2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

Self-learning PD Algorithms Based on Approximate Dynamic Programming for Robot Motion Planning

Huiyuan Yang, Qi Guo, Xin Xu, Chuanqiang Lian
Institute of Unmanned Systems, College of Mechatronics and Automation
National University of Defense Technology
Changsha, China
yanghuiyuan1105@163.com; guoqics@gmail.com; xuxin_mail@263.net; wzdslcq@163.com

***Abstract*—Motion planning is a key technology of the navigation and control for mobile robots. However, when considering the complexity of exterior environment and mobile robot's kinematics and dynamics, the motion planning results obtained by some traditional methods are often hard to optimize. In this paper, we propose two self-learning PD algorithms to solve motion planning for mobile robots. We firstly utilize a virtual Proportional Derivative (PD) control strategy to transform the motion planning problem into an optimization problem of the virtual control policy. Afterwards, two approximate dynamic programming algorithms, which are the Least Squares Policy Iteration (LSPI) algorithm and the Dual Heuristic Programming (DHP) algorithm, are incorporated into the virtual control strategy to tune the PD parameters automatically, namely the LSPI-PD algorithm and the DHP-PD algorithm. Simulations have been performed to validate the effectiveness of the two algorithms, where the LSPI-PD algorithm is suitable for solving problems with discrete action spaces while the DHP-PD algorithm has an advantage in solving problems with continuous action spaces.**

Keywords—approximate dynamic programming; mobile robot; motion planning

I. INTRODUCTION

The problem of motion planning and control for autonomous mobile robots is to find a control law which can traverse the robot from the initial state to the destination while avoiding obstacles and obeying the system dynamics [5]-[6]. Considering the complexity of exterior environments and the uncertainty of the robot's kinematics and dynamics, motion planning problems are often separated into two submissions, involving a high-level geometric path planning and low-level control laws generated considering the robot's dynamics. The high-level path planning concentrates on the geometric solution to the shortest distance problem. Plenty of approaches have been proposed to deal with this problem, like A* algorithm and Randomized Path Planner (RPP), etc. The path it obtained is always hard to implement on the unmanned mobile robots' path tracking problem, and increases the difficulties and expenses on the low-level control. Therefore, motion planning becomes necessary in the navigation of mobile robots. The executable actions like accelerator-pedal position and steering angle are always served as outputs of motion planning. Because of accounting for the robots' kinematics and dynamics, the motion planning outcomes are often cost-effective and practically significant.

Traditional methods for mobile robots' motion planning refer to model-based approaches [7]-[9]. However these kind of model-based methods often fail in the real-world mobile robots' navigation problems, due to the fact that the accurate models of the environment and the mobile robots' dynamics are difficult to obtain. Although there are some new methods proposed, like parametric trajectory methods referring to [28]-[29], all these approaches use kinematic models in generating trajectories. As a consequence, the motion planning results also depend heavily on the accuracy of the robots' model. Sensor-based approaches are proposed to deal with the unknown environment by reacting to obstacles detected by sensors in real time [10]. But the lack of sensor information and the limitation of sensor ability may lead the robots to get lost even if a path to the goal exists. In [11], Van presented a new approach to deal with the sensing uncertainty by using the Partially Observable Markov Decision Process (POMDP). However, POMDP planning faces two major computational challenges: the "curse of dimensionality" and the "curse of history" [12]-[13]. In recent years, the intelligent computation methods play an important role in the motion planning field. Fuzzy-logic based algorithms, Artificial Neural Network (ANN) based methods, Genetic algorithm and some hybrid algorithms have been proposed referring to [14]-[19]. Reinforcement Learning (RL) is also an important intelligent computation method which provides an efficient framework to solve learning control problems which are difficult or even impossible for supervised learning methods, for it releases the requirement of the prior knowledge and the teaching instances [25]. The application of RL in the motion planning area is also drawing more and more attention by researchers.

Approximate Dynamic Programming (ADP) can be seen as an interdiscipline of RL and Dynamic Programming (DP) ideas to solve sequential decision problems which can be modeled as Markov Decision Processes (MDPs) [1]. In the past decade, the researches on RL with function approximation have been brought together with the ADP community, for they share a common object to solve MDPs with large or continuous state and action spaces [26]-[27]. Werbos [2] categorized ADP algorithm into the following major groups: heuristic dynamic programming (HDP), dual heuristic programming (DHP),

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

globalized dual heuristic programming (GDHP), and their action dependent (AD) versions [3]. DHP is the most popular one among them and has been proven to be more efficient than HDP method [4]. The Least Squares Policy Iteration (LSPI) algorithm was firstly proposed by Michail G. Lagoudakis and Ronald Parr in [23], which integrates the value function approximation with linear structure with the approximate policy iteration. It is a sample-based method which can learn the policy from samples to realize the optimal control. Generally speaking, The LSPI algorithm can also be regarded as an ADP algorithm to solve sequential decision problems. ADP algorithm also contributes to the parameter adjustment problems, due to its abilities to optimize the parameters automatically and improve their performances on line. In [20], an adaptive PID controller based on RL was designed to solve the mobile robots path tracking control problem. M. J. Er [21] combined the fuzzy control with actor neural network (actor NN) to realize the path planning, where RL was utilized to adjust the parameters in actor NN and fuzzy logic. Moreover, the ADP algorithm has a great value in the application of the motion planning field and a better performance can be expected owning to the learning feature of ADP by interacting with the outer environment and its independence on the accurate system model.

The motivation of this paper is that although ADP algorithm has been widely used in optimal control problems, the applications in the motion planning field is not that common. Therefore we design two self-learning PD algorithms to implement the optimization of the virtual control policy in motion planning, namely the LSPI-PD method and the DHP-PD method, respectively. On account of the characteristics of the LSPI and DHP algorithms, the LSPI-PD algorithm is particularly suitable for dealing with motion planning problems with discrete action spaces, while the DHP-PD algorithm can deal with problems with continuous action spaces.

The main contribution of the paper stems from the fact that we firstly design a virtual PD control strategy to transform the motion planning problem into an optimization problem. Then by formulating the motion planning problem as a Markov decision process, we can incorporate the LSPI and DHP algorithms into the virtual PD control strategy to realize the adjustment of PD parameters automatically and optimally. Taking advantage of the characteristics that ADP method can improve the optimization ability by interacting with the environment, the usage of the LSPI-PD algorithm and the DHP-PD algorithm possess motion planning for mobile robots of the self-learning and self-adaptive ability. The advantages of the proposed algorithms over those methods we mentioned above lie in the fact that they don't need the accurate model or the prior knowledge, and by constantly interacting with the unknown environment, they can improve the performance when considering the uncertainties in the real word.

The rest of this paper will be organized as follows: in section II, we formulate the kinematics model of the mobile robot and build up the virtual PD control strategy in motion planning. In section III, by introducing the LSPI and DHP algorithms into the virtual PD control strategy, we propose the LSPI-PD and DHP-PD algorithms to solve motion planning. Simulation results are given in section IV, as well as the performance comparison between the proposed algorithms and the PD control strategy with fixed parameters. Section V comes to the conclusions.

II. PROBLEM FORMULATION

Considering the nonlinear kinematics model of the mobile robot discussed in [22]:

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \frac{v}{L}\tan\delta \cdot G_{ss} \\ \dot{\delta} &= \frac{1}{T_d}(\delta_c - \delta) \end{aligned} \tag{1}$$

where θ is the direction angle of the robot. L is the length of the robot's body. δ is the average steering angle of the front wheels. δ_c is the steering command. The steering rate can be computed by the time constant T_d. G_{ss} is the influence of the side slipping and is ignored in this paper, as G_{ss} =1.

The goal region of motion planning is assumed as given by a high-level geometric path planning result. Then a low-level controller generates the robot's control laws while considering the robot's dynamics. The target of motion planning is based on these control inputs, the mobile robot can realized the maneuvering tracking of the reference path with less tracking error.

The motion planning flowchart based on the virtual PD control strategy is as Fig.1 shows, where the reference r is a 2D path which is the outputs of the high-level path planning, $u(t)$ is the motion planning results and $x(t+1)$ stands for the future state of the mobile robot at the next time step. In order to utilize the ADP algorithm to realize motion planning, a virtual PD controller is introduced after the high-level geometric path planning. It can transform the motion planning problem into an optimal control problem and output a sequence of actions for the mobile robot. That is to say the sequence of expected path points is transferred into a sequence of executable actions.

The virtual PD controller runs motion planning over closed-loop dynamics as the second block shows in Fig.1. This

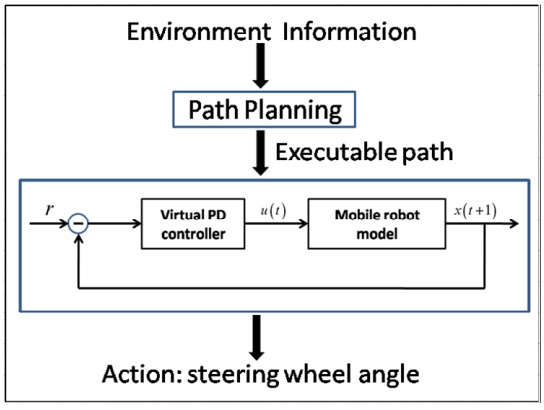


Fig. 1. The flowchart of motion planning based on the virtual PD control strategy.

closed-loop approach takes not only the environment, but also the robot's dynamics into consideration. The virtual control law is defined as $u(t)=k_p\times x_e(1)+k_d\times x_e(2)$, where $x_e(1)$ is the error term and $x_e(2)$ is the derivation term of the virtual control strategy. A more detail illustration is demonstrated in Fig.2.

We only consider the situation where the robot keeps a constant speed. d_{path} is the vertical distance from the center of the robot to the reference path. $\delta_c(t)$ is the expected steering angle of front wheel at time-step t. $v_{vert}(t)$ is the velocity component in vertical direction. Then we have:

$$x_e(1) = d_{path}, \quad x_e(2) = v_{vert}, \quad u(t) = \delta_c(t) \tag{2}$$

$$\delta_c(t) = k_p \times d_{path}(t) + k_d \times v_{vert}(t) \tag{3}$$

According to the kinematics function in (1), we have:

$$\dot{\delta}(t) = \frac{1}{T_d}(\delta_c(t) - \delta(t-1)) \tag{4}$$

$$\delta_c(t) = \delta(t-1) + ds \times \dot{\delta}(t) \tag{5}$$

where T_d is the time constant and ds is the simulation step size. $\dot{\delta}(t)$ is the output as the motion planning result.

III. Self-learning PD Algorithms Based on ADP for Motion Planning

A. The MDP model of self-learning PD algorithms based on ADP for motion planning

MDP is denoted as a 4-tuple $\{S\ A,\ R,\ P\}$, where S is the state space, A is the action space, P is the state transition probability and R is the reward function. The policy of the MDP is defined as a function π: $X\rightarrow P_r(A)$, where $P_r(A)$ is a probability distribution in action spaces. The LSPI and DHP algorithms are to estimate the optimal policy $\pi^*(a|x)$, which satisfies the following equation:

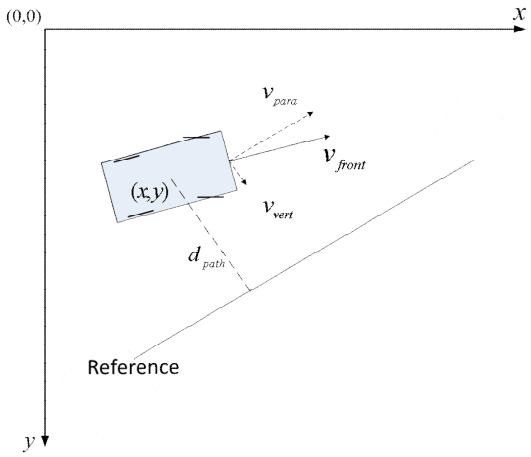


Fig. 2. The illustration of the error term and the derivation term in the virtual PD control strategy.

$$J_{\pi^*} = \max_{\pi} J_{\pi} = \max_{\pi} E_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t] \tag{6}$$

where γ is the discount factor, r_t is the reward at time-step t and $E_\pi[\cdot]$ stands for the expected total reward.

In order to use the LSPI and DHP algorithms to adjust the PD parameters in motion planning, we firstly have to formulate the optimal control problem as a MDP model.

The MDP model of self-learning PD algorithms for motion planning based on ADP is formulated in TABLE I, where d_{path} is the vertical distance from the center of the robot to the reference path, v_{vert} is the velocity component in the vertical direction, v_{para} is the velocity component in horizontal direction and θ_{path} is the orientation error between the direction of the robot and the reference path. These four components compose the state space S. The action space of LSPI-PD is three sets of PD parameters selected as a rule of thumb in advance, while the action space of DHP-PD is the entire continuous $\mathbb{R}^2$ space.

By using a virtual PD control strategy, the motion planning problem can be transformed into an optimal control problem of PD parameters. The negative of the cumulative tracking error is defined as the value function. By using ADP methods to maximize the value function, which means to minimize the cumulative tracking error, we can obtain the optimal control policies, namely the optimal PD parameters. Then the outputs of the virtual PD control strategy can be served as the motion planning results.

B. The self-learning PD algorithm based on LSPI for motion planning

The target of LSPI is to use a set of linear basis functions to approximate the expected total reward $Q(s,\ a)$ under optimal policy. The LSPI method is a sample-based algorithm. Before using LSPI to tune the PD parameters, we have to collect samples to train the weights in the approximate state-action value function:

$$\hat{Q}(s,a,w) = \sum_{i=1}^{L} \phi_i(s,a) w_i \tag{7}$$

where $\phi_i(s,\ a)$ is the linear basis function, w_i is the weight to be estimated. The sample policy can be arbitrary, either by using a conventional controller or just by observing the MDP

TABLE I. The MDP model of self-learning PD algorithms based on ADP for motion planning

State Space	$[d_{path}, v_{vert}, v_{para}, \theta_{path}]$	
	LSPI-PD	**DHP-PD**
Action Space: PD Parameters [*kp*, *kd*]	*action* 1: $a1$=[0.03, 0.0008] *action* 2: $a2$=[0.03, 0.004] *action* 3: $a3$=[0.05, 0.0018]	Generate *kp* and *kd* continuously
Reward	$r = -d_{path}$	

running with a random action policy. The sample process is shown in TABLE II, where the *PDModel* refers to the mobile robot's kinematics model adopted by the virtual PD control strategy. Then the state transition process can be presented as $s(t+1)=PDModel(s(t),a(t))$.

After sampling, we have to firstly train the state-action value function before solving motion planning. Referring to [23], the state-action value function $Q(s, a)$ is the fixpoint of Bellman operator T_π, where

$$(T_\pi Q)(s,a) = R(s,a)+\gamma\sum_{s'\in S}P(s,a,s')\sum_{a'\in A}\pi(a';s')Q^\pi(s',a') \quad (8)$$

The estimation $\hat{Q}(s,a,w)$, which approaches $Q(s, a)$ with high precision, also can meet the fixpoint condition of Bellman operator, which means $T_\pi\hat{Q}^\pi \approx \hat{Q}^\pi$. By letting

$$\vec{\phi}(s,a)=\begin{pmatrix}\phi_1(s,a)\\ \phi_2(s,a)\\ \cdots\\ \cdots\\ \phi_L(s,a)\end{pmatrix} \quad \Phi=\begin{pmatrix}\phi(s_1,a_1)^T\\ \cdots\\ \phi(s,a)^T\\ \cdots\\ \phi(s_{|S|},a_{|A|})^T\end{pmatrix}$$
$$\hat{Q}(w)=\begin{pmatrix}\hat{Q}(s_1,a_1,w)^T\\ \cdots\\ \hat{Q}(s,a,w)^T\\ \cdots\\ \hat{Q}(s_{|S|},a_{|A|},w)^T\end{pmatrix} \quad (9)$$

we can compute the weights in the approximate state-action value function $\hat{Q}(s,a,w)$. The more detail computing process is listed in TABLE III. Then for every current state of the robot, by using a greedy policy, we can select an action, namely a set of PD parameters in the predefine action sets, to maximize the approximate state-action value function $\hat{Q}(s,a,w)$. The motion planning results are the outputs of the virtual PD control strategy. The proposed LSPI-PD algorithm for motion planning is shown in TABLE III.

C. *The self-learning PD algorithm based on DHP for motion planning*

In DHP structure, there is a critic and an actor where the actor estimates the optimal control policy and the critic estimates the derivatives of the value function with respect to states. Neural networks (NNs) can be used as approximation structure for the critic and the actor, which are CNN and actor NN, separately:

TABLE II. THE SAMPLING PROCESS FOR THE LSPI-PD ALGORITHM

1: **input**: reference path, robot's initial position.
2: **for** each sample sequence i, **do**
3: **for** each sample in the sequence i at time t, **do**
4: Choose an action randomly in the predefined action set, $action \in \{a1, a2, a3\}$.
5: Then we have $kp(t)=action(1)$; $kd(t)=action(2)$.
6: Calculate the state transition: $s(t+1)=$PDModel $(s(t), ([kp(t), kd(t)]))$
7: Get the reward at time t: $r(t)= -d_{path}(t)$
8: Sample$(i, t)=[s(t), ([kp(t), kd(t)]), r(t)]$.
9: **end for**
10: add Sample(i) into the sample set: Samples=[Samples Sample(i)].
11: **end for**
12: **output**: Samples

$$\lambda(x_t)=\frac{\partial V(x_t)}{\partial x_t}=\sum_{i=1}^{l}\phi_i(x_t)w_i \quad (10)$$

$$a(x_t)=\sum_{j=1}^{M}\sigma_j(x_t)\theta_j \quad (11)$$

where w_i and θ_j are the weight vectors, x_t is the input state, and $\boldsymbol{\phi}(x_t)=[\phi_1(x_t), \ldots, \phi_l(x_t)]^T$ and $\boldsymbol{\sigma}(x_t)=[\sigma_1(x_t), \ldots, \sigma_M(x_t)]^T$ are vectors of basis functions. TABLE IV shows the proposed DHP-PD algorithm for motion planning.

DHP utilize the samples of trajectory gathered from the motion planning simulation of the mobile robot to train the critic network and actor network. The learning algorithm in the critic of DHP is based on the Bellman recursion of value function's derivatives:

TABLE III. THE LSPI-PD ALGORITHM FOR MOTION PLANNING

1: **input:** A_0, b_0, sample sequence, reference path and the robot's initial state.
2: **for** each sample sequence i, **do**
3: **do** each sample in the sequence i at time t:
4: Update A_t, b_t by using:
$A_{t+1} = A_t + \vec{\phi}(x_t,a_t)(\vec{\phi}(x_t,a_t)-\gamma\vec{\phi}(x_t,a_t'))^T$
$b_{t+1} = b_t + \vec{\phi}(x_t,a_t)r_t$
5: Compute w_{t+1} by using: $w_{t+1}=A_{t+1}^{-1}b_{t+1}$
6: **until** $|\tilde{w}_{t+1}-\tilde{w}_t|<\varepsilon$.
7: **end for**
8: Output the approximate state-action value function:
$$\hat{Q}(s,a,w)=\sum_{i=1}^{L}\phi_i(s,a)w_i$$
9: **for** every current state in the time order, **do**
10: Compute the state-action value function $\hat{Q}(s,a,w)$.
11: Select an action by using greedy policy:
$$a=[k_p(t), k_d(t)]=\arg\max_{a\in A}\hat{Q}(s,a,w)$$
12: Compute the state transition:
$s(t+1)=$PDModel $(s(t), ([kp(t), kd(t)]))$
13: **end for**
14: **output:** the policy: *Policy*(a, $\hat{Q}(s,a,w)$), where the *action* a=[$kp(t)$, $kd(t)$].

TABLE IV. THE DHP-PD ALGORITHM FOR MOTION PLANNING

input: reference path and robot's initial state.
Create actor network and critic network.
for each sample sequence *i*, **do**
  Initialize the robot's state;
  for each sample in the sequence *i* at time *t*, **do**
    Compute *kp* and *kd* by using actor NN:

$$a(x_t)=[k_p(t),\ k_d(t)]=\sum_{j=1}^{M}\sigma_j(x_t)\theta_j$$

    Compute the state transition:
      $s(t+1)$=PDModel ($s(t)$, ([$kp(t)$, $kd(t)$]))
    Compute CNN by (10).
    Update CNN:

$$w_{k+1}=w_k+\alpha\left[\lambda(x_t)-\left(\frac{\partial R_t}{\partial x_t}+\gamma\lambda(x_{t+1})\frac{\partial x_{t+1}}{\partial x_t}\right)\right]\frac{\partial\lambda(x_t)}{\partial w}$$

    Update actor NN:

$$\theta_{t+1}=\theta_t-\eta\Delta\theta_t=\theta_t-\eta\frac{\partial V(x_{t+1})}{\partial a(x_t)}\frac{\partial a(x_t)}{\partial\theta_t}$$

    if $|w_{t+1}-w_t|<\varepsilon$ **then**
      seccessful_flag=1, break.
    else if time *t* exceeds the limitation **then** break.
    end if
    if the training episode *i* exceed the limitation **then**
      The training fails, return to step 1.
    else if seccessful_flag=1 **then** break.
    end if
  end for
end for
output: actor NN.

$$\lambda(t)=\frac{\partial J(x_t)}{\partial x_t}=\frac{\partial R(x_t,a_t)}{\partial x_t}+E\left[\frac{\partial J(x_{t+1})}{\partial x_t}\right] \tag{12}$$

where $E[\cdot]$ is with respect to a stationary state transition probability and $R(x_t,a_t)$ is the expected single-step reward. When the weight vector of CNN converges, policy gradient learning is performed based on the outputs of the critic to train the actor NN. Then the trained actor NN can be used to realize the optimization of PD parameters in the motion planning problem.

IV. SIMULATION RESULTS AND COMPARATIVE DISCUSSION

In this section, we present the simulation results of the proposed self-learning PD algorithms for motion planning. We firstly test the validity of the algorithms with a simple straight line as the pre-planned path, and then come to some more complicated situations. In the simulations, the mobile robot can realize the maneuvering tracking of the pre-planned path based on the motion planning results with less tracking error.

A. Motion planning by using LSPI-PD algorithm

The simulation is implemented on a map with 40m×40m size. The kinematics model of the mobile robot is as (1) shows. The initial angle of the robot is 0.785rad and the velocity is set to v=10m/s constantly. Based on the manual adjustment experiment, three sets of PD parameters can be selected as candidates, which are shown in TABLE I. We collected 10 sample sequences. Every sequence contained 1000 samples and every 5 samples adopt an action randomly. We firstly use these sample sequences to train the weights in the approximate state-action value function. Then the trained virtual PD control strategy based on LSPI can be used to realize motion planning. The simulation result is shown in Fig.3. The process of the PD parameters adjustment is shown in Fig.4. It illustrates that the LSPI-PD algorithm can tune the PD parameters by switching automatically in the pre-defined discrete action spaces.

According to the feature of the LSPI algorithm, LSPI-PD adjusts the PD parameters discretely by choosing between some preselected PD parameter sets, which also have to be determined through the prior knowledge and experience in advance. Therefore, the performance of the LSPI-PD algorithm is somehow determined by the quality of the chosen PD parameter sets. In order to eliminate the human labors and extend the application of self-learning PD algorithm to the problems with continuous action spaces, the DHP-PD algorithm is tested below.

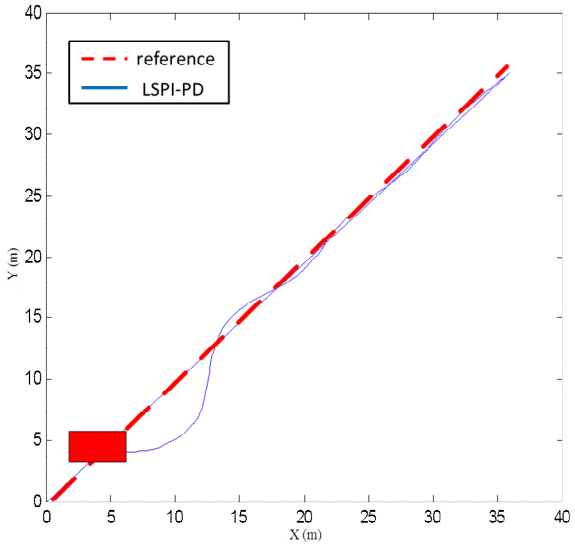


Fig. 3. The maneuvering trajectory by using the LSPI-PD method.

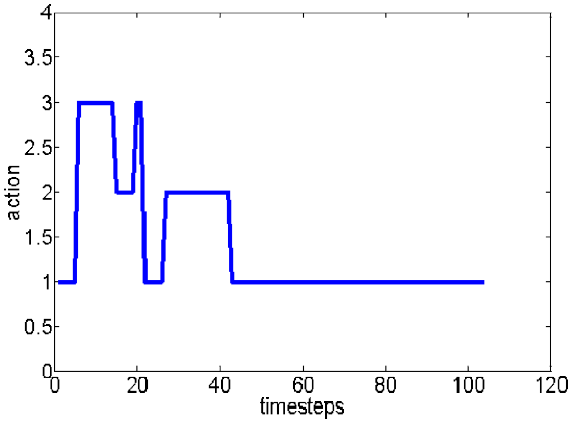


Fig. 4. The process of PD parameters adjustment by using LSPI-PD.

B. Motion planning by using DHP-PD algorithm

When the pre-planned path is a straight line, we assume the simulation settings in this part are the same as the simulations of the LSPI-PD algorithm. Only the initial angle of the mobile robot is increased to 1rad. In order to highlight the advantages of the DHP-PD algorithm, we select two sets of fixed PD parameters by experiences, which are PD1: $[k_p(t), k_d(t)] = [0.1, -0.8]$ and PD2: $[k_p(t), k_d(t)] = [0.2, -0.5]$, and compare their performances with the DHP-PD method. We plot the simulation results together in Fig.5. From the figure we can see apparently that the motion planning results by using the DHP-PD method can drive the robot to track the reference path quicker and with less tracking error than the PD control strategy with fixed parameters.

In order to test the validity of the algorithm in more complicated situations, like the pre-planned path is no longer a straight line, or there are obstacles in the map. We firstly modified the robot's dynamic model and adopt the incremental PID algorithm as the virtual control strategy.

Firstly we calculate the tracking error as the inputs for the virtual control strategy. Referring to [30], the tracking error model of the robot can be formulated as:

$$\begin{cases} \dot{e}_x = \omega e_y - v + v_r \cos e_\theta \\ \dot{e}_y = -\omega e_x + v_r \sin e_\theta \\ \dot{e}_\theta = \omega_r - \omega \end{cases} \tag{13}$$

And we rewrite (13) in the discrete-time form as:

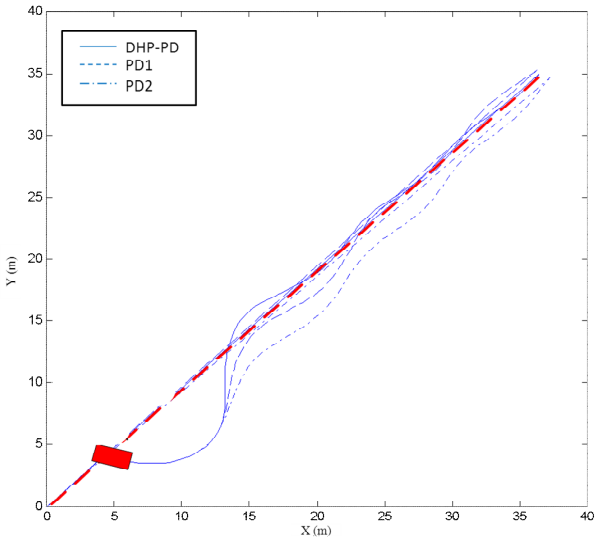


Fig. 5. The comparison of maneuvering trajectories between using the DHP-PD method and the PD method when the initial angle is 1rad.

$$e_{k+1} = e_k + Ts * \begin{bmatrix} \omega e_y(k) - v + v_r \cos e_\theta(k) \\ -\omega e_x(k) + v_r \sin e_\theta(k) \\ \omega_r - \omega \end{bmatrix} \tag{14}$$

where Ts is the simulation step size, $e = (e_x, e_y, e_\theta)^T$ is the tracking error state, $(x_r, y_r, \theta_r)^T$ is the reference path. The state vector $s = (x, y, \theta)^T$ stands for the position of the mobile robot, and the control input $u = (v, \omega)^T$ consists of the velocity v and angle speed ω.

Then the incremental PID algorithm can be written as:

$$\begin{aligned} v(k) &= v(k-1) + k_1 \times \left(e_x(k) - e_x(k-1)\right) + k_2 \times e_x(k) \\ &\quad + k_3 \times \left(e_x(k) - 2e_x(k-1) + e_x(k-2)\right) \\ \omega(k) &= \omega(k-1) + k_4 \times \left(e_y(k) - e_y(k-1)\right) + k_5 \times e_y(k) \\ &\quad + k_6 \times \left(e_y(k) - 2e_y(k-1) + e_y(k-2)\right) \\ &\quad + k_7 \times \left(e_\theta(k) - e_\theta(k-1)\right) + k_8 \times e_\theta(k) \\ &\quad + k_9 \times \left(e_\theta(k) - 2e_\theta(k-1) + e_\theta(k-2)\right) \end{aligned} \tag{15}$$

where $[k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9]$ are the parameters to be optimized. To simplify the optimization process, we assume the parameters k_3, k_5, k_6, k_8 and k_9 are all zeros. We only optimize k_1, k_2, k_4, and k_7 by using the DHP-PD algorithm showing in TABLE IV.

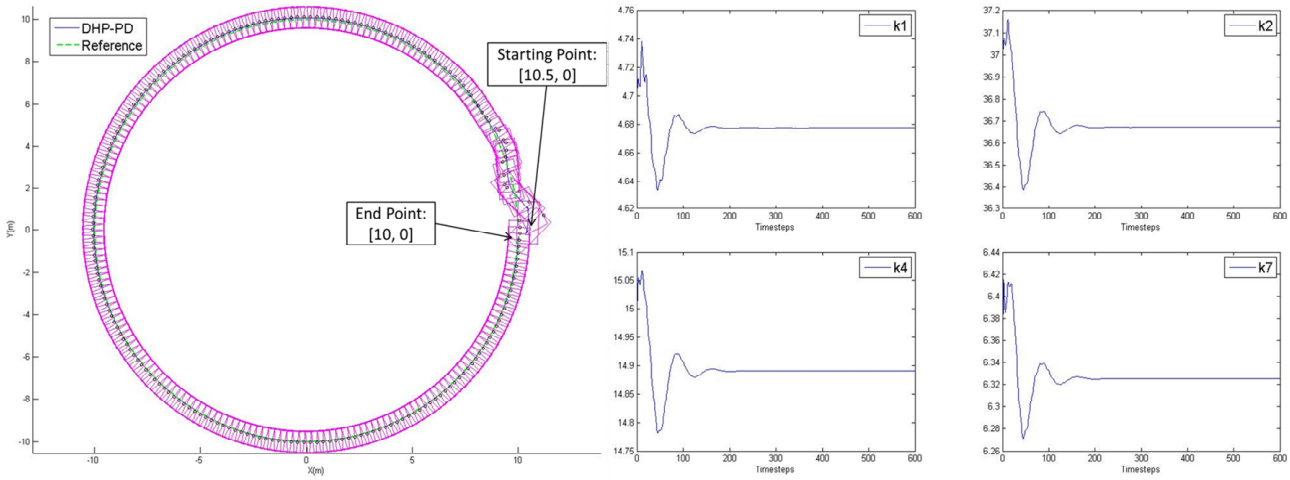
The first reference path is a circle which is defined as follows:

$$\begin{cases} x_r = 10\cos(0.5t) \\ y_r = 10\sin(0.5t) \end{cases} \tag{16}$$

The initial position of the robot is [10.5, 0, pi/4], and the initial tracking error is [-0.25, 0.46, 0.8]. The simulation results are shown in Fig.6.

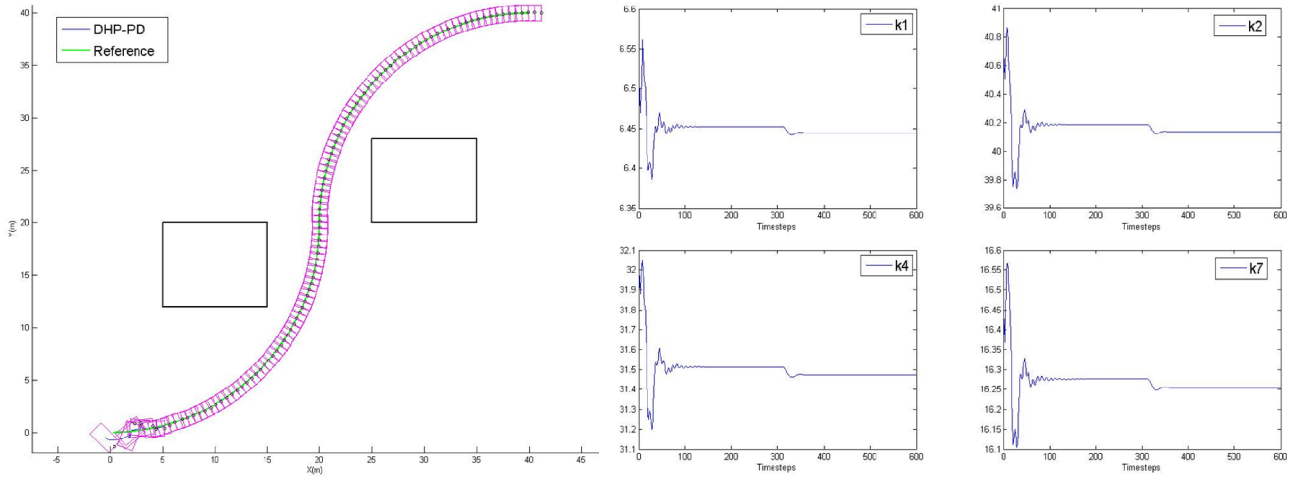
When there are obstacles, we firstly adopt a high-level path planning method to get a feasible path in advance and then use the DHP-PD method to solve motion planning. The simulation results are shown in Fig.7, where the rectangles on the left and right sides of the picture stand for the obstacles. The initial position of the robot is [-0.5, -0.5, -pi/4], and the initial tracking error is [0.21, 0.92, 0.8].

The main difference between the two self-learning PD algorithms lies in the fact that the DHP-PD method inherits the characteristics from the DHP algorithm and thereby is more suitable for dealing with problems with continuous action spaces. Given the fact that motion planning for mobile robots is a problem with continuous action spaces, the DHP-PD method can generate the PD parameters smoothly which, comparing to the LSPI-PD algorithm in a discrete way, consequently more suitable for solving motion planning for mobile robots.



(a) The trajectories (b) The adjustment of parameters

Fig. 6. The maneuvering tracking of circle path by using the DHP-PD method.



(a) The trajectories (b) The adjustment of parameters

Fig. 7. The maneuvering tracking of pre-planned path when avoiding obstacles by using the DHP-PD method.

V. CONCLUSION

In this paper, we have proposed two self-learning PD algorithms to solve the motion planning problem, namely the LSPI-PD algorithm and the DHP-PD algorithm. Firstly, by utilizing a virtual PD control strategy, we transform the motion planning problem into an optimization problem of the virtual control policy. Then the LSPI and DHP algorithms are incorporated into the virtual PD control strategy respectively to realize the adjustment of the PD parameters automatically. The LSPI-PD algorithm improves the performance by optimally switching among some predefined parameter sets, which is a kind of RL method with discrete action spaces. The DHP-PD algorithm generates the PD parameters continuously, which is more suitable for solving problems with continuous action spaces. Simulations have been done and the results demonstrate the effectiveness of both the proposed self-learning PD algorithms for motion planning of mobile robots.

REFERENCES

[1] Barto, A. G., Powell, W. B., & Wunsch, D. C. (Eds.). (2004). *Handbook of learning and approximate dynamic programming* (pp. 125-151). Los Alamitos: IEEE Press.

[2] Werbos, P. (1992). Approximate dynamic programming for real-time control and neural modeling. In White & Sofge (Eds.), Handbook of intelligent control (pp. 493–525). New York, USA: V.N. Reinhold.

[3] D.V. Prokhorov, D.C. Wunsch, Adaptive critic designs, IEEE Transactions Neural Networks 8 (5) (1997) 997–1007.

[4] G.K. Venayagamoorthy, R.G. Harley, D.C. Wunsch, Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for 1399 neurocontrol of a turbogenerator, IEEE Transactions on Neural Networks 13 (3) (2002) 764–773.

[5] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun,Principles of Robot Motion: Theory, Algorithms, and Implementations. Cambridge, MA: MIT Press, 2005.

[6] S. M. LaValle, Planning Algorithms. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[7] J. Latombe, Robot Motion Planning. Norwell, MA: Kluwer, 1991.

[8] N. Nilsson, "A mobile automation: An application of artificial intelligence techniques," in Proc. 1st Int. Joint Conf. Artificial Intelligence, Washington, DC, 1969, pp. 509–520.

[9] J. Mitchell, "Planning Shortest Paths," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1986.

[10] X. Yang, R. V. Patel, and M. Moallem, "A fuzzy-Braitenberg navigation strategy for differential drive mobile robots," in Proc. 3rd IFAC Symp. Mechatronic Systems, Sydney, Australia, Sep. 2004.

[11] Van Den Berg J, Pail S, Alterovitz R. Motion planning under uncertainty using iterative local optimization in belief space[J] The International Journal of Robotics Research, 2012, 31(11): 1263-1278.

[12] Kurniawati H, Hsu D and Lee WS (2008) SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. Proceedings of the Robotics: Science and Systems.

[13] Kurniawati H, Du Y, Hsu D, et al. Motion planning under uncertainty for robotic tasks with long time horizons[J]. The International Journal of Robotics Research, 2011,30(3):308-323.

[14] C. Juang, Y. Chang. Evolutionary-Group-Based Particle-Swarm-Optimized Fuzzy Controller With Application to Mobile-Robot Navigation in Unknown Environments [J]. IEEE Transactions on Fuzzy Systems, 2011, 9(2): 379~392.

[15] M. Garcia, O. Montiel, O. Castillo, et al. Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation[J]. Applied Soft Computing, 2009, 9: 1102~1110.

[16] Mahmoud Tarokh. Hybrid intelligent path planning for articulated rovers in rough terrain[J]. Fuzzy Sets and Systems, 2008, 159: 2927~2937.

[17] A. Nazemi, F. Omidi. An efficient dynamic model for solving the shortest path problem[J]. Transportation Research Part C, 2013, 26: 1~19.

[18] J. Ni , S. X. Yang. Bioinspired Neural Network for Real-Time Cooperative Hunting by Multirobots in Unknown Environments[J]. IEEE Transactions On Neural Networks, 2011, 22(2): 2062~2077.

[19] V. Roberge, M. Tarbouchi, and G. Labonté. Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real-Time UAV Path Planning[J]. IEEE Transactions On Industrial Informatics, 2013, 9(1): 132~141.

[20] Peng J. and R.J.Williams. Incremental multi-step Q-learning[J]. Machine Learning, 1996, 11: 283-290.

[21] M J Er, Chang Deng. Obstacle Avoidance of a Mobile Robot Using Hybrid Learning Approach [J]. IEEE Transactions On Industrial Electronics, 2005, 52(3): 898-905.

[22] Y. Kuwata, J. Teo and G. Fiore. Real-time Motion Planning with Applications to Autonomous Urban Driving[J]. IEEE Transactions On Control Systems Technology, 2009, 17: 1105-1118.

[23] M. G. Lagoudakis, R. Parr. Least-Squares Policy Iteration[J]. Journal of Machine Learning Research, 2003, 4: 1107-1149.

[24] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks,"Neural Netw., vol. 3, no. 5, pp. 551–560, 1990.

[25] Xin Xu, Zhenhua Huang, Lei Zuo. "Reinforcement learning algorithms with function approximation: Recent advances and applications". Information Science, 2013, in press.

[26] W.B. Powell, Approximate Dynamic Programming: Solving the Curses of Dimensionality, Wiley, NY, 2007.

[27] F.Y. Wang, H. Zhang, D. Liu, Adaptive dynamic programming: an introduction, IEEE Computational Intelligence Magazine (2009) 39–47.

[28] Yuan, H. and Qu, Z., "Optimal Real-time Collision-Free Motion Planning for AUVs in a 3D Underwater Space," IET Control Theory and Applications, Vol. 3, No. 6, pp. 712-721, 2009.

[29] Yuan H, Shim T. Model Based Real-Time Collision-Free Motion Planning for Nonholonomic Mobile Robots in Unknown Dynamic Environments[J]. International Journal of Precision Engineering and Manufacturing, 2013, 14(3): 359-365.

[30] Gu D, Hu H. Receding Horizon Tracking Control of Wheeled Mobile Robots[J]. Control Systems Technology, IEEE Transactions on, 2006, 14(4): 743-749.