Path Planning of a Mobile Robot as a Discrete Optimization Problem and Adjustment of Weight Parameters in the Objective Function by Reinforcement Learning

Harukazu Igarashi

School of Engineering, Kinki University,
Higashi-Hiroshima, Hiroshima, 739-2116, Japan
igarashi@info.hiro.kindai.ac.jp

Abstract. In a previous paper, we proposed a solution to path planning of a mobile robot. In our approach, we formulated the problem as a discrete optimization problem at each time step. To solve the optimization problem, we used an objective function consisting of a goal term, a smoothness term and a collision term. This paper presents a theoretical method using reinforcement learning for adjusting weight parameters in the objective functions. However, the conventional Q-learning method cannot be applied to a non-Markov decision process. Thus, we applied Williams's learning algorithm, REINFORCE, to derive an updating rule for the weight parameters. This is a stochastic hill-climbing method to maximize a value function. We verified the updating rule by experiment.

1 Introduction

There have been many works on the navigation [1] and path planning [2] of mobile robots. In a previous paper, we proposed a solution to path planning and navigation of a mobile robot[3]. In our approach, we formulated the following two problems at each time step as different discrete optimization problems: 1) estimation of position and direction of a robot, and 2) action decision. For the first problem, we minimized an objective function that includes a data term, a constraint term and a prediction term. This approach was an approximation of Markov localization[4].

For the second problem, we minimized another objective function that includes a goal term, a smoothness term and a collision term. While the results of our simulation showed the effectiveness of our approach, the values of weights in the objective functions were not given theoretically. This paper presents a theoretical method using a reinforcement learning to adjust the weight parameters used in the second problem.

In our reinforcement learning, the value function is defined by the expectation of a reward given to a robot's path. The path is generated stochastically because we used a probabilistic policy with a Boltzmann distribution function for determining the actions of the robot. This optimizes the local objective function stochastically to search for the globally optimal path. However, the stochastic process is not a Markov decision process because the objective function includes an action at the preceding time in the smoothness term. The usual Q-learning method cannot be applied to such a non-Markov decision process. Thus, we applied Williams's learning algorithm, REINFORCE[5], to derive an updating rule for the weight parameters. This is a stochastic hill-climbing method to maximize the value function. The updating rule was verified by our experiments.

2 Objective Function for Path Planning

We assume that a series of actions decided at each time step would derive a desirable path or trajectory of a robot. We define the following objective function E_v of a velocity v_t at time t,

$$E_v\left(v_t; r_t, v_{t-1}, r_{goal}\right) = b_1 E_{goal} + b_2 E_{smth} + b_3 E_{clsn} . \tag{1}$$

The minimal solution of Eq. (1) gives an estimate for the robot's action at time t.

The first term represents an attractive force to the goal r_{goal}. It is defined as

$$E_{goal}(v_t; r_t, r_{goal}) = \mathrm{sgn}\left[G(v_t)\right] \cdot G(v_t)^2 , \tag{2}$$

where sgn(x) denotes the sign of x and $G(v_t)$ is defined by

$$G(v_t) = \left\| r_{goal} - r'_{t+1}(v_t; r_t) \right\| - \left\| r_{goal} - r_t \right\| . \tag{3}$$

In Eq.(3), r'_{t+1} is defined by $r'_{t+1}=r_t+v_t$. The second term in Eq. (1), E_{smth}, is defined by

$$E_{smth}(v_t; v_{t-1}) = \left\| v_t - v_{t-1} \right\|^2 \tag{4}$$

to minimize changes in a robot's velocity vector. The last term in Eq. (1), E_{clsn}, represents a repulsion force for avoiding collisions with obstacles and walls. We define the term as

$$E_{clsn}(v_t; r_t) = \begin{cases} D_{clsn} & \text{if } Dist(r'_{t+1}) \le 0 \\ -Dist(r'_{t+1})^2 & \text{if } 0 < Dist(r'_{t+1}) \le L \\ -L^2 & \text{if } Dist(r'_{t+1}) > L \end{cases} , \tag{5}$$

where Dist(r) means the shortest distance from the robot's position r to obstacles and walls. The constant D_{clsn} represents a degree of penalty given when the robot collides with obstacles or walls. The size L means a range within which the repulsion force starts to work on a robot. In our simulation, we set D_{clsn}=100000 and L=15.

Thrun et al. had proposed another optimization approach for motion planning [6]. In their approach, a non-Markov term, such as E_{smth} was not taken into account. We restricted and discretized a search space when minimizing the function $E_v(v_t)$. We only took into account velocity vectors whose lengths were smaller than a constant[3].

3 Learning Weights of Terms

3.1 Value Function and Probabilistic Policy

Trajectories given by minimizing $E_v(v_t)$ depend on weights of terms, $\{b_j\}$(j=1,2,3)[3]. In order to control the weights properly, we applied a reinforcement learning, REINFORCE[5], which was proposed by Williams, 1992, to our path planning.

We define a value function $V(\theta)$, which is an expectation of a reward $R(l_i)$ given to a trajectory l_i, as

$$V(\theta) = E\left[R(l_i)\right] = \sum_i P(l_i) \cdot R(l_i) , \tag{6}$$

where P(l) is a probability that trajectory l is produced by a policy □. A trajectory l_i is a series of robot's positions at times t(t=0,1,...,N$_i$).

We define the policy □ using a Boltzmann distribution function as

$$\square\left(v_t;r_t,r_{t\square1},\{b_k\}\right)\square\frac{e^{\square E_v(v_t)/T}}{\displaystyle\square_{v_t} e^{\square E_v(v_t)/T}}\ , \tag{7}$$

where E$_v$(v$_t$) is the ojective function shown in Eq. (1) and T is a parameter to control the randomness in choosing an action v$_t$ at each time.

3.2 Steepest Gradient Method

We use a steepest gradient method for maximizing the value funciton in Eq. (6). We have to caluculate the right-hand side of the following equation:

$$\frac{\square}{\square b_k}V(\square)\square\square_i\left\{\frac{\square}{\square b_k}P(l_i)\right\}R(l_i). \tag{8}$$

The probability distribution P(l_i) is expressed by a product of P$^{\square}$(r$_t$,r$_{t+1}$)'s as

$$P(l_i)\square P^{\square}(r_0,r_1)P^{\square}(r_1,r_2)\cdots P^{\square}\left(r_{N_i\square1},r_{N_i}\right), \tag{9}$$

where P$^{\square}$(r$_t$,r$_{t+1}$) is a probability that a robot moves to position r$_{t+1}$ when it stands at r$_t$ and takes a policy □. By differentiating Eq. (9) with b$_k$, we obtained

$$\frac{\square}{\square b_k}P(l_i)\square\square_{t\square0}^{N_i\square1}\left[\square_{n\square0,n\square t}^{N_i\square1}P^{\square}(r_n,r_{n\square1})\square\frac{\square}{\square b_k}P^{\square}(r_t,r_{t\square1})\right]. \tag{10}$$

By the following equation,

$$P^{\square}(r_t,r_{t\square1})\square\square_{v_t}P^{v_t}(r_t,r_{t\square1})\square\square(v_t), \tag{11}$$

we obtained that

$$\begin{aligned}\frac{\square}{\square b_k}P^{\square}(r_t,r_{t\square1})&\square\square_{v_{tt}}P^{v_t}(r_t,r_{t\square1})\left\{\frac{\square}{\square b_k}\square\left(v_t;r_t,r_{t\square1},\{b_k\}\right)\right\}\\&\square\square_{v_t}P^{v_t}(r_t,r_{t\square1})\square(v_t)\square\frac{\square}{\square b_k}\left[\ln\square\left(v_t;r_t,r_{t\square1},\{b_k\}\right)\right]\\&\square\square_{v_t}P^{v_t}(r_t,r_{t\square1})\square(v_t)e_k(t)\ ,\end{aligned} \tag{12}$$

where $e_k(t)$ is called the *characteristic eligibility* of b$_k$[5]. Substituting Eq. (12) into the right-hand side of Eq. (10), we obtained from Eq. (8) that

$$\frac{\square}{\square b_k}V(\square)\square E\left[R(l_i)\square\square_{t\square0}^{N_i\square1}e_k(t)\right] \tag{13}$$

This is the same result that Williams derived as his episodic REINFORCE algorithm [5]. He used a neural network model for a probabilistic policy □.

However, we obtained a different updating rule from the REINFORCE algorithm because we used Eq. (7) instead of a neural network model for action decision. Using Eqs. (1) and (7), the characteristic eligibility of b$_k$ is expressed as

$$e_k(t) \square \frac{\square}{\square b_k}\left[\ln \square\left(v_t; r_t, r_{t\square 1}, \{b_k\}\right)\right] \square \square \frac{1}{T}\left[\frac{\square E_v}{\square b_k} \square \left\langle \frac{\square E_v}{\square b_k}\right\rangle_{T,\{b_k\}}\right], \qquad (14)$$

where operation <...> refers to the expectation weighted with a Boltzmann factor, i.e.,

$$\langle X\rangle_{T,\{b_k\}} \square \frac{\underset{v_t}{\square} X \square e^{\square E_v\left(v_t;r_t,r_{t\square 1},\{b_k\}\right)/T}}{\underset{v_t}{\square} e^{\square E_v\left(v_t;r_t,r_{t\square 1},\{b_k\}\right)/T}} . \qquad (15)$$

By substituting Eq. (14) into $e_k(t)$ in Eq. (13) and using the steepest gradient method, we can derive the following learning rule of weights $\{b_k\}(k=1,2,3)$:

$$\square b_k \square \square\square\frac{\square V(\square)}{\square b_k} \square \square \frac{\square}{T} E\left[R(l_i)\square \overset{N_l\square 1}{\underset{t\square 0}{\square}}\left\{\frac{\square E_v}{\square b_k} \square \left\langle \frac{\square E_v}{\square b_k}\right\rangle_{T,\{b_k\}}\right\}\right] . \qquad (16)$$

The constant □is a learning rate factor to be set at a positive small number.

Moreover, we can prove that b_k's converge without the averaging operator E[...] in Eq. (16) by analogy with a back propagation rule used in multi-layer perceptrons. We used the learning rule without operator E[...] in the experiment in the next section.

4 Simulation

We consider a sample problem where a single robot moves to a goal from a starting position while avoiding static obstacles. Figure 1 shows the locations of obstacles, walls, the start point, and the goal point.

4.1 Reward Function

The objective of this sample problem is to find the path that minimizes the robot's moving time from the start point to the goal point and keeps a safe distance from obstacles and walls. We consider the following reward function R(*l*) reflecting these two requirements : $R(l)=c_1R_{time}(l)+c_2R_{dist}(l)$, where $R_{time}(l)$ represents the degree of a user's satisfaction to the moving time from the start to the goal in a trajectory *l*. The function $R_{dist}(l)$ represents the degree of a user's satisfaction to the shortest distance from the robot to obstacles and walls if a robot moves along a path *l*. We call these functions *achievement functions*. We used the trapezoid-like functions shown in Fig. 2 as achievement functions. They take a value between 0 and 10 and are characterized by the parameters, □, □, □and □.

In our experiment, we set □=□=0,□□=20,□=50 for $R_{time}(l)$ and □=0, □=15, □=□=2000 for $R_{dist}(l)$. This means that trajectories whose moving times are shorter than 20 and trajectories that force a robot to keep a distance longer than 15 are most desirable. The weights c_1 and c_2 are set at 0.5 to find a path that is balanced between moving time and safety.

4.2 Experimental Conditions

We set initial values of b_k's as $b_1=b_2=b_3=1.0$ and set □ in Eq. (16) at 0.00001. Parameter T is fixed to 5.0 and we did not carry out any annealing procedure. Under these experimental conditions, we repeated the learning cycle one million times. It

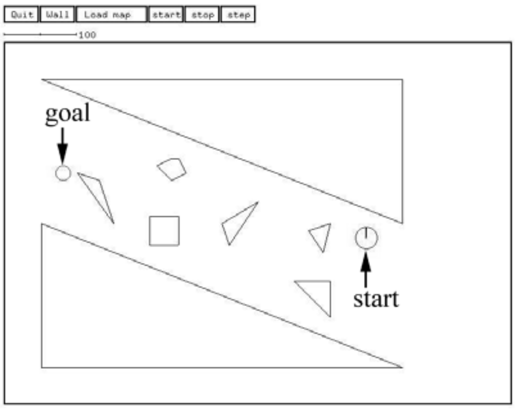

Fig. 1 Robot's world in our simulation

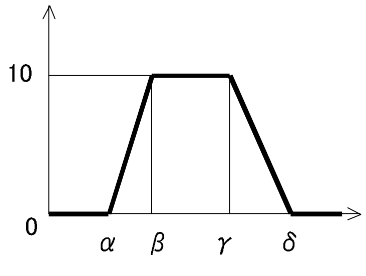

Fig. 2 General shape of achievement function

took about 17 hours to complete one experiment using a work station, SUN Ultra Sparc 30 (CPU: Ultra Sparc-II, 248 MHz).

4.3 Experimental Results

Figure 3 shows changes in R, R_{time} and R_{dist}. The values are averaged over each period of 10000 updating steps. Changes in parameters $\{b_k\}(k=1,2,3)$ are shown in Fig. 4.

In Fig. 3, the expectation of the reward function R increases as the learning proceeds. It increased from 4.35 to 8.17 at the end of learning. This value is about two times larger than that obtained before the learning. This increase comes from the improvement in the second term R_{dist}. The function R_{dist} forces a robot to keep a certain distance from obstacles and walls for safety. Figure 4 shows that the weight b_3 of the collision term E_{clsn} increased gradually. This improved the safety of the path.

Figures 5 and 6 show the robot's trajectories obtained using unlearned values and learned values of $\{b_k\}(k=1,2,3)$, respectively. The parameter T was set to a very small positive value when we obtained these two trajectories. Because we considered that the trajectory obtained at T=0 is a center point in the solution space for searching an optimal path and can be used to check whether values of weights $\{b_k\}(k=1,2,3)$ in $E_v(v_t)$ are optimized in producing desirable trajectories.

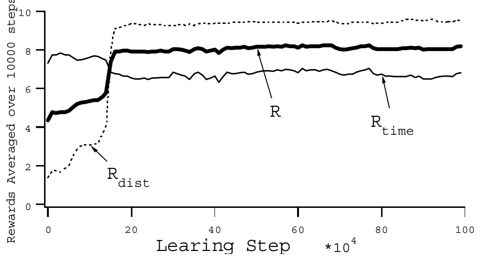

Fig.3 R, R_{time} and R_{dist} averaged over 10000 learning steps

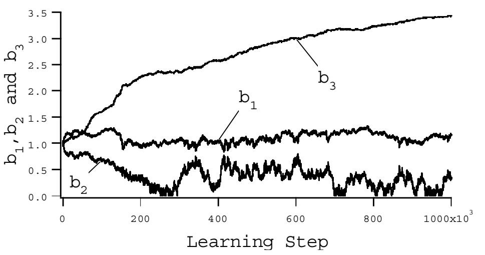

Fig.4 Learning of weights b_1, b_2 and b_3

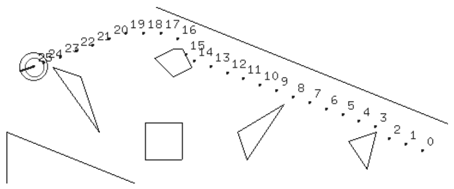

Fig. 5 A path planned deterministically with weights that had not been learned

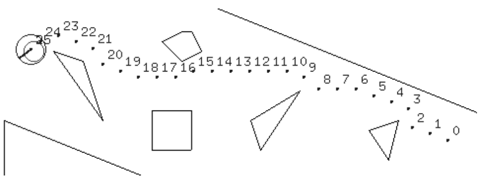

Fig. 6 A path planned deterministically with weights that had been learned

In Fig. 5, one can see that the value of the unlearned weight b_3 is so small that the robot ran into an obstacle at time 15. Figure 6 shows that the problem in safety at time 15 was solved by learning without delaying a robot's arrival time. Moreover, the trajectory was not improved locally. A global change in trajectory was achieved because the policy □□in Eq. (7) does not depend on the robot's position. Thus, if we would like to change trajectories locally to search for an optimal trajectory, we had better consider term weights $\{b_k(x,y)\}(k=1,2,3)$, which depend on a robot's position (x,y). Our updating rule can be applied to $b_k(x,y)$'s in the same way as applied to b_k.

5 Future Work

We plan to apply our method to path planning problems of multi-robot systems. We can express interactions between robots in the objective function E_v. If a user wants to move two robots while keeping them close to or avoiding each other, it is sufficient to introduce an attractive or repulsion force between the robots into E_v. This shows the flexibility that our method can be applied to many scheduling problems in the wide-range field of engineering.

References

1. Kortenkamp, D., Bonasso, R.P., and Murphy, R. (eds.), Artificial Intelligence and Mobile Robots. AAAI Press/The MIT Press (1998).
2. Hwang, Y.K. and Ahuja, N.(1992), Gross Motion Planning—A Survey. ACM Computing Surveys 24(3): 219-291
3. Igarashi, H. and Ioi, K., Path Planning and Navigation of a Mobile Robot as Discrete Optimization Problems. In Sugisaka, M. and Tanaka, H., editors, *Proceedings of the Fifth International Symposium on Artificial Life and Robotics*, ISBN4-9900462-0-X, Oita, Japan, 2000: 297-300.
4. Burgard, W., Fox, D. and Thrun, S. (1997), Active Mobile Robot Localization. In: *Proceedings of 15th Joint Conference on Artificial Intelligence (IJCAI97)*, Nagoya, Japan, August 23–29, 1997, Vol.2, pp. 1346- 1352
5. Williams, R.J. (1992), Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. Machine Learning 8: 229-256.
6. Thrun, S. et al., Map Learning and High-Speed Navigation in RHINO, in [1], pp. 21-52.