Real-Time Decision Making under Uncertainty of Self-localization Results

Takeshi Fukase, Yuichi Kobayashi, Ryuichi Ueda,
Takanobu Kawabe, and Tamio Arai

The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
{fukase,kobayasi,ueda,kawabe,arai}@prince.pe.u-tokyo.ac.jp
http://www.arai.pe.u-tokyo.ac.jp/

Abstract. In this paper, we present a real-time decision making method for a quadruped robot whose sensor and locomotion have large errors. We make a State-Action Map by off-line planning considering the uncertainty of the robot's location with Dynamic Programming (DP). Using this map, the robot can immediately decide optimal action that minimizes the time to reach a target state at any state. The number of observation is also minimized. We compress this map for implementation with Vector Quantization (VQ). Using the differences of the values between the optimal action and others as distortion measure of VQ minimizes the total loss of optimality.

Keywords: Dynamic Programming, Vector Quantization, Planning under Uncertainty, Real-time Decision Making

1 Introduction

In Sony Four-Legged Robot League, self-localization with insufficient sensor information and unreliable locomotion is an big problem. Moreover, to localize itself, the robot must swing its head to look for landmarks because the robot's camera has a narrow visual field. It is required to keep the frequency of this "off-ball" observation behavior as small as possible. As a result, the robot is required to judge whether it should execute landmarks observation action or walking action. The simplest criterion for the judgment is to adapt a fixed threshold of the location's uncertainty [2], however, there are many situations in which the robot can decide its action without precise self-localization results.

Mitsunaga *et al.* proposed a decision making tree that gives consideration to the observational strategy based on information criterion [3]. The tree is made from the large experimental teaching data, which contains the information of the motion planning and the probability distribution models of sensing and locomotion. In order to apply larger problems, however, the decision making architecture should once analyze these two kinds of information separately.

Our approach to the real-time decision making method deals with:

- modeling uncertainty in the robot's locomotion and observations,
- adopting Dynamic Programming (DP) [4] to motion planning,

G.A. Kaminka, P.U. Lima, and R. Rojas (Eds.): RoboCup 2002, LNAI 2752, pp. 375–383, 2003.

– enlarging the state space of planning (configuration space) so as to include the uncertainty parameters,
– compressing the off-line calculated information using Vector Quantization.

The robot's locomotion models and observation models are taken into consideration respectively in the process of DP, which guarantees the optimality. By the expansion of the state space to include uncertainty parameters, the observational cost can be computed in the framework of DP.

From another point of view, an effective design of reflective behavior has been proposed by Hugel *et al.*[5]. But it needs highly sophisticated designer's empirical intuition. Our framework realizes the similar behavior as [2,3,5], but is based on the automatic design. So the idea can be applied to the larger problems more easily.

In section 2, the task in the Legged Robot League is specified. Section 3 outlines the proposed real-time motion decision method. In Section 4,5, and 6, the implementation of the method to the task is described. In Section 7, the proposed method is evaluated in the simulation and the experiment.

2 Task and Assumption

The robot's task is to approach the ball from the proper direction so as not to attack the own goal. The followings are the assumptions for the later discussion.

– there are eight discrete walking actions and one observation action,
– the walking actions yield large odometry errors,
– the six unique landmarks are placed around the field,
– the measurement of distance to the landmark contains large errors,
– the robot does not look away from the ball while walking towards it,
– the robot swings its head horizontally for self-localization at the observation action.

The state of the robot and the ball are represented by the next five variables (x, y, θ, r, ϕ), which are shown in Fig.5. (x, y, θ) is the robot's pose on the field. r and ϕ are the distance and orientation of the ball from the robot.

3 Real-Time Decision Making

From the above-mentioned discussion, real-time decision making methods are required to meet following properties: 1) automatic design which can discuss optimality, 2) low computational cost, and 3) ability to express the observational cost and the uncertainty of localization.

To meet the first characteristic, we adopt DP, which is widely used to solve the optimal control problems. The low computational property means that the robot ERS-2100 has 32MB RAM and its calculation speed is equivalent to the 200MHz PC. The volume of DP result is too large to implement on the robot. In order to compress it, we apply Vector Quantization (VQ) [6].

The aspects of uncertainty and observational costs are important in this paper. The uncertainty can be considered as variables of the state space [7].

3.1 Motion Planning with Dynamic Programming

Let $\boldsymbol{x} \in \mathcal{X} \subset R^n$ denote the state vector and $\boldsymbol{u} \in \mathcal{U} \subset R^m$ denote the control input vector. The system dynamics in discrete time is expressed as:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}[\boldsymbol{x}_k, \boldsymbol{u}_k]. \tag{1}$$

The deterministic control policy is given by $\boldsymbol{u}_k = \pi(\boldsymbol{x}_k)$. The purpose of the optimal control problem is to find the optimal policy $\pi^*(\boldsymbol{x})$ that maximizes

$$S = \sum_{k=1}^{T} R[\boldsymbol{x}_k, \boldsymbol{u}_k], \tag{2}$$

where $R[\boldsymbol{x}_k, \boldsymbol{u}_k]$ is the immediate evaluation function of each state and control input pair and T is the time step until the task ends.

We substitute discrete s and a for $\boldsymbol{x}$ and $\boldsymbol{u}$, respectively. Here, $\mathcal{S}$ and $\mathcal{A}$ are the set of discrete states and actions. Bellman equation in discrete time and space (without the discount factor) can be formulated as follows:

$$V^*(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + V^*(s')], \tag{3}$$

$$Q^*(s,a) = \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \max_{a'} Q^*(s',a')], \tag{4}$$

where $\mathcal{P}^a_{ss'}$ denotes the transition probability from state s to s' by taking action a, and $\mathcal{R}^a_{ss'}$ denotes the immediate evaluation given to the state transition from s to s' by taking action a. The optimal state-value function $V^*(s)$ denotes the expected evaluation which is given by taking actions at state s under the optimal policy π^*. The optimal action-value function $Q^*(s,a)$ denotes the expected evaluation after taking action a at state s, in the same way. We call π^* State-Action Map in this paper.

3.2 Planning Optimal Behavior under Uncertainty

When the motions are planned, the variance of pose estimation and the observational cost should be taken into consideration. Fig.1 shows an example where the variance of the posture estimation enlarges when the robot executes a walking action. Fig.2 shows an example where the variance decreases when the robot takes observation. These factors can be formulated as:

$$\left(\boldsymbol{x}_{k+1}, \boldsymbol{\psi}_{k+1}\right) = \boldsymbol{f}'[\left(\boldsymbol{x}_k, \boldsymbol{\psi}_k\right), \left(\boldsymbol{u}_k, \boldsymbol{\omega}_k\right)], \tag{5}$$

where $\boldsymbol{\psi}$ denotes the state variance vector and $\boldsymbol{\omega}$ denotes the observational control vector. Thus, the optimal control problem can be solved in the expanded state space $\{\boldsymbol{x}, \boldsymbol{\psi} | \boldsymbol{x} \in \mathcal{X}, \boldsymbol{\psi} \in \ominus\}$. Fig.3 shows the abstraction of the state transition in the expanded state space. The increase of the variance in the original state space can be expressed as the transition along the $\boldsymbol{\psi}$ axis.

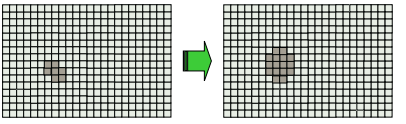

Fig. 1. A state's transition on the occasion of the robot's movement.

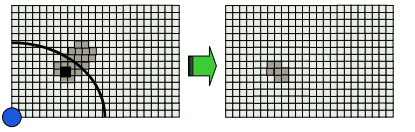

Fig. 2. A state's transition on the occasion of a landmark observation.

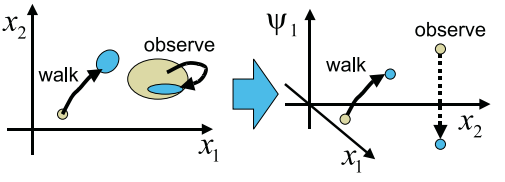

Fig. 3. The state transition in the expanded state space.

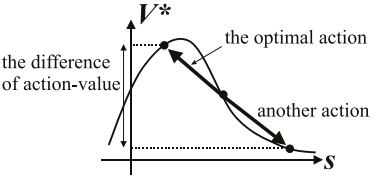

Fig. 4. The differences between the optimal action and another.

3.3 Compression of State-Action Map with Vector Quantization

The map should be compressed in order to implement on the limited amount of robot's memory. We apply Vector Quantization as a data compression method. The map is distorted through compression, and the optimality of action data is lost. We should pay attention not to maximize the decode rate but to minimize the increase of the time to reach the target. Hence, we calculate the differences between the optimal action and others based on the value function as Fig.4, and utilize it as a distortion measure.

4 Implementation 1: Dynamic Programming

4.1 Symbols Definition

Firstly, we quantize (x, y, θ, ϕ) as 100[mm]×100[mm]×15[deg]×100[mm] (Fig.5). We divide r into 12 intervals. Each have different width since the qantized interval does not need to be shorted when the ball is far from the robot. The shortest interval width is 100[mm] and the widest interval width is infinite. Moreover, we add one more parameter ψ which denotes the shape of region in which the robot exists with high probability. ψ is represented to a combination of some $s_{\mathrm{p}i_x i_y i_\theta}$s, which are cuboids in the $xy\theta$-space (Fig.6). The area which the robot exists with high probability is represented as $s_{\mathrm{r}i_x i_y i_\theta i_\psi}$. We restrict the number of ψs to 811 in a lot of combinations of cuboids so as to save the amount of calculation. We define i_ψ as the larger i_ψ becomes, the more the number of ψ's cuboids increases. Eventually, we let a state $\forall s \in \mathcal{S}$ have six indexes as $s_{i_x i_y i_\theta i_r i_\phi i_\psi}$. Hereafter, we often describe $s_{i_x i_y i_\theta i_r i_\phi i_\psi}$, $s_{\mathrm{r}i_x i_y i_\theta i_\psi}$ and $s_{\mathrm{b}i_r i_\phi}$ as s_i, $s_{\mathrm{r}i}$ and $s_{\mathrm{b}i}$, respectively.

Secondly, we define some symbols on actions. The robot has nine fixed actions as Table 1. Each action has the following attributes:

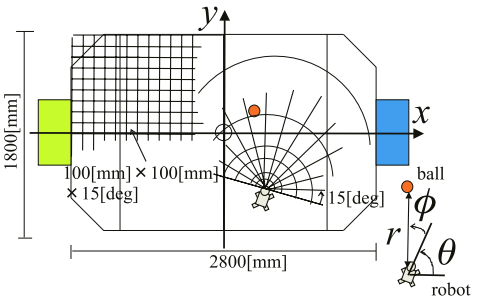

Fig. 5. The robot's position is divided into three-dimensional grids, and the ball's position is divided into two-dimensional grids.

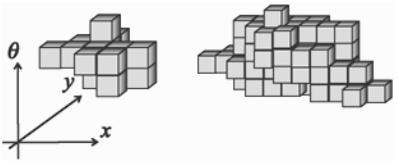

Fig. 6. A quantized ψ consist of three-dimensional cuboids in (x, y, θ) space. When the robot's position is estimated precisely, the number of cuboids in ψ is small.

Table 1. Actions and Parameters of them.

Action a_i	$M_p^{a_i}$[mm],[deg] $(p = (0, 0, 0))$	$\mathcal{R}^{a_i}$ [msec]
1:forward	$(70 \pm 30, 0 \pm 15, 0 \pm 6)$	-768
2:backword	$(-40 \pm 40, 0 \pm 15, 0 \pm 6)$	-768
3:rightside	$(0 \pm 20, -60 \pm 30, 4 \pm 4)$	-896
4:leftside	$(0 \pm 20, 60 \pm 30, -4 \pm 4)$	-896
5:rightforward	$(10 \pm 10, 37.5 \pm 17.5, -14.5 \pm 6.5)$	-832
6:leftforward	$(10 \pm 10, -37.5 \pm 17.5, 14.5 \pm 6.5)$	-832
7:rollright	$(35 \pm 15, -35 \pm 15, 11.5 \pm 6.5)$	-832
8:rollleft	$(35 \pm 15, 35 \pm 15, -11.5 \pm 6.5)$	-832
9:observation	$(0 \pm 0, 0 \pm 0, 0 \pm 0)$	-2800

- Time consumption : $\mathcal{R}^{a_i}(< 0)$. We regard a action's time consumption as negative reward. We assume that $\forall \mathcal{R}^{a_i}$ is independent of $\forall s \in \mathcal{S}$.
- Capable region of the robot's pose after an action a_i which is caused at a pose p : $M_p^{a_i}$. For brief calculation, we assume that the robot moves to a random pose in $M_p^{a_i}$ with the uniform probability, and that the shape of $M_p^{a_i}$ is a cuboid.

Finally we assume that the optimal policy $\pi^*(s)$ is deterministic, i.e. $\pi^*(s)$ chooses one action when a state s is designated.

4.2 Calculation of Probability and Execution of DP Algorithm

We use the value iteration algorithm [4] to obtain the optimal policy π^* with the equation (3). We should refer to the calculation algorithm of $\mathcal{P}_{s_i s_j}^{a_k}$ to execute the value iteration algorithm. We calculate $\mathcal{P}_{s_i s_j}^{a_k}$ with the next two algorithms.

Calculation of Pose's Transition. We do not treat stochastically the renewal of $s_{\mathrm{r}i}$ after an action a_k (we represent it as $s_{\mathrm{r}i}^{a_k}$) since $s_{\mathrm{r}i}$ and $s_{\mathrm{r}i}^{a_k}$ are already stochastic in themselves. We choose the most proper $s_{\mathrm{r}i}^{a_k}$ with the following way;

1. Choose a pose p randomly from the region $s_{\mathrm{r}i}$.
2. Choose a pose q randomly from the region $M_p^{a_k}$.
3. Record the region $s_{j_x j_y j_\theta}$ which contains the pose q.
4. Iterate 1-3 sufficiently.
5. Bind all recorded regions and make a region $\hat{s}_{\mathrm{r}i}^{a_k}$.
6. Choose the most proper $s_{\mathrm{r}i}^{a_k}$ to approximate $\hat{s}_{\mathrm{r}i}^{a_k}$.

Calculation of Ball Position's Transition. We define $\mathcal{P}_{s_{\mathrm{b}i}s_{\mathrm{b}j}}^{a_k}$ as the probability which the ball's position becomes $s_{\mathrm{b}j}$ after an action a_k from $s_{\mathrm{b}i}$. $\mathcal{P}_{s_{\mathrm{b}i}s_{\mathrm{b}j}}^{a_k}$ is calculated with the following way;

1. Assume that the robot's pose is p.
2. Choose a pose q randomly from the region $M_p^{a_k}$.
3. Choose a ball posistion b randomly from the region $s_{\mathrm{b}i}$.
4. Calculate the new ball position b' from p, q and b.
5. Record the region $s_{\mathrm{b}j}$ which includes the position b'.
6. Iterate 1-5 sufficiently.
7. Calculate $\mathcal{P}_{s_{\mathrm{b}i}s_{\mathrm{b}j}}^{a_k}$ from the frequency that $s_{\mathrm{b}j}$ is recorded.

If $s_{\mathrm{r}i}^{a_k} = s_{\mathrm{r}j}$, the value of $\mathcal{P}_{s_i s_j}^{a_k}$ is the same as that of $\mathcal{P}_{s_{\mathrm{b}i}s_{\mathrm{b}j}}^{a_k}$, if not, $\mathcal{P}_{s_i s_j}^{a_k}$ becomes zero.

5 Implementation 2: Compression of State-Action Map

We use Vector Quantization (VQ) [6] so as to compress the State-Action Map π^*, since the data amount of π^* is 588 MB and the robot does not have such a huge amount of RAM. The maximum volume of data that can be transferred to the robot is 16MB. We use Pairwise Nearest Neighbor (PNN) algorithm to choose initial codebooks, and Generalized Lloyd Algorithm (GLA) to refine them [8].

5.1 Definition of Vector

We explain the way with which representative vectors are made. At first, we divide the map since the State-Action Map is too large to be executed VQ all at once. We divide the map $\mathcal{S}$ into $\mathit{\Gamma}_j$ $(j = 1, 2, \ldots, N_{\mathit{\Gamma}_j})$. $\mathit{\Gamma}_j$ has all states whose indexes of ψ are $2j-1$ or $2j$. VQ is executed in each $\mathit{\Gamma}_j$ independently. Furthermore, we divide each $\mathit{\Gamma}_j$ into six dimensional cuboids $\mathit{\Omega}_{jk}$ $(k = 1, 2, \ldots, N_{\mathit{\Omega}})$. Any two cuboids in the same $\mathit{\Gamma}$ must be congruence. We arrange all states of a $\mathit{\Omega}_{jk}$ in a order, and we define $\boldsymbol{v}_{jk} = (\pi^*(s_1^{(jk)}), \pi^*(s_2^{(jk)}), \ldots, \pi^*(s_{N_{\mathrm{e}}}^{(jk)}))$ as a vector which is used in VQ.

Each edge's width of each $\mathit{\Omega}_{jk}$ should be decided that the same vectors are produced in $\mathit{\Gamma}_j$ as much as possible, since it is favorable for VQ. We decide them to minimize the next entropy function:

$$H = -\sum_{k=1}^{N_{\mathit{\Omega}}} \frac{1}{N_{\mathit{\Omega}}} \log \frac{N_{\mathrm{s}}(\boldsymbol{v}_{jk})}{N_{\mathit{\Omega}}}, \tag{6}$$

where $N_{\mathrm{s}}(\boldsymbol{v}_{jk})$ means the number of the same vectors with $\boldsymbol{v}_{jk}$ (it counts $(\boldsymbol{v}_{jk})$).

5.2 Definition of Distorsion

Next, we must define distorsion of any two vectors for VQ. We define the distorsion between two vectors, $\boldsymbol{v}$ and $\boldsymbol{w}$ as

$$D[\boldsymbol{v}, \boldsymbol{w}] = \sum_{\ell=1}^{N_{\mathrm{e}}} D[v_\ell, w_\ell], \tag{7}$$

where $v_\ell, w_\ell \in \mathcal{A}$ are the ℓth elements of $\boldsymbol{v}$ and $\boldsymbol{w}$ respectively. We must define the distorsion $D[a_m, a_n]$ about $\forall m, n$. $D[a_m, a_n]$ is calculated from the optimal action-value function (4) as

$$D[a_m, a_n] = \frac{\sum_{k=1}^{N_\Omega} \sum_{\ell=1}^{N_{\mathrm{e}}} \delta_{\pi^*(s_\ell^{(jk)}), a_m} \{Q^*(s_\ell^{(jk)}, a_m) - Q^*(s_\ell^{(jk)}, a_n)\}}{\sum_{k=1}^{N_\Omega} \sum_{\ell=1}^{N_{\mathrm{e}}} \delta_{\pi^*(s_\ell^{(jk)}), a_m}}, \tag{8}$$

where, $\delta_{\alpha,\beta}$ is Kronecker delta.

To execute PNN and GLA based on this distortion, we can obtain the compressed State-Action Map. For the calculation of DP and VQ, we spend three days on a Pentium Ⅲ 866 MHz PC.

6 Implementation 3: The On-Line Algorithm

The tasks of the on-line part are to recognize the current state of the robot and to search an optimal action from the compressed map. We use Uniform Monte Carlo Localization (Uniform MCL) [1] for self-localization, and for modeling of state transitions which are caused by the landmark observation action. Simulations of Section 7 use this state transition models. In experiments of Section 7, the robot specifies the current state of the robot with Uniform MCL results.

7 Simulation and Experiment

7.1 Purpose and Conditions of Simulation

We inspect the efficiency of our method by simulation. We compare the results of following two methods in order to verify the effectiveness to consider the self-localization's uncertainty.

1. Referring map method: utilizes the compressed map for all the decision making including the judgment of observation.
2. Threshold method: uses the compressed map without variance for making choice of walking actions. The robot observes landmarks when the width of the probability distribution is over a fixed threshold, which is $(x_{\mathrm{th}}, y_{\mathrm{th}}, \theta_{\mathrm{th}}) = (600[\mathrm{mm}], 500[\mathrm{mm}], 60[\mathrm{deg}])$.

Other conditions are settled as follows.

Table 2. The results of the simulations.

Condition	Time[sec]	# of obs.	Success rate
	Referring map method		
1	31.4	2.6	10/10
2	37.6	3.6	10/10
3	35.9	2.8	10/10
	Threshold method		
1	34.4	4.0	10/10
2	41.5	4.6	8/10
3	38.6	3.6	9/10

Table 3. The results of the experiment.

Condition	Time[sec]	# of obs.	Success rate
1	27.5	1.4	9/10
2	42.0	3.0	6/10
3	41.1	2.3	8/10

Table 4. Initial positions.

Condition	r[mm]	ϕ[deg]	x[mm]	y[mm]	θ[deg]
1	2100	30	-1000	-600	0
2	1800	0	1000	0	180
3	2100	60	1000	-600	90

- Table 4 shows initial conditions. The robot knows the initial positions.
- The robot's real pose is updated with random errors. The robot updates its estimating state according to the transition probability.
- The robot acquires the relative position to landmarks with random errors.
- In referring the map case, the task is terminated when the robot's estimating state belongs to the terminative states. In using threshold case, it is terminated when the robot's estimating pose belongs to the terminative states.
- Success cases are that the robot actually reaches a target position.

7.2 Results and Discussion of Simulation

We simulated 10 times on each case and initial conditions. The results are shown in Table 2. In the referring the map case, the robot succeeded to reach the target position at all trials. This indicates that the calculation of DP converged and the distortion of the compressed map was small. In the using the thresholds case, there were some failures. In the referring the map case, both the average number of observation and the time to reach the target were smaller than using thresholds case. These results indicate that the robot observed more effectively as a result of the off-line planning.

Fig. 7. An example of experiments.

7.3 Experiment

We implemented the described method and evaluated it with experiments. The initial conditions are settled exactly the same as the simulation. We judged a trial to be success if the robot touched the ball at first time from proper direction. The results of the experiments are shown in Table 3. The total success rate was about 75%. The failure cases were that the robot touched the ball unintentionally. These were due to the measurement error of the ball, which was not assumed in our model.

8 Conclusion

We took the uncertainty of the robot's pose into account by expanding the state space and designed a State-Action Map with DP by off-line calculation. The map was compressed with VQ in order to implement on the limited amount of robot's memory. We also defined the distortion between any two actions based on the action-value function. The total distortion of the map through compression was minimized as a result. By the simulations and experiments, it was verified that the robot observes the landmarks more efficiently than the fixed threshold case.

References

1. R. Ueda, T. Fukase, Y. Kobayashi, T. Arai, H. Yuasa and J. Ota: "Uniform Monte Carlo Localization – Fast and Robust Self-localization Method for Mobile Robots," Proc. of ICRA-2002, to appear.
2. E. Winner and M. Veloso: "Multi-Fidelity Robotic Behaviors: Acting With Variable State Information," Proc. of the Seventeenth National Conference on Artificial Intelligence, Austin, August 2000.
3. N. Mitsunaga and M. Asada: "Observation strategy for decision making based on information criterion," Proc. of IROS-2000, pp.1211-1216, 2000.
4. R. S. Sutton and A. G. Barto: "Reinforcement Learning: An Introduction," The MIT Press, 1998.
5. V. Hugel, P. Bonnin and P. Blazevic: "Reactive and Adaptive Control Architecture Designed for the Sony Legged Robots League in RoboCup 1999," Proc. of IROS-2000, pp.1032-1037, 2000.
6. A. Gersho and R. M. Gray: "Vector Quantization and Signal Compression," Kluwer Academic Publishers, 1992.
7. S. M. LaValle: "Roboto Motion Planning: A Game–Theoretic Foundation," Algorithmica, Vol. 26, pp.430-465, 2000.
8. W. H. Equitz: "A new vector quantization picture coding," IEEE Trans. Acoust. Speech Signal Process., pp. 1568-1575, October 1989.