Sensor Space Segmentation for Mobile Robot Learning

Yasutake Takahashi, Minoru Asada, Shoichi Noda[*] and Koh Hosoda
Dept. of Mech. Eng. for Computer-Controlled Machinery
Osaka University, 2-1, Yamadaoka, Suita, Osaka 565, Japan
asada@robotics.ccm.eng.osaka-u.ac.jp

Abstract

Robot learning such as reinforcement learning generally needs a well-defined state space in order to converge. However, to build such a state space is one of the main issues of the robot learning because of the inter-dependence between state and action spaces, which resembles to the well known "chicken and egg" problem. This paper proposes two methods of action-based state space construction for vision-based mobile robots. Basic ideas common to the two methods to cope with the inter-dependence are that we define a state as a cluster of of input vectors from which the robot can reach the goal state or the state already obtained by a sequence of one kind action primitive regardless of its length, and that this sequence is defined as one action. The first method clusters the input vectors as hyper ellipsoids so that the whole state space is segmented into a state transition map in terms of action from which the optimal action sequence is obtained. In order to obtain the such a map, we need a sufficient number of data, which means longer learning time. To cope with this, we proposed the second method by which a robot learns purposive behavior within less learning time by incrementally segmenting the sensor space based on the experiences of the robot. The incremental segmentation is performed by constructing local models in the state space, which is based on the function approximation of the sensor outputs to reduce the learning time and on the reinforcement signal to emerge a purposive behavior. To show the validity of the methods, we apply them to a soccer robot which tries to shoot a ball into a goal. The simulation and real experiments are shown.

INTRODUCTION

Building a robot that learns to perform a task has been acknowledged as one of the major challenges facing Robotics and AI (Connel & Mahadevan 1993). Recently, reinforcement learning (Watkins & Dayan 1992; Sutton 1992) and memory-based learning (Connel & Mahadevan 1993) have been receiving increased attention as a method for robot learning with little or no *a priori* knowledge and higher capability of reactive and adaptive behaviors. In these robot learning methods, a robot and an environment are generally modeled by two synchronized finite state automatons interacting in a discrete time cyclical processes. The robot senses the current state of the environment and selects an action. Based on the state and the action, the environment makes a transition to a new state and generates a reward that is passed back to the robot. Through these interactions, the robot learns a purposive behavior to achieve a given goal.

To apply robot learning methods such as reinforcement learning to real robot tasks, we need a well-defined state space by which the robot learns to select an adequate action for the current state to accomplish the task at hand. Traditional notions of state in the existing applications of the reinforcement learning schemes fit nicely into deterministic state transition models (e.g. one action is forward, backward, left, or right, and the states are encoded by the locations of the agent). However, it seems difficult to apply such deterministic state transition models to real robot tasks. In real world, everything changes asynchronously (Matataric 1994).

Generally, the design of the state space in which necessary and sufficient information to accomplish a given task is included depends on the capability of agent actions. On the other hand, the design of the action space also depends on the capability of perception. This resembles the well-known "chicken and egg problem" that is difficult to be solved (see Figure 1).

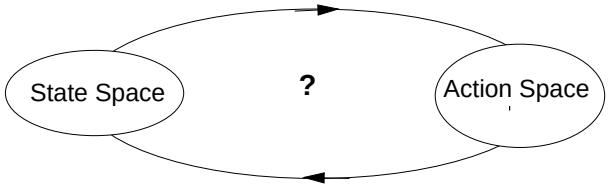


Figure 1: The inter-dependence between state and action spaces

One can construct a state space fixing the action s-

[*]Currently, he is with Hitachi Co.

pace first. Chapman and Kaelbring (Chapman & Kaelbling 1991) proposed an input generalization method which splits an input vector consisting of a bit sequence of the states based on the already structured actions such as "shoot a ghost" and "avoid an obstacle." However, the original states have been already abstracted, and therefore it seems difficult to be applied to the continuous raw sensor space of real world.

Dubrawski and Reingnier (Dubrawski & Reingnier 1994), and Kröse and Dam(Kröse & Dam 1992) proposed methods similar to each other which abstracted sonar information into the form useful for mobile robots to avoid obstacles. Ishiguro et al. (Ishiguro, Sato, & Ishida96 1996) dealt with a problem of state space categorization by statistically analyzing the sensor patterns, actions, and rewards given at the end of goal achievement. Since they deal with reflexive behaviors such as obstacle avoidance, these methods do not suffer from the fixed length physical actions. However, in case of a task to achieve the goal farther from the viewpoint based on the visual information, the same physical actions might cause different changes in image, and therefore it seems difficult to specify the state and action spaces by which learning converges correctly [1].

Asada et al. (Asada *et al.* 1995a) called this "a state-action deviation problem" due to the difference in resolution between the robot action in a 3-D space and the projection of its effect onto a 2-D image. They have given one solution for this problem by restructuring the action space so that one action may cause one state transition. That is, first they divided the sensor space by hand, and then constructed the action space so that the sensor and action spaces can be consistent with each other. However, there is no guarantee that such a state space is always appropriate for the robot.

This paper propose two methods that recursively split an input vector from the sensor based on a definition of action primitive that is a a motor command executed during the fixed time interval. The basic ideas common to the both of two methods are as follows: We define

1. a state as a set of input vectors from which the robot achieves the goal or already acquired state by a variable sequence of one kind action primitive, and
2. an action as a sequence of action primitive that causes a state transition.

The first method clusters the input vectors as hyper ellipsoids so that the whole state space is segmented into a state transition map in terms of action from which the optimal action sequence is obtained. In order to obtain the such a map, we need a sufficient number of data, which means longer learning time. To reduce the learning time, we proposed the second method by which a robot learns purposive behavior within less learning time by incrementally segmenting the sensor space based on the experiences of the robot. The incremental segmentation is performed by constructing local models in the state space, which is based on the function approximation of the sensor outputs to reduce the learning time and on the reinforcement signal to emerge a purposive behavior.

The remainder of the article is structured as follows: In the next section, we briefly review the reinforcement learning, and the problem of the state and action space construction, then explain the first method of state space segmentation. Next, we explain the second method. The both methods are described along with the experimental results by the computer simulations and the real robot system and conclusions.

BASICS OF REINFORCEMENT LEARNING

Before getting into the our method, we briefly review the basics of the reinforcement learning.

We assume that the robot can discriminate the set $\boldsymbol{S}$ of distinct world states, and can take an action from the action set $\boldsymbol{A}$. The world is modeled as a Markov process, making stochastic transitions based on its current state and the action taken by the robot. For each state-action pair (s, a), the reward $r(s, a)$ is defined.

The general reinforcement learning problem is typically stated as finding a policy that maximizes discounted sum of the reward received over time. Watkins' Q-learning algorithm (C.J.C.H.Watkins 1989) gives us elegant method for doing this.

In the Q-learning algorithm, the robot takes an action $a \in \boldsymbol{A}$ in a state $s \in \boldsymbol{S}$ and transits to the next state $s' \in \boldsymbol{S}$, then it updates the action-value function $Q(s, a)$ as follows.

$$Q(s,a) \Leftarrow (1-\alpha)Q(s,a) + \alpha(r(s,a) + \gamma \max_{a' \in \boldsymbol{A}} Q(s',a')) \tag{1}$$

where α is a learning rate and γ is a discounting factor.

After a sufficient number of trials, the action a which maximize the $Q(s, a)$value is the optimal decision policy at the state s.

SENSOR AND ACTION SPACE CONSTRUCTION

As we described in the above, the state space[2] designed by programmer is not always appropriate for the robot to accomplish a given task. If multiple states to be discriminated from each other are categorized into the same state, the distribution of that state transitions widely spreads out, and therefore it seems difficult for the robot to achieve the goal. On the other hand, if

[1] In case of vision sensors, the same action might cause large change in image if the object is close to the observer, and small change if it is farther.

[2] Here, we suppose the state space is a space consisting of input vector from sensors. In control theory, this is not always true.

the size of the state space is too large due to unnecessary separations, the learning incredibly takes long time (it is generally an exponential order in the size of the state space (Whitehead 1991)). Then, we attempt at solving this problem by making the robot construct the state space, that is, it should find a state space by itself through interactions with the environment. The following are the requirements for the problem:

1. The state and action spaces should reflect physical sensor(s) and actuator(s) of a robot. The deterministic state transition models (e.g. one action is forward, backward, left, or right, and the states are encoded by the locations of the agent) are useful only for simple toy problems in computer simulations.
2. Since everything changes asynchronously in real world (Mataric 1994), the state and action spaces directly reflecting the physical sensors and actuators suffer from the state - action deviation problem. The state and action spaces should be restructured to cope with this problem.
3. The sensor space categorization should be robust against the various disturbances such as sensor noise, delay, and uncertainty of action execution.

BASIC IDEA

Basic ideas of our method are that we define:

1. an action primitive a_i $(i = 1, 2, ..., n)$ as a resultant action caused by a motor command executed during the fixed time interval,
2. a state as a set of input vectors from which the robot achieves the goal or already acquired state by a variable sequence of one kind action primitive, and
3. an action as a sequence of action primitive that causes a state transition,

and that such states are found in the order of closeness to the goal state in the first method, or such states are constructed by function approximation of the state changes in the second method.

THE FIRST METHOD

Figure 2 shows a state space in terms of the goal state and actions. S_1^i, $i = a_1, a_2, a_3, ..., a_n \in \mathbf{A}$ (a set of action primitives) and $\mathbf{S_1}$ denote a state from which the robot can achieve the goal by only one action i, and a set of these states, respectively. Further, $\mathbf{S_2}$ denotes a set of states from each of which the robot can achieve $\mathbf{S_1}$ only by one action. Similarly, $\mathbf{S_j}$ denotes a set of states from which the robot can achieve the goal at least j actions. Any input vector from which the robot can achieve the goal can be included in any state in $\mathbf{S_k}$ $(k = 1, 2, .., j)$. The algorithm to obtain such a state space is given below.

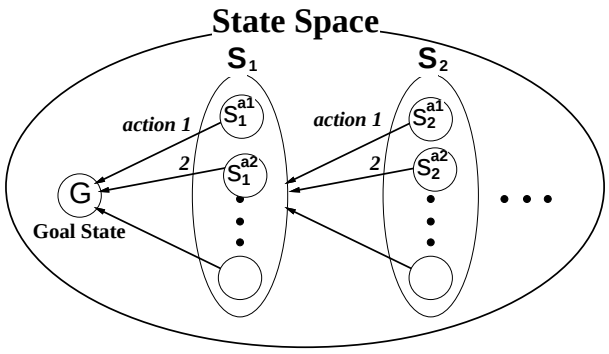


Figure 2: The goal-directed and action-based state space construction

Algorithm for the first method

1. Set the goal state as a target zone $\boldsymbol{Z}_T$.
2. Take an action randomly. From the definition, the same action primitive is iteratively executed until the robot achieves the target zone or the fixed time interval expires.
3. Store an input vector $\boldsymbol{x} \in \boldsymbol{R}^m$ (m: the size of the vector) with an index of the action $a \in \boldsymbol{A}$ the robot took when it could succeed in achieving $\boldsymbol{Z}_T$ from $\boldsymbol{x}$. Do not include the vectors that have been already categorized into the previously divided states.
4. Fit a multi-dimensional uniform distribution function (a concentration ellipsoid (Cramër 1951)) to a cluster of stored vectors with the same action index $a_i \in \boldsymbol{A}$ obtained above, and construct a state s_{a_i} $(a_i \in \boldsymbol{A})$. The boundary surface of the ellipsoid is given by:

$$(\boldsymbol{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu}) = m + 2, \qquad (2)$$

   where μ and Σ denote the mean vector and the covariance matrix, respectively.
5. Update the target zone $\boldsymbol{Z}_T$ as a union of states s_{a_i} $(i = 1, 2, .., n)$ obtained in the previous step. If a vector is categorized into plural states s_{a_j} $(j = 1, ..)$ (clusters are overlapped), select one state so that the following distance normalized by its covariance can be the minimum:

$$\Delta_j = (\boldsymbol{x} - \boldsymbol{\mu}_j)^T \Sigma_j^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_j)$$

6. Stop if the state space is almost covered by the divided clusters. Else, go to 2.

We call a set of states s_a $(a \in \boldsymbol{A})$ the i-th closest to the goal state $\mathbf{S_i}$. By the above algorithm, we can obtain not only the sensor space categorization but also the optimal path to the goal from everywhere.

Experimental Results and Remarks

Simulation (I)

To show the validity of the proposed method, we show a simple computer simulation in toy world consisting of 100×100 grids. The task for the robot is to enter the circle area whose radius is 5, located at the center of the world. The action primitives are 1.0 grid motion into any of four directions (up, down, left, and right). The input vector is an absolute coordinate (x, y) (real number) of the robot location.

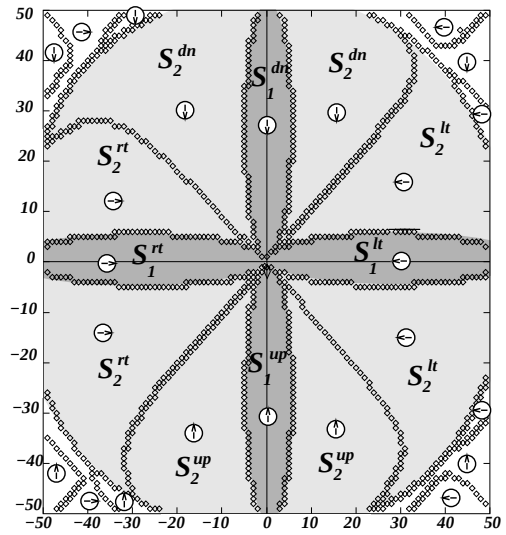


Figure 3: Result of the variable action-based construction of 2D state space

Figure 3 shows a result where the whole state space is separated into twelve states. The states obtained earlier have darker colors. Each arrow indicates an action by which the state is classified. A set of states $\mathbf{S_1}$ from which the robot can achieve the goal by only one action consists of S_1^{up}, S_1^{dn}, S_1^{lt}, and S_1^{rt} corresponding to upward motion and so on. A set of states $\mathbf{S_2}$ from which the robot can achieve $\mathbf{S_1}$ by only one action consists of four ellipsoids S_2^{up}, S_2^{dn}, S_2^{lt}, and S_2^{rt} which are projected onto eight regions in Figure 3.

This figure can be interpreted as follows: the top-left region S_2^{dn} indicates downward motion, and the robot takes this action. Then, the robot seems to enter a state S_2^{rt} of which action is rightward. However, this state is classified into the same $\mathbf{S_2}$, therefore, the robot continues to take the same action (downward) until entering the state S_1^{rt} belonging to $\mathbf{S_1}$, and state transition occurs[3] This means that the same input vector, the absolute coordinate of the robot location here, might be different depending on its action. We sometimes have a similar situation that we failed to recognize the intersection which we usually cross over from the certain direction when we cross it from the different direction.

The top-left quarter in the figure can be originally either downward or leftward motion, and accidentally classified into one by selecting the shorter one of the distances from the center of two ellipsoids.

Let us consider to apply the existing methods such as (Dubrawski & Reingnier 1994; Kröse & Dam 1992) to the same problem. Since the length of one action is fixed in these methods, the size of the obtained state space depends on the length of one action and/or the size of the world. However, the size of the state space would not be so affected by these parameters in our method since the length of one action can be variable. This suggests that if the topology of an environment is similar to another, the state spaces obtained by our method are not so different regardless of the absolute size of the environment.

Simulation (II)

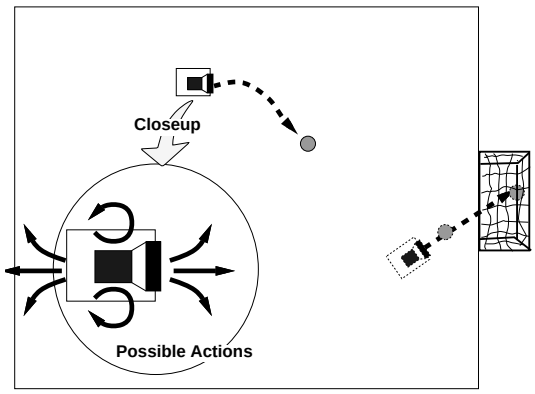


Figure 4: Task

Task and environment As a more complicated task for the robot, we consider an environment shown in Figure 4 where the task for a mobile robot is to shoot a ball into a goal (same as in (Asada *et al.* 1995a)). The environment consists of a ball and a goal, and the mobile robot has a single TV camera. The robot does not know the location and the size of the goal, the size and the weight of the ball, any camera parameters such as focal length and tilt angle, or kinematics/dynamics of itself.

We performed the computer simulation with the following specifications. The field is a square of 3.0m $\times$ 3.0m. The goal post is located at the center of the top line of the square (see Figure 4) and its height and

[3]Even if the robot changes its actions when it crosses the state boundary between S_2^{dn} and S_2^{rt}, the consequent actions are iterations of downward and leftward ones, and finally it achieves the goal state with physically the same steps.

width are 0.23m and 0.9m, respectively. The robot is 0.31m wide and 0.45m long and kicks a ball of diameter 0.09m. The maximum translation velocity is 1.1m/s, and the maximum angular velocity is 4.8 rad/s. The camera is horizontally mounted on the robot (no tilt), and its visual angle is 36 degrees. The velocities of the ball before and after being kicked by the robot is calculated by assuming that the mass of the ball is negligible compared to that of the robot. The speed of the ball is temporally decreased by a factor 0.8 in order to reflect the so-called "viscous friction." The values of these parameters are determined so that they can roughly simulate the real world.

The robot can select an action to be taken in the current state of the environment. The robot moves around using a PWS (Power Wheeled Steering) system with two independent motors. Since we can send the motor control command to each of the two motors separately, we construct the action primitives in terms of two motor commands ω_l and ω_r, each of which has 3 sub-actions, forward, stop, and back. All together, we have 9 actions in the action primitive set $\boldsymbol{A}$. Actually, a stop motion does not causes any changes in the environment, we do not select this action primitive.

In computer simulation, we take into account two sources of disturbances which make the method unstable. They are delays due to sensory information processing and uncertainty of action execution. The contents of the image processing are color filtering (a ball and a goal are painted in red and blue, respectively), edge enhancement, localizing and counting edge points, and vector calculation (Asada *et al.* 1995a). We have been using a pipeline image processor for the real robot experiments and it takes about 33 ms to perform these processes, that is, a period of one action primitive. The latter is caused by the delay necessary to stabilize motor rotation after sending motor commands, and it is about 100 ms. Therefore, the uncertainty of the action execution increases when motor commands often change.

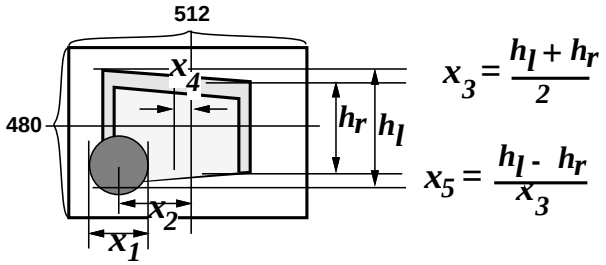
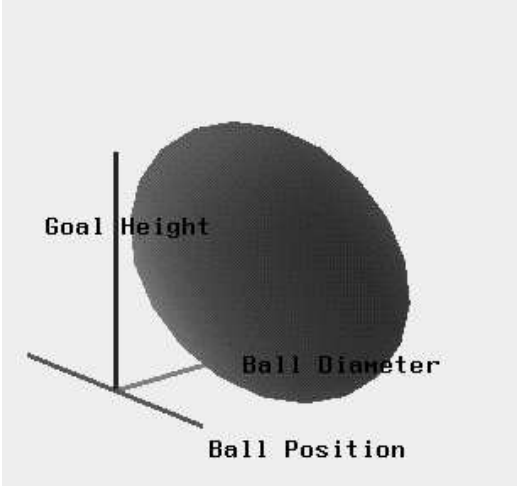
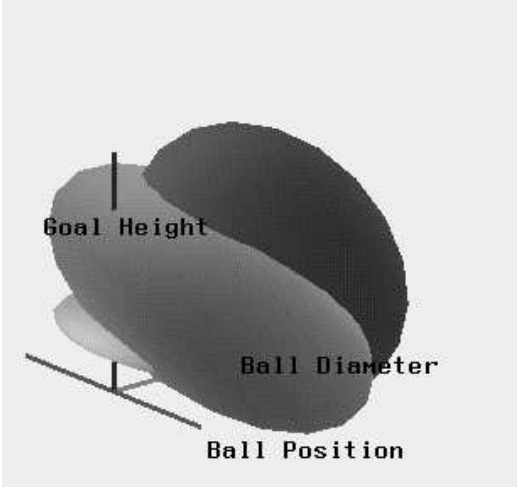


Figure 5: Input vector consisting of five parameters



(a) the first stage ($S_1^F = \mathbf{S_1}$)



(b) the second stage ($\mathbf{S_1} + \mathbf{S_2}$)

Figure 6: Construction process of the state space for the shooting robot

Results The size of the observed image is 512 by 480 pixels, and the center of image is the origin of the image coordinate system (see Figure 5). An input vector $\boldsymbol{x}$ for a shooting task consists of:

- x_1: the size of the ball, the diameter that ranges from 0 to about 270 pixels,
- x_2: the position of the ball ranging from -270 to +270, considering the partial observation,
- x_3: the size of the goal, the height average of the left and right poles (or ends in image) that ranges from 0 to 480 pixels,
- x_4: the position of the goal, the average of the positions of the left and right poles (or ends in image)

that ranges from -256 to +256, and

- x_5: the orientation of the goal, the ratio of the height difference between the left and right poles (or ends) to the size of the goal x_3. x_5 ranges from -1.00 to +1.00.

As a result of the state space construction, we can obtain the state transition graph simultaneously. Actually, we have other states than obtained by the method such as "only a ball is observed" or "only a goal is observed." State transitions from these states to the obtained states is possible if they can be realized by only one action. Otherwise, it seems difficult to find a path because the robot might have many hidden states during the desirable state transition. For example, the robot exists between a ball and a goal, and the robot must take a circular motion so that it can get a position from where the ball and the goal can be observed simultaneously. During such a motion, the robot might have many hidden states. We do not deal with the "hidden states" problem here.

Figure 6 shows the process of state space division. The state space in terms of ball size, ball position, and goal size is indicated when the position and the orientation of the goal (x_4 and x_5) are both zeros (in front of the goal). In the first step, only one big ellipsoid ($\mathbf{S_1^F}$) is obtained that corresponds to the forward motion (Figure 6 (a)). In the second step, two ellipsoids (S_2^F and S_2^B) corresponding to forward and backward motions, respectively, are obtained and they construct ($\mathbf{S_2}$) (Figure 6 (b)). In this figure, the state space designed by the programmer in (Asada *et al.* 1995a) corresponds to blocks parallel to the axes which are quite different from the obtained ones.

Table 1: Comparison with existing methods

	Number of States	Search Time	Success Rate (%)
Previous work	243	500M*	77.4
Proposed method	33	41M	83.3

* indicates Q-learning time.

Table 1 compares the method with existing ones. Success rates are obtained from 5000 trials for each, and the number of states are counted when both the ball and the goal are observed. The search time in the previous work (Asada *et al.* 1995a) means the learning time in terms of the period of one action primitive (33 ms). It takes about 500M (M=10^6) ticks because the size of the state space is much larger. The proposed method performs better than the previous work. The reductions of the size of the state space and the search time are about 1/8 and 1/12 of the previous work, respectively. For the reference, we show the result by the fixed action length of 33 ms. Compared with the previous work (Asada *et al.* 1995a), the size of the state space and the search time are reduced into the half, but the success ratio has not been improved because the simulation has been done taking into account the delays of image processing (33 ms) and the latency of motor rotation (about 100 ms), and these effects occur when state changes, the number of which is so many due to the fixed length action. While, the size of the state space by the proposed method is small, and the size of each state is considerably larger, which is preferable for the stability of the control because the effect of sensor information processing delay becomes negligible and the stability of motion execution improves due to no changes of action commands inside one state. Only the problem due to large volume of each state is that the possibility of the incorrect merging of input vectors into wrong states seems high. This might be partly a reason why the success rate is less than 90%.

Experiments with a real robot

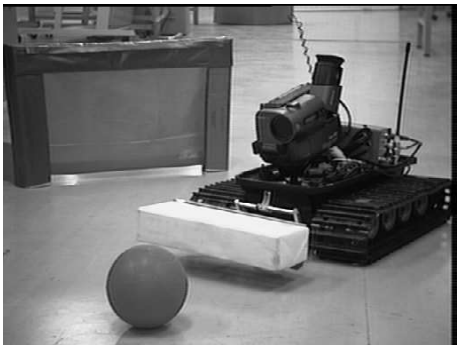


Figure 7: Our real robot

Figure 7 shows a picture of the real robot with a TV camera (Sony handy-cam TR-3) used in the real experiments. The system used in the experiments is the same one in (Asada *et al.* 1995a). The experiment consists of three stages. First, we collect the data obtained by real robot motions. Next, we construct the state and action spaces based on the sampled data. Finally, we control the robot based on the acquired the state and action spaces and their transition graph.

The number of sampled input vectors is about 20,000 which corresponds to about 10 minutes. Figure 8 indicate the projection of the constructed state space in terms of the sizes of the ball and goal (x_1 and x_3) when the ball and the goal are observed at the center of image ($x_2 = x_4 = x_5 = 0$). Labels "F" and "B" indicate the motions of forward and backward, respectively, and subscript shows the number of state transitions towards the goal. "FL" means left forward motion. Due to the capacity of the image processor, the image size is reduced into 1/16 (128×120), and values of each components of the input vector is also reduced into 1/4. The whole state is separated into 10 states, which is smaller than in simulation because of smaller number of experiences.

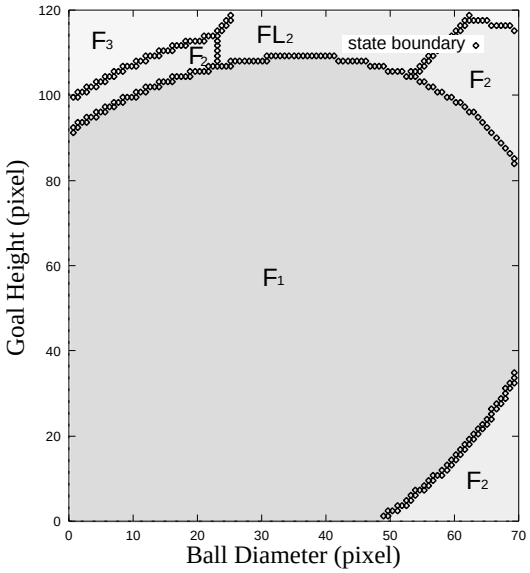


Figure 8: Result of state space construction for the real robot

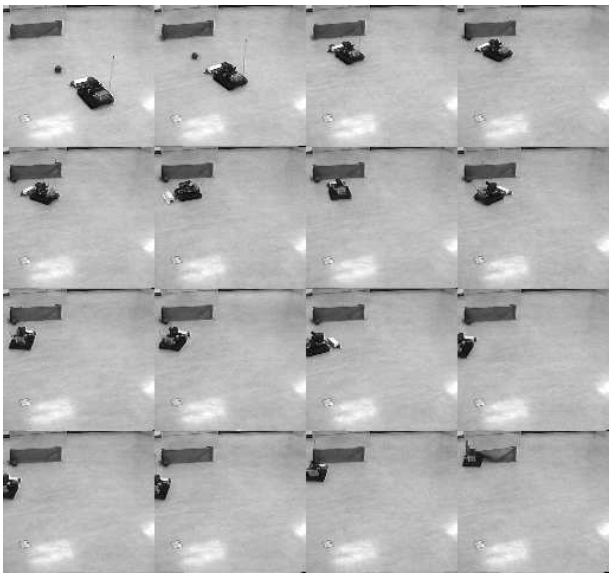


Figure 9: The robot succeeded in finding and shooting a ball into the goal

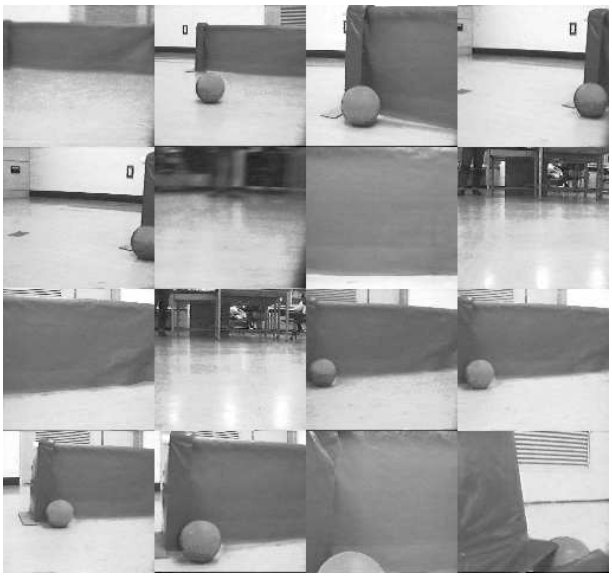


Figure 10: Images taken by the robot during the task execution

We applied the result to a real robot. Success ratio is worse than the simulation because of the disturbances due to several causes such as eccentricity of the ball centroid and slip of the tires that make the ball or the robot move into unpredictable directions. Figure 9 shows how a real robot shoots a ball into a goal by using the state and action map obtained by the method. 16 images are shown in raster order from the top left to the bottom right in every 1.5 seconds, in which the robot tried to shoot a ball, but failed, then moved backward so as to find a position to shoot a ball, finally succeeded in shooting. Figure 10 shows a sequence of images taken by the robot during the task execution shown in Figure 9. Note that the backward motion for retry is just the result of learning and not hand-coded.

Discussion

There are two kinds of trade offs:

- If the sampling data (input vectors) are biased, parameters of the ellipsoids change, which affects the size of the state space and the search time, and as a result a behavior also changes. In our method, we randomly initialized robot positions so that it can observe both ball and goal. However, there is no guarantee of no biases. Generally, for the less biased data, the more data and longer time are necessary. An effective method for data sampling should be developed, but there is a trade-off between the effectiveness and *a priori* knowledge on the environment and the robot.
- We used a concentration ellipsoid (Cramër 1951) as a model of cluster (state) of input vectors, inside which

a uniform distribution is assumed. However, actual distributions are not always uniform. Ideally, situations that input vectors to be included in the model are not included and vise versa should be avoided. Complete modeling seems impossible because of uncertainties in real world. A model which includes much less error but needs complicated procedures to obtain the model parameters and/or much memory to store and recall them is not desirable because of realtime execution of robot actions. We should find a model taking into account these issues.

THE SECOND METHOD

Basic Idea

In the second method, we put more emphasis on the reduction of learning time. As a basic idea coping with this problem, we adopt the incremental segmentation of the state space by which the state space is autonomously segmented, and as a results we expect the reduction of the learning time and the capability of coping with dynamic change of the environment.

A key issue is to find the basic policy to segment the state space so as to realize the desirable features described above. The following two policies can be considered.

A: Segment the state if the prediction of sensor outputs is incorrect.

B: Segment the state if the same action causes the desirable or undesirable result (ex., transition to the goal states or non-goal states) even though the prediction itself is correct.

According to the first policy, the robot can discriminate the world situations with as few states as possible based on the experiences until the current time. This contributes to the followings:

1. as long as the prediction of sensor outputs is correct, tedious exploration process can be eliminated, and therefore
2. reinforcement learning converges immediately. Further
3. the robot can cope with dynamic change of the environment due to its incremental processes of the segmentation.

However, the policy **A** does not care where the goal state is. On the other hand, the policy **B** contributes to the emergence of the purposive behavior. Even though the prediction is correct, it would be nonsense if the same action from the same state resulted in different situations. This state should be separated so that the same action can always cause the desirable transition.

From the above arguments, the policy **A** is related to the world model construction by coarse mapping between states and actions far from the good states. While, the policy **B** is related to the the goal oriented segmentation based on the reinforcement signals. As a result fine mapping between states and actions near the goal states is obtained.

Algorithm

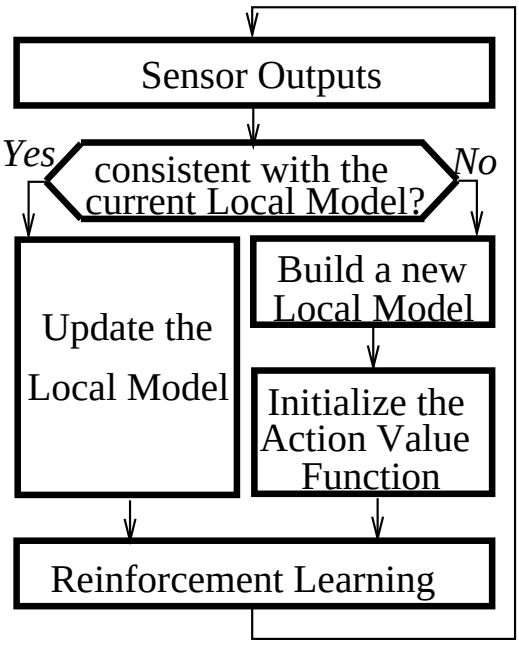


Figure 11: The rough flow of the proposed method

Fig.11 shows the rough flow of the proposed method. First, the robot acquires sensor outputs as data. If the data are consistent with the current local models, the robot updates the local models. Else, the robot builds new local models, and initialize the action value function by reusing the knowledge obtained by the past experiences. Then, it learns the policy using reinforcement learning, and returns the beginning. The robot iterates this cycle forever.

Action Space and Data Structure

In the conventional reinforcement learning methods, an "action" is defined as an execution of motor command per fixed sampling interval. In real situation, this definition often causes "state-action deviation problem" as pointed out by Asada et al.(Asada *et al.* 1995b). They defined such an action as an action primitive, and a action is defined as a sequence of action primitives until the current state changes. Here, we follow their definition.

We define a data set $d_i \in D$, $(i = 1, 2, \cdots)$ as a triplet of action primitive $m_i \in M$, sensor output $\boldsymbol{s}_i \in S$ and its gradient $\dot{\boldsymbol{s}}_i \in \dot{S}$.

If the robot stores the all data of its experiences, the amount of data will exceed the capacity of the robot. Therefore, it is not practical to store the all data. Further, the robot often receives incorrect data because of

sensor noise, change of the environment, and the uncertainty of motor commands. Then, we update the gradient of inputs vector $\dot{\boldsymbol{s}}_i$, when the robot receives a new data set d_j.
if

$$|\boldsymbol{s}_i - \boldsymbol{s}_j| < \epsilon \ \ and \ \ m_i = m_j$$

then

$$\dot{\boldsymbol{s}}_i = (1-\beta)\dot{\boldsymbol{s}}_j + \beta\dot{\boldsymbol{s}}_i$$

else

register $\boldsymbol{s}_j$ as a new datum.

Here, $0 < \beta < 1$ and ϵ stands for a similarity threshold. $|\cdot|$ means weighted Euclidean norm.

Local Model Construction

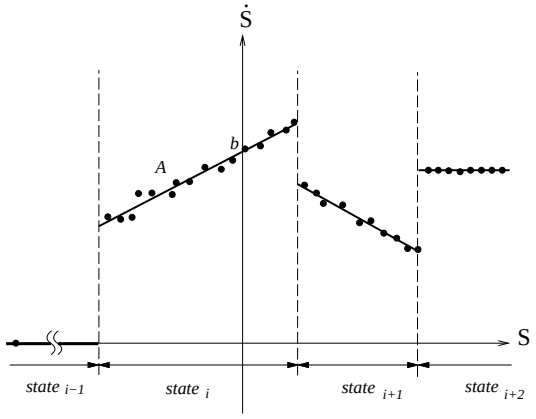


Figure 12: The construction of local model and the segmentation of sensor space

We first explain the method of local model construction by using a linear model of the the gradient of sensor outputs, that is,

$$\dot{\boldsymbol{s}} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{b}.$$

The algorithm for local model construction and segmentation is as follows:

1. Gather data sets which have the same action primitive.
2. Apply the weighted linear regression method to fit a linear model to the data sets.
3. Divide the data into two with a method of cluster analysis using weighted Euclidean norm as similarity and return 2 if the unbiased variance of the residual exceeds a certain threshold, else stop.

Fig.12 shows an example of the construction of local model and the segmentation of sensor space in case of one dimension of the sensor output.

Near the goal state, the segmented region obtained by the above process is not always appropriate because multiple transitions (success in the reaching the goal state or failure), from a same pair of the sensor outputs in the same region and the action primitive can be often observed. Then, we use the reinforcement signals to divide the segmented region so that the same action primitive from the divided region can reach the unique state (the goal state or others).

The segmented regions obtained by the above process are regarded as "states" for the reinforcement learning method. Each segmented region has several data sets d_i, and let the $\boldsymbol{s}_i$ $(i = 1, 2, \cdots)$ be the representatives of the region. A new sensor outputs $\boldsymbol{s}_q$ is classified into one of the states by finding a representative in the corresponding state based on NN(nearest neighbor) methods.

Action Generation

As we stated in the section , we define "action" as "a sequence of several action primitives until the current state changes". The sequence of action primitives with the local model is generated as follows.

One can calculate the desired gradient of sensor outputs $\dot{\boldsymbol{s}}_d$ from the current sensor outputs $\boldsymbol{s}_j$ and desired sensor outputs $\boldsymbol{s}_d$, that is,.

$$\dot{\boldsymbol{s}}_d = \boldsymbol{s}_d - \boldsymbol{s}_j.$$

Since the linear model parameters have been obtained in each local model, we can predict a desirable action to satisfy the above equation. The robot carries out the action primitive m_d which is closest to the desired gradient of sensor outputs.

$$m_d = arg\min_{m_i}(\dot{\boldsymbol{s}}_d - \dot{\boldsymbol{s}}_{m_i})^2 \qquad (3)$$

We assumed the continuity of sensor space. However, if the robot cannot observe the objects in the environment, the robot cannot obtain the information about the objects from sensors. Therefore, there is a case that equation (3) cannot be applied. In such cases, however, an action for a state transition is needed, then we adopt a sequence of the same action primitive as one action until the current state changes.

Reuse of the Knowledge Obtained by Experiences

Theoretically, the action value function should be reset every time the new state space is constructed by the incremental segmentation of the state space. This prevents the knowledge obtained by the past experiences from being used efficiently in the learning process. Then, we consider to reuse the knowledge by calculating the new action value function for the new segmented state space from the old state space and its action-value function.

Basic idea is to adopt a new action value function calculated by weighted sum of the old action value function as the initial knowledge for reinforcement learning. The weights are calculated based on the

numbers of the sensor output representatives in both the new and old states. Concrete procedure is given as following.

S^{old} and S^{new} denote the old and new state spaces, respectively. $\boldsymbol{s}_k(k = 1, 2, \cdots, n)$, ${state_j}^{old}(j = 1, 2, \cdots, n^{old})$ and ${state_i}^{new}(i = 1, 2, \cdots, n^{new})$ denote the sensor output of stored data d_i, a state of the old state space and a state of the new state space. We prepare a $n^{old} \times n^{new}$ matrix $T(state^{old}, state^{new})$ of which component $t(state_i^{old}, state_j^{new})$ represents the number of sensor output representatives $\boldsymbol{s}_k$ that are classified into $state_j^{new}$ from $state_i^{old}$. Then, we can calculate the action-value function of the new state space $Q(state_i^{new}, a)$ as follows.

$$Q(state_i^{new}, a) = \sum_{j=1}^{n^{old}} \omega_{ji} Q(state_j^{old}, a), \qquad (4)$$

where

$$\omega_{ji} = \frac{t(state_j^{old}, state_i^{new})}{\sum_{l=0}^{n^{old}} t(state_l^{old}, state_i^{new})}. \qquad (5)$$

Task and Assumptions

Only one assumption we need is continuity of the sensor space. This makes local model construction efficient, and therefore contributes to eliminate unnecessary exploration.

We apply the method to shooting behavior acquisition by a soccer robot as an example of robot tasks which is the same one as the first method dealt.

The robot often loses the ball and/or goal because of its narrow angle of view (65°). In such a case, there are no feature values of ball and/or goal. However since the robot knows into which direction it lost the ball and/or the goal by memorizing the previous state, large absolute constants (opposite signs) are assigned to these lost states. As the result, the local models for these states are obtained with their gradients equal zeros (The left side of Figure 12 indicates such a case).

Simulation

We assign a reward value 1 when the ball was kicked into the goal or -0.1 otherwise. 90% of the time the robot selects the action specified by its optimal policy, the remaining 10% of the time it takes a random action.

Fig.13 shows the success rate and the number of states during the incremental state space segmentation and the processes of shooting behavior acquisition. Here, the success rate indicates the number of successes in the last twenty trials.

Fig.14 shows a projection of the state space after 1,110 trials, where the state space in term of ball size and goal size is indicated when the position of the ball and the goal are center of the screen and the orientation of the goal is frontal.

Fig.15 shows the success rate and the number of states in the case that the ball diameter suddenly became twice at the 500th trial. It suggests the proposed method can deal with dynamic change of the environment.

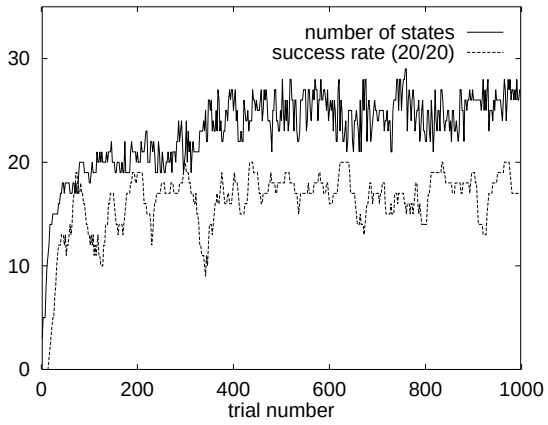


Figure 13: The success rate and the number of states

Experiment on the Real Robot

Fig.16 shows a configuration of the real mobile robot system. Fig.17(a) and (b) show the images taken by a TV camera mounted on the robot and processed by Datacube MaxVideo 200, a real-time pipeline video image processor. The image processing and the vehicle control system are operated by VxWorks OS on MC68040 CPU which are connected with host Sun workstations via Ether net. The result of image processing are sent to the host CPU to decide an optimal action against the current state. The sampling time is about 30ms.

Fig.18 shows the state space after 72 trials. The state space in term of ball size and goal size is indicated when the position of the ball and the goal are center of the screen and the orientation of the goal is frontal. The numbers of acquired states and data are 18 and 151, respectively.

Fig.19 shows how the robot tries to shoot a ball into the goal. Because of the sensor noise and the uncertainty of the motor commands, the robot often misunderstands the states, and takes wrong actions, therefore it fails to do the task. ① indicates that the robot is going to shoot a ball into the goal and move forward. But it fails to kick the ball at ② because the speed is too hight to turn. The ball is occluded by the robot in ②. Then, it goes left back so that it can shoot a ball at ③. But it fails again at ④. Then it goes left back again at ⑤. After all, the robot does the shooting task successfully at ⑥.

CONCLUDING REMARKS

We have proposed a method for constructing the state and action spaces based on experiences, and shown the validity of the method with computer simulations and real robot experiments.

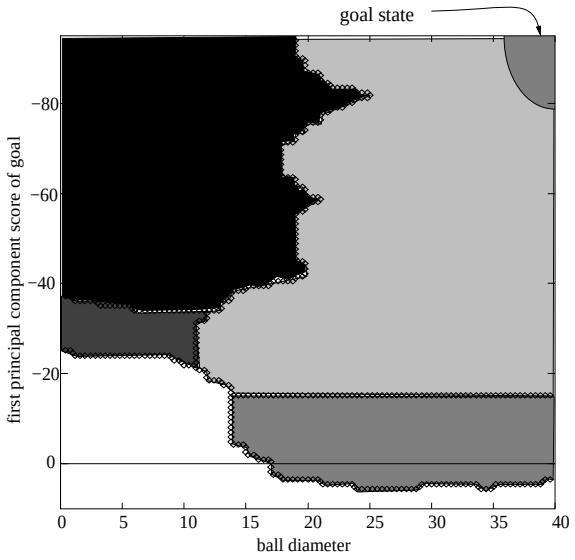


Figure 14: Result of state space construction

- The sensory information in our task is an image of a red ball and a blue goal filtered by color image segmentation, and their image features such as positions, areas, and orientation are used as axes of the state space. Generally, selection of features from a ordinary images is considerably a hard problem. A problem which feature is necessary to accomplish a give task might be much harder when such a feature changes depending on situations. Since use of all possible sensory information seems impossible, selection of features obtained by the given capability for feature detection is more important. For example, behavior acquisition based on the visual motion cues (Nakamura & Asada 1995) and based on stereo disparity and motion cues (Nakamura & Asada 1996) have been proposed. A learning mechanism for selecting features from the sensory data processing available should be developed.
- Coping with "hidden" states is another essential problem although we have not dealt with it here. This corresponds to coping with the temporal complexity of the state space structure while the above with the spatial complexity of it. How many differential operations of feature vectors are necessary and sufficient for the given task? An essential problem is selection of input vectors including the temporal axis.
- We can regard that as a result of state space construction, the action space also is temporally abstracted by defining an action as a sequence of action primitives and parameterizing its length. Since our robot has only two DOFs (degrees of freedom) s-

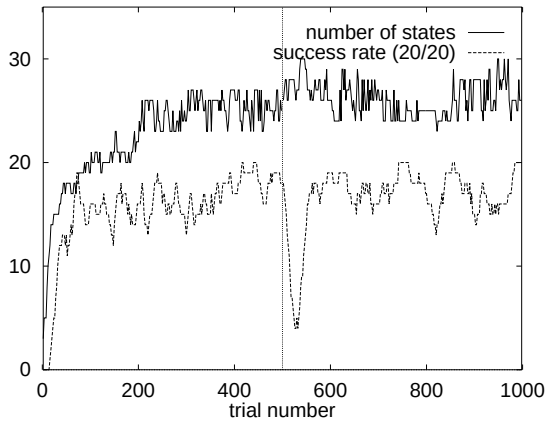


Figure 15: The success rate and the number of states in the case that environment change one the way

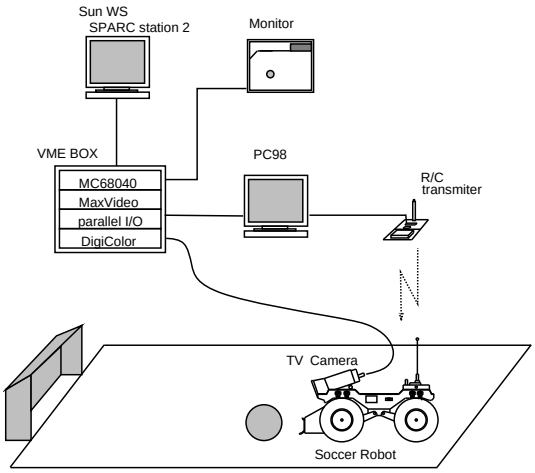


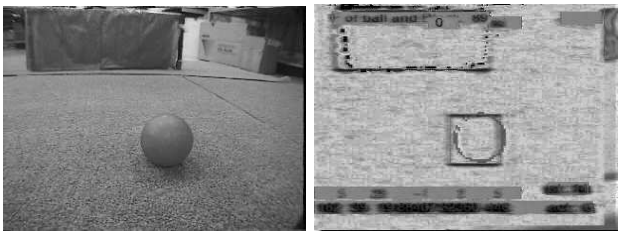
Figure 16: A configuration of the real robot

patial abstraction of action space is not necessary. However, spatial abstraction of action space is generally needed if the robot has many DOFs. For example, we human beings easily grasp something by controlling a very simple parameter (close or open your hand) although it has many DOFs physically. Both spatial and temporal abstraction of action space is necessary with state space construction.

References

Asada, M.; Noda, S.; Tawaratsumida, S.; and Hosoda, K. 1995a. Vision-based reinforcement learning for purposive behavior acquisition. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, 146–153.

Asada, M.; Noda, S.; Tawaratsumida, S.; and Hosoda, K. 1995b. Vision-based reinforcement learning for purposive behavior acquisition. In *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, volume 1, 146–153.



(a) input image (b) detected image

Figure 17: Detection of the ball and the goal

Chapman, D., and Kaelbling, L. P. 1991. "Input generalization in delayed reinforcement learning: An alogorithm and performance comparisons". In *Proc. of IJCAI-91*, 726–731.

C.J.C.H.Watkins. 1989. *Learning from delayed rewards*. Ph.D. Dissertation, King's College, University of Cambridge.

Connel, J. H., and Mahadevan, S., eds. 1993. *Robot Learning*. Kluwer Academic Publishers.

Cramër, H. 1951. *Mathematical Methods of Statistics*. Princeton, NJ: Princeton University Press.

Dubrawski, A., and Reingnier, P. 1994. Learning to categorize perceptual space of a mobile robot using fuzzy-art neural network. In *Proc. of IEEE/RSJ/GI International Conference on Intelligent Robots and Systems 1994 (IROS '94)*, 1272–1277.

Ishiguro, H.; Sato, R.; and Ishida96, T. 1996. Robot oriented state space construction. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS96)*, ???–???

Kröse, B., and Dam, J. 1992. Adaptive state space quantisation for reinforcement learning of collision-free navigation. In *Proc. of IEEE/RSJInternational Conference on Intelligent Robots and Systems 1992 (IROS '92)*, 1327–1332.

Mataric, M. 1994. "Reward functions for accelerated learning". In *Proc. of Conf. on Machine Learning-1994*, 181–189.

Nakamura, T., and Asada, M. 1995. Motion sketch: Acquisition of visual motion guided behaviors. In *Proc. of IJCAI-95*, 126–132.

Nakamura, T., and Asada, M. 1996. Stereo sketch: Stereo vision-based target reaching behavior acquisition with occlusion detection and avoidance. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1314–1319.

Sutton, R. S. 1992. "Special issue on reinforcement learning". In Sutton(Guest), R. S., ed., *Machine Learning*, volume 8. Kluwer Academic Publishers. –.

Watkins, C. J. C. H., and Dayan, P. 1992. "Technical note: Q-learning". *Machine Learning* 8:279–292.

Whitehead, S. D. 1991. "A complexity analysis of cooperative mechanisms in reinforcement learning". In *Proc. AAAI-91*, 607–613.

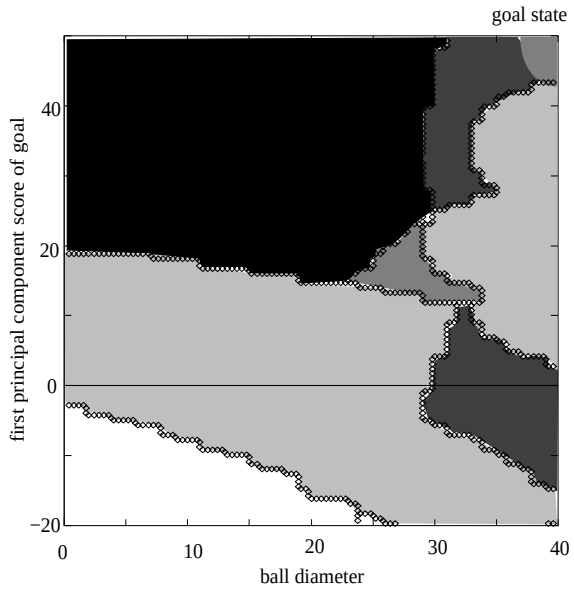


Figure 18: state space construction of real robot experiment

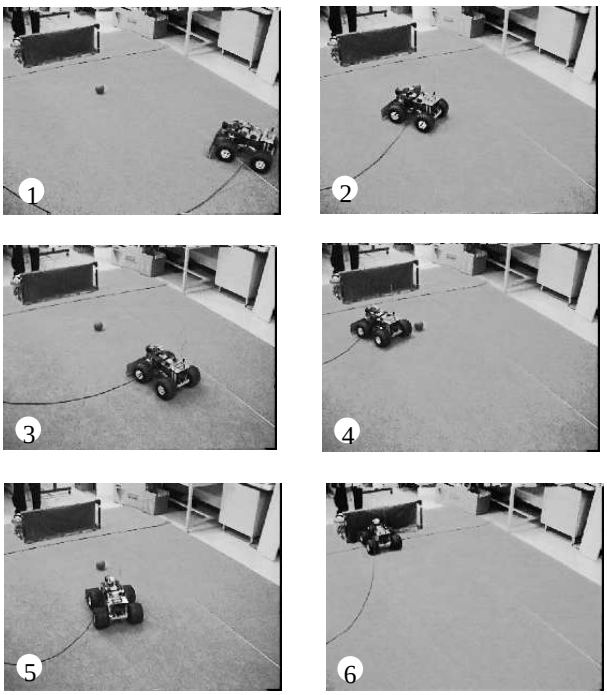


Figure 19: The robot succeeded in shooting a ball into the goal