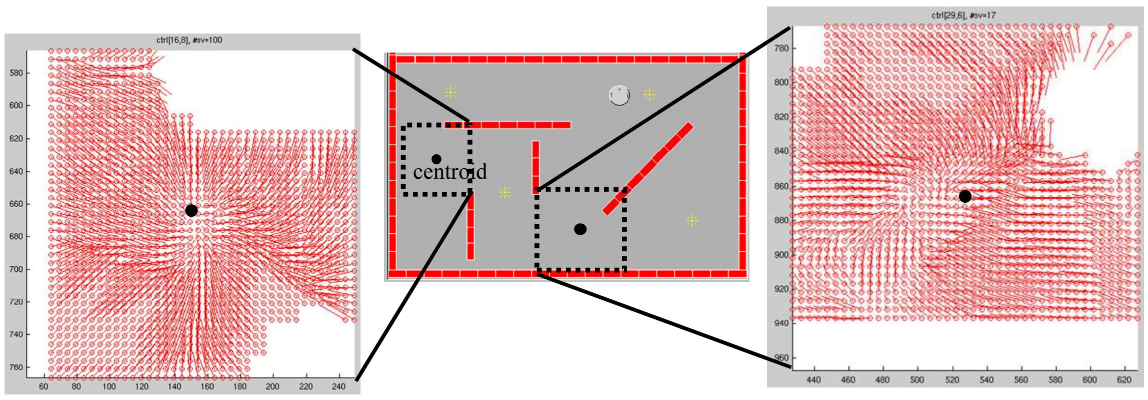

Figure 8: Result of attractors formed around HMM nodes
Circles in zoomed figures are grid positions on simulator. Each arrow represents robot motion caused at the circle using the action from local controllers.

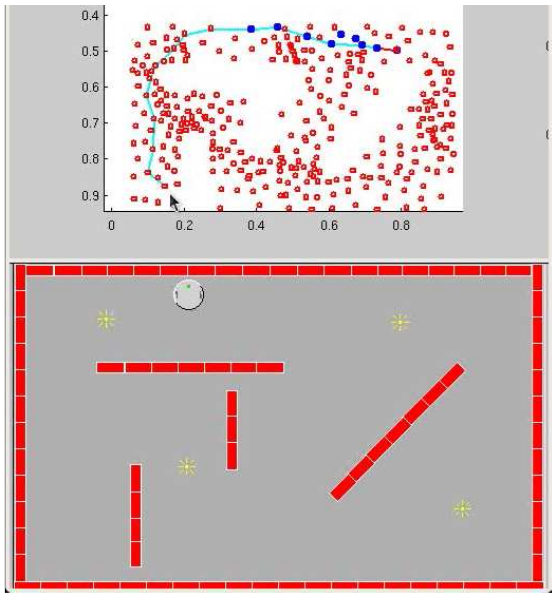

Figure 9 : snapshot of execution
Lines connecting the nodes are the path. Filled circles are the nodes activated recently. Lower part of the figure is the simulated robot moving right to left along the corridor

Out of 118 trials, 108 times were successful in reaching the target, which means the agent thought it had reached the target. The success rate was 95.6%.

We showed that after learning the structure of the environment and learning to control arbitrary transitions, the optimal combinations of these local controls to achieve desired states become some meaningful and appropriate behaviors even no task specific objectives are given in the learning framework.

4. Robot Manipulator Environment

4.1 Experimental Settings

We have also tested the agent on a different environment, which is a simple robot manipulator shown in Figure 10(a). We have used the same physical parameters described in (Doya et al., 2002). It has 1DOF joint with some friction. The torque of the joint is set weaker than gravity so that it needs to swing back and forth to be in a desired position. The task of swinging up to the top position is often used as the standard reinforcement-learning problem. Since the state of this dynamical system can be fully described with angle θ and angular velocity ω, a torque controller that maximize the reward which is given at the top position can be learned in the state space(θ, ω).

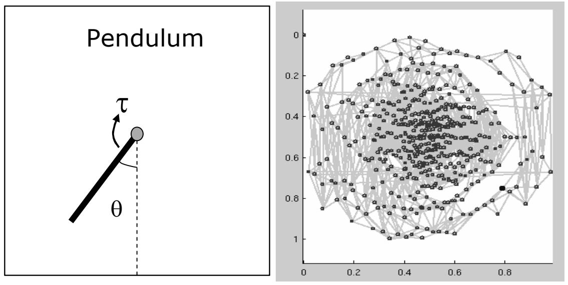

Figure 10: (a) Pendulum (b) Learned HMM
In (b), states in HMM is plotted on ground truth of (θ, ω) space

In our environmental settings, no reward is given and angular velocity ω is hidden. Even with these difficult settings we are going to show that not only the swing-up but also the ability to control to arbitrary states can be learned with our proposed methods.

For the innate controller, we design again a random torque generator that switches torque from one of three values $\{-\tau_{\max}, 0, \tau_{\max}\}$ every 30 time steps.

HMM is trained using the one dimensional observation sequence of angle θ. Almost same learning parameters are used for this environment.

Parameters

Length of the data: 10,000 steps
Number of nodes (2D Grid): 22x22=484
Number of neighbor connections: 12 (θ_d=2.0)
Max. EM iterations: 200
Subsample rate: 1
Min. observation variance σ^2 : 0.01^2

4.2 Results

As a result, a graphical representation in figure 10(b) is learned. Same as the previous experiment, each node is plotted on the ground truth of the system state (θ, ω). It captures very well of the hidden structure of pendulum dynamics. Local controllers are also learned for this task. The execution tests are performed with random selections of the targets out of the learned states.

Figure 11 shows the snapshot of the execution. The left figure is the plot of the node in (θ, ω) state space and the center is θ=0, ω=0. The created path goes around the center, which means that it swings back and forth to go down to the lowest position with highest speed. Green circles are ground truth trajectory of the pendulum and blue are the estimated nodes. Some recognition results come off the path but the actual pendulum is controlled well on the planned path.

5. Conclusion

We have proposed a novel framework of reward-free learning through which the agent can acquire appropriate behaviors or the skill suitable for the environment without any task specific objectives. Two experimental results are shown in which the agent acquired navigation and manipulation skills respectively.

We have also shown that learning the entire

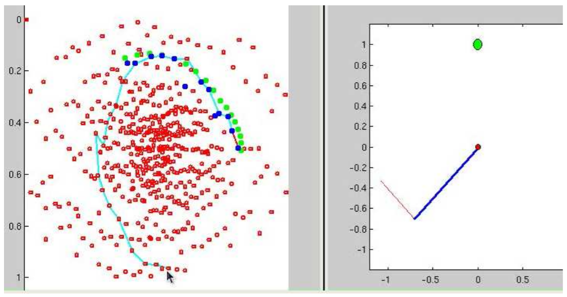
Figure 11 : snapshot of execution

observation sequence with single large sparsely-connected HMM can be possible and it enables the self-organization of hidden structures of environments.

The roles of skill formation and intrinsic motivation are important factors in open-ended learning. In previous works (Sutton, et al., 1999), (Barto et al., 2004), (Oudeyer, P.-Y et al., 2007) dealing with these functions, the main focuses are put on implementing individual mechanisms in which states or state space are predefined and goals and sub-goals are predetermined. Our work provides the outer framework integrating these functions to achieve open-ended learning. It is important for agents to create and to solve problems other than given ones.

There seems to be many extensions needed to cope with larger scale problems, especially hierarchy and recursiveness are important factors to implement. There are hierarchical approaches to improve the efficiency of training and data representation such as Hierarchical HMM (Fine et al., 1998), Layered HMM (Oliver et al., 2004). By applying the knowledge from such systems, the extension may not be so difficult.

References

Barto, A. G., Singh, S., and Chentanez, N. (2004). Intrinsically motivated learning of hierarchical collections of skills. *Proceedings of the Third International Conference on Developmental Learning, pp. 112-119.*

Baum, L.E., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Ann. Math. Statist., vol. 41, no. 1, pp. 164--171.*

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. *In Proceedings of the Tenth International Conference on Artificial Intelligence, pages 183-188. AAAI Press, San Jose, California.*

Clarkson, B. and Pentland, A. (1999). Unsupervised Clustering of Ambulatory Audio and Video. *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing, IEEE CS Press, vol. 6, pp. 3037-3040.*

Csikszentmihalyi, M. (1990). Flow: The psychology of optimal experience. *New York: Harper and Row.*

Doya, K., Samejima, K., Katagiri, K., and Kawato, M. (2002). Multiple model-based reinforcement learning. *Neural Comput. 14, 6, 1347-1369.*

Elfes, A. (1989). Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation. *PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University.*

Fine, S., Singer, Y., and Tishby, N. (1998). The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning, vol. 32, p. 41-62.*

Forney, G. D. (1973). The Viterbi algorithm. *Proceedings of the IEEE 61(3), 268–278, March 1973.*

Fujita, M. (2009). Intelligence Dynamics: a concept and preliminary experiments for open-ended learning agents. *Autonomous Agents and Multi-Agent Systems*.

Leonard, J. J., Durrant-Whyte, H. F. (1991). Simultaneous mapbuilding and localization for an autonomous mobile robot. *Proceedings of IEEE Int. Workshop on Intelligent Robots and Systems: 1442-1447.*

Ma, J., Theiler, J., and Perkins, S. (2003). Accurate on-line support vector regression, *Neural Computation Vol.15, Issue 11 pp.2683-2703.*

Michel, O (1996). Khepera Simulator Package version 2.0. *Freeware Khepera simulator. It can be downloaded from* http://diwww.epfl.ch/lami/team/michel/khep-sim/

Oliver, N., Garg, A., and Horvitz, E. (2004). Layered Representations for Learning and Inferring Office Activity from Multiple Sensory Channels. *Computer Vision and Image Understanding (CVIU), 96, pp. 163-180.*

Oudeyer, P.-Y., Kaplan, F., Hafner, V.V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation, Special Issue on Autonomous Mental Development, 11 (1), pp. 265-286.*

Sutton, R. S.., Precup, D., and Singh, S. (1999). Between MDPSs and semi-MDPs: A framework for temporal abstraction in reinforcement Learning, *Artificial Intelligence 112, pp.181—211.*

Rabiner, L. and Juang, B.-H. (1993). Fundamentals of Speech Recognition. *Prentice Hall Signal Processing Series.* Prentice Hall, Englewood Cliffs, NJ.

Sabe, K., Hidai, K., Kawamoto, K., and Suzuki, H. (2005). A proposal of intelligence model, MINDY for open-ended learning system. *Proceedings of IEEE International Conference on Humanoid Robots Workshop on Intelligence Dynamics.*

Shatkay, H. and Kaelbling, L. P. (1997). Learning topological maps with weak local odometric information. *Proceedings of Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97), pp. 920-929.*