Research Article

Robot Obstacle Avoidance Learning Based on Mixture Models

Huiwen Zhang,^{1,2} Xiaoning Han,^{1,2} Mingliang Fu,^{1,2} and Weijia Zhou¹

¹State Key Laboratory of Robotics, Shenyang Institute of Automation, Shenyang 110016, China

²University of Chinese Academy of Sciences, Beijing, China

Correspondence should be addressed to Huiwen Zhang; zhanghuiwen@sia.cn

Received 27 May 2016; Accepted 21 July 2016

Academic Editor: Keigo Watanabe

Copyright © 2016 Huiwen Zhang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We briefly surveyed the existing obstacle avoidance algorithms; then a new obstacle avoidance learning framework based on learning from demonstration (LfD) is proposed. The main idea is to imitate the obstacle avoidance mechanism of human beings, in which humans learn to make a decision based on the sensor information obtained by interacting with environment. Firstly, we endow robots with obstacle avoidance experience by teaching them to avoid obstacles in different situations. In this process, a lot of data are collected as a training set; then, to encode the training set data, which is equivalent to extracting the constraints of the task, Gaussian mixture model (GMM) is used. Secondly, a smooth obstacle-free path is generated by Gaussian mixture regression (GMR). Thirdly, a metric of imitation performance is constructed to derive a proper control policy. The proposed framework shows excellent generalization performance, which means that the robots can fulfill obstacle avoidance task efficiently in a dynamic environment. More importantly, the framework allows learning a wide variety of skills, such as grasp and manipulation work, which makes it possible to build a robot with versatile functions. Finally, simulation experiments are conducted on a Turtlebot robot to verify the validity of our algorithms.

1. Introduction

With the development of robot technology, especially the development of computer science and cognitive science, some promising technologies come out, such as deep learning and computer vision. These technologies greatly improve the intelligence level of robots and also laid the foundation for the modern intelligent robots. Unlike the traditional mature industrial robots, intelligent robots are expected to coexist with people, such as service robots. To achieve this purpose, robots are required to have strong environmental adaptability and autonomous decision-making ability. Obstacle avoidance and planning are basic skills of robots, which are also research focus in the field of robotics. There have been a lot of mature algorithms for obstacle avoidance, but how to develop obstacle avoidance algorithm with strong robustness, especially algorithms that can make robots fulfill task under unstructured environment, is still worthy of further research.

The obstacle avoidance algorithms can be classified into traditional obstacle avoidance algorithms and intelligent algorithms based on the developments history. In [1]

the authors concluded them as classical approaches and heuristic approaches. In [2] the authors called them classical algorithms and evolutionary algorithms. Here the traditional approaches cover the same content, while the intelligent algorithms combine some machine learning methods applied to robot obstacle avoidance in recent years, which is different from the terms coined by other authors and covers a wider range. Classical obstacle avoidance algorithms can be categorized into three classes based on their main idea: (i) methods based on geometric model, (ii) methods based on virtual field, and (iii) methods based on mathematical programming. Details of classic obstacle avoidance algorithms can be found in [3]. Methods based on geometric models can be divided into two steps. First build the geometric model of the environment, and then choose the right kind of search algorithms. The main idea of virtual field methods is to assume that all robots stay in a virtual field, the change of virtual field reflects the feasible solution for the path. This concept is first put forward and used in [4, 5]. Mathematical programming methods represent constraints for obstacle as some parameterized inequities in configuration space, so

a geometric problem is turned into a mathematical problem; the best obstacle-free path is a solution for the inequities which can be solved by optimization methods. All these methods have been successfully used, but the traditional method generally is only applicable to the static environment and easy to fall into a local optimal solution. To solve these problems, more and more intelligent approaches are used. By 2003, more than half of articles use heuristic algorithms to solve obstacle avoidance problem [1].

The core idea of heuristic algorithms is to simulate some intelligent behaviors of animals or people. Algorithms are designed by bionic mechanism. There are a large number of representative algorithms, such as Ant Colony Optimization (ACO) algorithm, Particle Swarm Optimization (PSO) Algorithm, and Genetic Algorithm (GA). In [6] ACO is used as a global path planning method to avoid obtaining a local optimal solution when using potential field method. In [7] distance and memory feature are introduced to speed up the computation of the ACO. The emergence of PSO derived from the simulation of birds migrating behavior. Applications of particle swarm optimization to solve the path planning problem include [8–10]. GA, derived from the theory of evolution and genetics, is a kind of global probabilistic optimization algorithm. In [11] a special encoding technology is used to speed up the path generation. The fitness function is designed according to security and distance relative to a target, and then a barrier-free path is solved through a genetic algorithm. In [12], a variable-length chromosome genetic algorithm is designed for path planning. In [13], the authors use artificial network to solve the obstacle avoidance problem of a redundant manipulator. An improved Grossberg neural network method is used in [14] to solve the UAV path planning under a “danger mode.” Neural network can also realize the obstacle avoidance in conjunction with other methods, such as fuzzy neural network learning [15]. Among all the intelligent algorithms, there are broad categorical approaches which are based on the learning. A representative work is using reinforcement learning. Reinforcement learning derives from the psychology. The main idea is through constant interaction with the environment and “trial and error” to maximize the reward defined by a reward function [16]. Based on this concept, a lot of obstacle avoidance algorithms are proposed [17–22]. It is worth noting that, with a great improvement of hardware and the development of deep learning theory, some methods can directly learn obstacle avoidance strategy from the sensor data, called end-to-end learning [23]. Intelligent algorithms usually can obtain the global optimal solution and have good ability to adapt to the environment. But often these methods bear high computational complexity and bad real-time performance.

In this paper, we proposed a kind of obstacle avoidance method based on teaching and learning. First, learning sample is obtained by showing robots how to avoid the obstacle and then using probabilistic mixture model for extraction and encoding task constraints autonomously. Finally, obstacle-free path is generated by GMR. Our approach differs from previous work in the following: (i) it is more in line with the habit of human cognition, drawing lessons from imitation learning and (ii) the algorithm has good generalization

ability and can adapt to dynamic unstructured environment. Besides, the computational complexity is low, and in particular the path generation process can be realized online efficiently; (iii) the learning framework not only can be used to learn obstacle avoidance but also can be used for learning other skills, which is a key point to achieve generalized artificial intelligence (GAI).

The main contributions of the paper are as follows: (i) the first contribution is using LfD for the obstacle avoidance task for the first time, making up for the disadvantages of other methods; (ii) due to the nature of LfD, nontechnical people can programme the robots to realize obstacle avoidance function; (iii) obstacle avoidance learning using mixture models is able to adapt the environmental changes and dynamic obstacles and at the same time shares low computational complexity.

The remainder of the paper is organized as follows. In Section 2, some background knowledge is introduced, including the concepts of LfD and GMM. Section 3 introduces parameters solving methods for GMM. Section 4 gives a detailed introduction of the obstacle avoidance learning architecture. In Section 5, simulation experiments based on a Turtlebot robot are conducted. In the end, conclusion is made and further improvements are introduced.

2. Learning from Demonstration

LfD is a paradigm which allows robots to automatically acquire new skills. Unlike the traditional robots, to fulfill a task, a previous task decomposition work and programming for each action need to be done by humans. LfD adopts the point that robot's skills can be acquired by observing the human operation. This approach has the advantages of simple programming and even people who do not understand programming technique can programme robots through teaching. Besides, due to the nature of LfD, robots are more advantageous to adapt to new environment. LfD is also named robot Programming by Demonstration (PbD) (Imitation Learning and Apprenticeship Learning), Learning by Demonstration (LbD), Learning by Showing, Learning by Watching, Learning from Observation, and so forth. It is worth noting that some authors think that imitation learning and LfD have different meanings; in [24], the authors distinguish imitation learning and LfD according to how learning data is recorded and how the imitator uses the data in the learning process.

2.1. Formulation of LfD Problem. Solving process for LfD can be divided into two steps. First, acquire training data; second, use the obtained data to derive a control policy. We use S and A to represent state set and action set, respectively. The probability distribution function $T(s' | s, a) : S \times A \times S \rightarrow [0, 1]$ represents the transfer probability from state (s, a) to state s' . An experience dataset D should include this information. Actually, in most situations, the true states S are unknown, the observable states are Z , and then a learning problem can be defined as deriving a policy π mapping from Z to A . For an obstacle avoidance task, the problem can

FIGURE 1: Framework of learning from demonstration (LfD).

be described as finding the more general barrier-free path (policy) from the demonstrated barrier-free path (sample D). The main process for LfD framework is illustrated in Figure 1.

2.2. Main Research Topics in LfD. Nehaniv and Dautenhahn [25] summarized the main problems in LfD as four issues: (1) What to imitate? (2) How to imitate? (3) When to imitate? (4) Whom to imitate? So far, only the former two problems are solved. For the problem of robot obstacle avoidance, what to imitate means the robot needs to extract the right constraints which can characterize the task, and problem of how to imitate refers to generating a trajectory that satisfies the constraints obtained in the previous step.

In order to solve the above problems (mainly the former two problems), there are three main ideas:

- (i) Directly get a map function $\pi = f(z)$ from training set $D = \{(z^i, a^i)\}$, where z stands for the observed state. This idea treats the problem as supervised learning problem.
- (ii) Learn a system model, namely, the return function and state transition probability distribution, from training set, and then derive a policy using MDP methods.
- (iii) Learn the pre- or postcondition of each action, and then obtain control sequence using programming methods.

For the obstacle avoidance problem, through learning the barrier-free path from teachers to produce a generalized path can be seen as a supervised regression problem. GMM is a probabilistic model which is suitable to model the natural variations in human and robot motions. Theoretically, GMM can model any probabilistic density functions (PDFs) by increasing the number of Gaussian components, so GMM has strong encoding capacity. Finally, movements recognizing, predicting, and generating can be integrated together within the same GMM and GMR encoding strategy. So it is very favorable for us to use GMM encoding our prior demonstration information and use GMR as the solution to the regression problem mentioned above. Details of GMM theory are described below.

3. Gaussian Mixture Model

For a probability density estimation problem, if, in addition to the known observation variables, there may exist some

unobservable latent variables, but we know the distribution of the latent variable and the conditional distribution of the observable variable given the latent variable, then we can use mixture models to estimate the parameters of the distribution, so as to get the approximate probability distribution of the known data. Before introducing GMM, we deviate to talk about Gaussian distribution.

3.1. Marginal and Conditional Distribution of a Multivariate Gaussian Distribution. Assume that we have a random variable

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad (1)$$

in which $x_1 \in \mathbb{R}^r$, $x_2 \in \mathbb{R}^s$, $x \in \mathbb{R}^{r+s}$. $x \sim \mathcal{N}(\mu, \Sigma)$, where

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \quad (2)$$

$$\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}.$$

And $\mu_1 \in \mathbb{R}^r$, $\mu_2 \in \mathbb{R}^s$, $\mu \in \mathbb{R}^{r+s}$, $\Sigma_{11} \in \mathbb{R}^{r \times r}$, and $\Sigma_{12} \in \mathbb{R}^{r \times s}$. Noting that the covariance matrix is symmetric, so we have $\Sigma_{12} = \Sigma_{21}^T$.

Because the marginal distribution of a Gaussian is also a Gaussian distribution, it is easy to get that $E[x_1] = \mu_1$, $\text{cov}(x_1) = \Sigma_{11}$, $E[x_2] = \mu_2$, and $\text{cov}(x_2) = \Sigma_{22}$, and we have $x_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$ and $x_2 \sim \mathcal{N}(\mu_2, \Sigma_{22})$. As a step forward, we can prove $x_1 | x_2 \sim \mathcal{N}(\mu_{1|2}, \Sigma_{1|2})$, where

$$\mu_{1|2} = \mu_1 + \Sigma_{12} \Sigma_{22}^{-1} (x_2 - \mu_2), \quad (3)$$

$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12} \Sigma_{22}^{-1} \Sigma_{21}.$$

3.2. Formulation of GMM. Assuming that we have a training set $D = \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ containing m samples, every sample $x^{(i)}$ is a vector. We are trying to find a density function to describe the data with the least error, which is equal to maximizing the probability of the data. In order to represent this probability, we need to make some assumptions. First we assume that the latent variable $z^{(i)} \sim \text{Multinomial}(\phi)$, in which $\phi_j \geq 0$, $\sum_{j=1}^k \phi_j = 1$; besides we suppose $x^{(i)} | z^{(i)} = j \sim \mathcal{N}(\mu_j, \Sigma_j)$. So we can model the data in our hands by

FIGURE 2: System architecture of obstacle avoidance learning.

adjusting parameters (ϕ, μ, Σ) . The likelihood function can be written as

$$\begin{aligned}
 l(\phi, \mu, \Sigma) &= \sum_{i=1}^m \log p(x^{(i)}; \phi, \mu, \Sigma) \\
 &= \sum_{i=1}^m \log \sum_{z^{(i)}=1}^k p(x^{(i)} | z^{(i)}; \mu, \Sigma) p(z^{(i)}; \phi).
 \end{aligned} \quad (4)$$

Maximize (4), and then we can find the right model parameters. To realize this purpose, typically we find the partial derivatives for each of the parameters and set them equal to zero, but actually we find that we cannot get a closed-loop solution for this problem using the above method. The main reason is that z is unknown; if z is known, the model degrades to a Gaussian discriminant model; following the previously mentioned steps, we can obtain a closed-loop solution. We will introduce a kind of commonly used iterative algorithm to solve the problem in (4).

3.3. EM for GMM. Expectation Maximization (EM) is an iterative algorithm, mainly including two steps, which are called E-step and M-step. The main idea is to find a tight lower bound function of the objective function and then maximize the lower bound function so that the objective function can be maximized indirectly. The reason for introducing a lower bound function is that finding the extreme value of the lower bound function is much easier compared to the original function. For GMM, the E-step tries to guess the value of $z^{(i)}$, equivalent to finding out which components the current sample belongs to, and then the model parameters are updated based on our conjecture. The algorithm can be summarized as follows.

E-step: for each i and j , compute the poster-probability of $z^{(i)}$:

$$w_j^i := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma). \quad (5)$$

M-step: modify our model parameters:

$$\begin{aligned}
 \phi_j &:= \frac{1}{m} \sum_{i=1}^m w_j^i, \\
 \mu_j &:= \frac{\sum_{i=1}^m w_j^i x^{(i)}}{\sum_{i=1}^m w_j^i}, \\
 \Sigma_j &:= \frac{\sum_{i=1}^m w_j^i (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^m w_j^i}.
 \end{aligned} \quad (6)$$

Note that the value of w_j^i can be computed using the following equation:

$$\begin{aligned}
 w_j^i &:= p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma) \\
 &= \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma) p(z^{(i)} = j; \phi)}{\sum_{l=1}^k p(x^{(i)} | z^{(i)} = l; \mu, \Sigma) p(z^{(i)} = l; \phi)},
 \end{aligned} \quad (7)$$

iterating between (6) and (7) until convergence, which means that the changes of model parameters are less than a threshold ϵ or the value of likelihood function changes in a very narrow range.

4. The Architecture of Obstacle Avoidance Learning

Figure 2 shows the architecture of obstacle avoidance learning and the information flow between different modules. The whole system contains three modules: data acquisition and preprocessing, constraints extraction and trajectory retrieval, and optimal control strategy derivation. Mileage information is collected by built-in sensors, such as odometers, and distance information is acquired by Kinect. Through noise filtering, invalid data reduction, and sequence alignment for the original data, we get the input for the learning machine. Then, the input signal is encoded by GMM and a general

FIGURE 3: Useful data collected by sensors. Line in blue shows the demonstrated trajectory. Sectors with a fan shape in shallow yellow color represent the visual field of the Kinect. $\{x, \omega, d^{ro}, d^{rg}\}$ represent location, orientation, and distance information of the robot, respectively.

trajectory is generated by GMR. The general trajectory satisfies all the constraints for the task. Eventually, a controller is designed to follow the expected input trajectory and output the control commands to the robot. Subsequent sections will describe each part in detail.

4.1. Data Acquisition. The experimental data is acquired from a Turtlebot robot. Turtlebot is a widely used research platform which is compatible with ROS, integrating a variety of sensors, such as odometer and extensible vision sensors. We can develop many kinds of applications by using Turtlebot, such as mapping and navigation. In this paper, we will implement our obstacle avoidance algorithm on the bases of ROS development kit for Turtlebot. The teaching process can be achieved by three ways: (i) using a remote joystick; (ii) with the aid of Turtlebot development kit for the navigation algorithms, simulating some trajectories as training samples; and (iii) by means of the following apps implemented on Turtlebot, the teacher can guide robot to avoid obstacles and record sensor data as samples. In this paper, we adopt the second method, which means that we create simulated environment in software and then make robot execute multiple obstacle avoidance tasks in the simulated environment using built-in navigation algorithms. During this process, linear velocity and angular velocity are obtained by built-in sensors like odometer. Using Kinect we can get the distance between the robot and obstacles, as well as distance from the robot to the target.

We assume that the important sensing information comes from: (i) states of the robot, defined as linear velocity and angular velocity; (ii) the robot's absolute location; (iii) distance between the robot and obstacles, noting that the Kinect returns many pieces of distance information representing the relative position of the obstacle in a specific range, and we choose the shortest distance as our reference value; and (iv) distance between the robot and the target, which can be computed indirectly. As illustrated in Figure 3, we use

$(x, \omega, d^{ro}, d^{rg})$ to represent the sensor information mentioned above, respectively. Here the location information and velocity information are redundant. We think it is necessary to take both kinds of information into consideration for the reason that location information is very important and direct for an obstacle avoidance task. In addition, choosing variables relating to the task as many as possible is generally very necessary in the initial stage of a task.

In the experiment, demonstration is conducted n times. In each demonstration, T data points are recorded, so the dataset contains $N = n \times T$ points in total. Each sample point is a collection of variables, represented as $D = \{x, \omega, d\}$, in which x stands for the location of robot, ω stands for the current orientation, and d stands for distance. Each variable contains a one-dimensional temporal value and $(d - 1)$ -dimensional spatial values; that is, $x = \{x_s, x_t\}$, $\omega = \{\omega_s, \omega_t\}$, $d^{ro} = \{d_s^{ro}, d_t^{ro}\}$, and $d^{rg} = \{d_s^{rg}, d_t^{rg}\}$. $x_s \in \mathbb{R}^{N \times 2}$ represent location, $\omega \in \mathbb{R}$ represent orientation, $d_s^{ro} \in \mathbb{R}^{N \times 2}$ represent distance between robot and obstacles, and $d_s^{rg} \in \mathbb{R}^{N \times 2}$ represent distance between robot and goal state. Supposing coordinates of the goal are $o_{s,i}$, then we have

$$d_s^{rg} = x_{s,i,j} - o_{s,i}, \quad \forall i \in \{1, \dots, n\}, \quad \forall j \in \{1, \dots, T\}, \quad (8)$$

where i indicates the trajectory number and j indicates time in a trajectory. We can rewrite the above equation for velocities:

$$\dot{d}_s^{rg} = \dot{x}_{s,i,j} - \dot{o}_{s,i}. \quad (9)$$

4.2. Data Preprocessing. Because of noise and bad demonstration quality, there exist serious mismatch and invalid data in the original data. As shown in Figure 4, there is no change at the beginning and end parts of these curves, so this part of data is useless. To reduce computation cost, it is a wise choice to abandon these data. In addition, each trajectory has different length; we should resample them to a uniform length so that we can handle them easier in the subsequent steps. So before inputting the data to our algorithms, preprocessing steps including noise filter, useless data reduction, and resampling are conducted. Figure 4 displays differences between the original data and the processed data.

4.3. Trajectory Encoding and Retrieval. Considering a data-point in training set $\xi_j = \{\xi_{t,j}, \xi_{s,j}\}_{j=1}^N \in D$, ξ_j is a D -dimensional vector. We encode the data with GMM, supposing we have K Gaussian components. The probability items in (4) can be represented as

$$\begin{aligned} p(k) &= \pi_k, \\ p(\xi_j | k) &= \mathcal{N}(\xi_j; \mu_k, \Sigma_k) \\ &= \frac{1}{\sqrt{(2\pi)^D |\Sigma_k|}} e^{-(1/2)((\xi_j - \mu_k)^T \Sigma_k^{-1} (\xi_j - \mu_k))}, \end{aligned} \quad (10)$$

in which $\{\pi_k, \mu_k, \Sigma_k\}$ are model parameters, representing prior-probability, mean value, and variance, respectively.

FIGURE 4: Comparison of original data and processed data. (a) shows the original data. Lines in red represent cutting position. (b) displays valid data after processing.

To solve these parameters, the EM algorithm introduced in the previous section can be used. When EM algorithm converges to a given value, we get the right $\{\pi_k, \mu_k, \Sigma_k\}$; that is, we encode the trajectory data successfully.

In order to get a general barrier-free trajectory, which is given an input, which can be temporal signals or other kinds of sensor data, then a barrier-free path is generated according to the given input, a natural idea is to divide the variable ξ into two parts $(\xi_t^{\mathcal{J}}, \xi_t^{\mathcal{O}})$, which represents input and output, respectively, and then trajectory retrieval problem can be formulized as follows: given $\xi \sim \sum_{i=1}^K \pi_i \mathcal{N}(\mu_i, \Sigma_i)$ and $\xi_t^{\mathcal{J}}$, solve the density probability distribution for $\xi_t^{\mathcal{O}}$. Solution to this problem is called Gaussian mixture regression. Regarding $\xi_t^{\mathcal{J}}$ as query points, $\xi_t^{\mathcal{O}}$ can be estimated using regression methods. In this paper, we use superscripts $\mathcal{J}$ and $\mathcal{O}$ standing for the input and output part for each variable. With this notation, a block decomposition of the data-point ξ_t , vectors μ_k , and matrices Σ_k can be written as

$$\xi_t = \begin{bmatrix} \xi_t^{\mathcal{J}} \\ \xi_t^{\mathcal{O}} \end{bmatrix},$$

$$\mu_k = \begin{bmatrix} \mu_k^{\mathcal{J}} \\ \mu_k^{\mathcal{O}} \end{bmatrix},$$

$$\Sigma_k = \begin{bmatrix} \Sigma_k^{\mathcal{J}} & \Sigma_k^{\mathcal{J}\mathcal{O}} \\ \Sigma_k^{\mathcal{O}\mathcal{J}} & \Sigma_k^{\mathcal{O}} \end{bmatrix}.$$

(11)

Based on (3), for a time step t , the mean vector and covariance matrix for the k th component are

$$\hat{\mu}_k^{\mathcal{O}}(\xi_t^{\mathcal{J}}) = \mu_k^{\mathcal{O}} + \Sigma_k^{\mathcal{O}\mathcal{J}} \Sigma_k^{\mathcal{J}\mathcal{J}}^{-1} (\xi_t^{\mathcal{J}} - \mu_k^{\mathcal{J}}),$$

$$\hat{\Sigma}_k^{\mathcal{O}} = \Sigma_k^{\mathcal{O}} - \Sigma_k^{\mathcal{O}\mathcal{J}} \Sigma_k^{\mathcal{J}\mathcal{J}}^{-1} \Sigma_k^{\mathcal{J}\mathcal{O}}.$$

(12)

Taking K components into consideration, the mean vector and covariance matrix for distribution $\xi_t^{\mathcal{O}}$ given $\xi_t^{\mathcal{J}}$ are

$$P(\xi_t^{\mathcal{O}} | \xi_t^{\mathcal{J}}) \sim \sum_{k=1}^K h_k(\xi_t^{\mathcal{J}}) \mathcal{N}(\hat{\mu}_k^{\mathcal{O}}(\xi_t^{\mathcal{J}}), \hat{\Sigma}_k^{\mathcal{O}}), \quad (13)$$

where

$$h_k(\xi_t^{\mathcal{J}}) = \frac{\pi_i \mathcal{N}(\xi_t^{\mathcal{J}} | \mu_i^{\mathcal{J}}, \Sigma_i^{\mathcal{J}})}{\sum_{k=1}^K \pi_k \mathcal{N}(\xi_t^{\mathcal{J}} | \mu_k^{\mathcal{J}}, \Sigma_k^{\mathcal{J}})}. \quad (14)$$

Equation (13) represents a multimodal distribution, which can be approximated by the following Gaussian distribution:

$$\begin{aligned} P(\xi_t^{\mathcal{O}} | \xi_t^{\mathcal{J}}) &= \mathcal{N}(\hat{\mu}_t^{\mathcal{O}}, \hat{\Sigma}_t^{\mathcal{O}}), \text{ with} \\ \mu_t^{\mathcal{O}} &= \sum_{i=1}^K h_i(\xi_t^{\mathcal{J}}) \mu_i^{\mathcal{O}}(\xi_t^{\mathcal{J}}) \\ \hat{\Sigma}_t^{\mathcal{O}} &= \sum_{i=1}^K h_i(\xi_t^{\mathcal{J}}) (\hat{\Sigma}_i^{\mathcal{O}} + \hat{\mu}_i^{\mathcal{O}}(\xi_t^{\mathcal{J}}) \hat{\mu}_i^{\mathcal{O}}(\xi_t^{\mathcal{J}})^T) \\ &\quad - \hat{\mu}_t^{\mathcal{O}} \hat{\mu}_t^{\mathcal{O}T}. \end{aligned} \quad (15)$$

Substituting $\xi_t^{\mathcal{J}}$ into $\xi_t^{\mathcal{O}}$ using equations described above, a general trajectory is obtained. The trajectory not only can avoid obstacles but also can be executed by robots due to its good smooth property.

4.4. Finding an Optimum Control Policy. Through the previous section, we can obtain a general trajectory, and this is a planning problem. How to combine planning with control? In other words, how to find a set of control inputs that can make the robot follow the general trajectory accurately? For the time-based GMR, $\xi_t^{\mathcal{O}}$ is equal to ξ_s , and s represents spatial location information. Using $\hat{\xi}_s$ represents the desired trajectory and ξ_s represents the actual reached location. In order to evaluate the tracking performance, a simple idea is using the weighted Euclidean distance:

$$\sum_{i=1}^{D-1} \omega_i (\xi_{s,i} - \hat{\xi}_{s,i})^2 = (\xi_s - \hat{\xi}_s)^T W (\xi_s - \hat{\xi}_s). \quad (16)$$

Naturally, the above equation can be applied to multiple variables. For this paper, we have four variables. Let $\{\hat{\xi}^x, \hat{\xi}^\omega, \hat{\xi}^{d^{ro}}, \hat{\xi}^{d^{rg}}\}$ represent the desired trajectory; the actual tracking performance P can be expressed as

$$\begin{aligned} P &= (\xi_s^x - C_x)^T W^x (\xi_s^x - C_x) \\ &\quad + (\xi_s^\omega - C_\omega)^T W^\omega (\xi_s^\omega - C_\omega) \\ &\quad + (\xi_s^{d^{ro}} - C_{d^{ro}})^T W^{d^{ro}} (\xi_s^{d^{ro}} - C_{d^{ro}}) \\ &\quad + (\xi_s^{d^{rg}} - C_{d^{rg}})^T W^{d^{rg}} (\xi_s^{d^{rg}} - C_{d^{rg}}), \end{aligned} \quad (17)$$

where

$$\begin{aligned} C_{*,i,j} &= \hat{\xi}_{s,i,j}^* - \xi_{s,i,j-1}^*, \\ \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, T\}, \forall * \in \{x, \omega, d^{ro}, d^{rg}\}. \end{aligned} \quad (18)$$

According to performance indicator P , finding a desired control policy is equivalent to maximizing P . Since P is a quadratic function, an analytical solution can be easily obtained by gradient method. But noting that there exist constraints between variables, as shown in (9), maximizing (17) turns into a constrained optimization problem. Usually, this kind of problem can be solved by constructing a Lagrange function, defined as follows:

$$L(\xi_s^x, \xi_s^\omega, \xi_s^{d^{ro}}, \xi_s^{d^{rg}}, \lambda) = P + \lambda^T (\xi_s^x - \xi_s^{d^{rg}}), \quad (19)$$

where λ is Lagrange multiplier. Letting $\nabla L = 0$, solving partial derivatives for $\xi_s^x, \xi_s^{d^{rg}}$, we find

$$\begin{aligned} 2W^x (\xi_s^x - C_x) + \lambda &= 0, \\ 2W^{d^{rg}} (\xi_s^{d^{rg}} - C_{d^{rg}}) - \lambda &= 0. \end{aligned} \quad (20)$$

Adding (20), we find

$$2W^{d^{rg}} (\xi_s^{d^{rg}} - C_{d^{rg}}) + 2W^x (\xi_s^x - C_x) = 0. \quad (21)$$

Substituting $\xi_s^x = -\xi_s^{d^{rg}}$ into the above equation, we find

$$\xi_s^x = (W^x - W^{d^{rg}})^{-1} (W^x C_x + W^{d^{rg}} C_{d^{rg}}). \quad (22)$$

Weight matrices in (22) can be computed by solving the inverse matrix of the current covariance matrix; namely,

$$W = (\Sigma_t^{\mathcal{O}})^{-1}. \quad (23)$$

5. Simulation Experiments

Experiments are implemented by using a Turtlebot robot. In addition to the robot's own hardware system, a Kinect is integrated to obtain distance information. The size of the map used in the simulation is 7.5×5 m. Two different maps are created to verify the learning effects. For the first map, the obstacle locates in the middle of the map, and a small gap lies in the lower side, and the robot can reach the target place through the gap. The second map is much similar to the first one, except that there exist two obstacles in the map. To get the destination, the robot must avoid every obstacle in the map. Figure 5 shows the shape of the map. Figures 5(a) and 5(b) represent the 2D map, and Figures 5(c) and 5(d) show a 3D simulation environment, which corresponds to the 2D maps. The color maps in maps (a) and (b) stand for cost cloud, which provides reference for path planning. The warm color, such as yellow and red, stands for a lower cost. To the contrary, the cool color tune stands for a higher cost. A path planner always tends to choose a trajectory with the least cost. The cost value is calculated by built-in functions in ROS package. For the convenience of description, we use task 1 and task 2 to represent the two different obstacle avoidance tasks in maps (a) and (b). It is worth mentioning that we select relatively simple environment for both tasks for the reason that

FIGURE 5: Two different maps used in our simulation. (a) and (b) are actual maps generated by software. (c) and (d) are simulation environment corresponding to maps above them.

FIGURE 6: Demonstrated trajectories in two situations. (a) represents sampled data in task 1. (b) represents sampled data in task 2.

we put more emphasis on the mechanism behind the obstacle avoidance, which means we can learn skills from simple environment, but these skills can be reused in complex situations.

Figure 6 shows the demonstrated trajectories in two different situations. Coordinates of goal states are set to $(6, 3)$ and $(6, 2)$ for task 1 and task 2, respectively. The start points for task 1 are randomly generated by sampling from uniform distributions between 0.5 and 2.5 for X -coordinates and 2

to 4 for Y -coordinates. The maximum range is 2 m, which can guarantee a good generalization performance for obstacle avoidance. For task 2, the start points are generated by sampling from uniform distributions from 0.5 to 2 and 2 to 3.5.

5.1. Experiments for Trajectory Encoding and Retrieval. Figure 7 shows the encoding and retrieval results for task 1. The encoded variables are location information x and distance

FIGURE 7: Trajectory encoding and retrieval using GMM and GMR for task 1. (a) represents model encoding with GMM. Ellipse in dark blue represents projections of Gaussian components. (b) represents general trajectory generated by GMR. Shades in red color represent generation region.

information d^{rg} , and both variables are 2-dimensional vectors. Number of Gaussian components is set to 6. It is worth noting that the optimal number of components K in a model may not be known beforehand so far. A common method consists of estimating multiple models with increasing number of components and selecting an optimum based on some model selection criterion, such as Cross Validation, Bayesian Information Criterion (BIC), or Minimum Description Length (MDL). In our paper, we decide K by using Cross Validation criterion. As the figures show, variance for starting position is large, corresponding

to a big retrieval error. It indicates that there are no strict requirements for starting position. Variance for end point is pretty small, which indicates that the robot can get the goal state accurately. Thus, they show that GMM can successfully extract useful constraints for characterization task. GMR can retrieve barrier-free trajectory by using information encoded by GMM, as shown in the second column of Figure 5. Figures in the second row show encoding and retrieval results for variable d^{rg} . As can be seen from the figures, d^{rg} shares similar distribution properties with x . This is because the two variables have a specific relationship between each other, so

FIGURE 8: Encoding and retrieval for orientation and distance information in task 1, sharing the same meaning with charts in Figure 7.

do mean value and covariance, which verify the validity of the method indirectly.

Figure 8 shows encoding results for orientation and distance information. Figures in the last two rows display encoding for d^{rg} in X and Y direction. As we see, coordinate values in X direction monotone decrease to 0. This is because the robot keeps moving from left to right in the map, while coordinates in Y direction keep going up at the beginning and then drop to zero. This is because the robot encounters obstacles in the middle moment; in order to avoid obstacles,

the robot has to go away from the goal state temporarily. Pay attention to the final distance value which is close to (0,0), which means the robot reaches the target successfully.

Figures 9 and 10 display encoding and retrieval results for task 2. Compared with task 1, task 2 is more complicated considering the law behind the data. In order to encode these laws, more Gaussian components are required. In our experiments K is set to 10 by using the method described in task 1. As illustrated in these figures, no matter how complex the environment is, this method still has very satisfactory effect.

FIGURE 9: Trajectory encoding and retrieval using GMM and GMR for variables x and d in task 2.

5.2. Experiments for Finding Control Strategy. Experiments in this part are mainly conducted to verify the derived control policy. For a Turtlebot robot, the control input is linear velocity ξ_s^x and angular velocity ξ_s^θ . Assuming we have a desired trajectory (the trajectory can be given arbitrarily; in our experiments, in order to find a reference, we choose the second trajectory in our samples), using theories derived in previous sections, we can compute ξ_s^x and ξ_s^θ . Iterating on every time, we can get a control sequence. Comparing the calculated value with the sensor recorded value, we can verify the validity of the derived results.

Figure 12 compares the calculated linear velocity and angular velocity with the desired value. Linear velocity is divided into velocity in X direction and Y direction. As the figure in the second row shows, expected velocity in Y direction remains close to 0. This is because the control inputs for Turtlebot are a linear velocity in the forward direction, that is, X direction, and an angular velocity. The line graph for velocity Y reveals that there is a minor fluctuation at the beginning and end stage, while the overall trend is consistent with the fact. Velocity x follows the expected value perfectly. Considering Figure 12(b), it can be seen that the calculated value remains consistent with the actual one. It is worth

noting that the length of the calculated sequence differs from the actual one, since the frequency of the readings from robot's sensors is lower than the frequency of commands. In this case, to compare the similarity of the two signals, sequence alignment is an indispensable step. In this paper, we use dynamic time warping (DTW) to achieve this purpose. Signals after alignments show a high correlation, which means the calculated values are correct.

Figure 11 shows the location information obtained by integrating velocity signals. The calculated control inputs allow the robot to follow the given trajectory accurately, except a minor fluctuation at the beginning time.

6. Conclusion

In this paper, based on the idea of the imitation learning, we propose a novel idea that the robot's obstacle avoidance skill can be learned by showing the robot how to avoid an obstacle. Under this consideration, task constraints extraction is a key issue, that is, how to make robots understand the task. In order to achieve this purpose, the Gaussian mixture model is used to encode the data. GMR is used to retrieve satisfactory

FIGURE 10: Encoding and retrieval for orientation and distance information in task 2.

FIGURE 11: Tracking performance using the derived policy.

FIGURE 12: Comparisons between the expected angular velocity and calculated value.

trajectories. Eventually, to obtain the control strategy, a multiobjective optimization problem is constructed. The control strategy is derived by solving this problem. Simulation experiments verified our algorithms.

For the future, improvement work mainly includes the following aspects: (i) in order to avoid obstacles online, online incremental GMM algorithm should be used; (ii) in this paper, a time-based trajectory is generated by GMR, that is, use of time signal as input. A more reasonable idea is using state information as input, so that the robot can change its decision according to the current environment, for example, using distance information as input and velocity as output; thus the control inputs can be directly generated; (iii) reinforcement learning algorithms can be integrated to strengthen robots' exploration ability, so that better generalization performance is achieved.

Competing Interests

The authors declare that the grant, scholarship, and/or funding do not lead to any conflict of interests. Additionally, the authors declare that there is no conflict of interests regarding the publication of this manuscript.

Acknowledgments

This work is supported by National Natural Science Foundation of China (Grant no. 51505470) and the Dr. Startup Fund in Liaoning province (20141152).

References

- [1] E. Masehian and D. Sedighizadeh, "Classic and heuristic approaches in robot motion planning—a chronological review,"

World Academy of Science, Engineering and Technology, vol. 23, pp. 101–106, 2007.

- [2] P. Raja and S. Pugazhenth, "Optimal path planning of mobile robots: a review," *International Journal of Physical Sciences*, vol. 7, no. 9, pp. 1314–1320, 2012.
- [3] J. C. Latombe, *Robot Motion Planning*, Springer, 2012.
- [4] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, 1989.
- [5] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.
- [6] H. Mei, Y. Tian, and L. Zu, "A hybrid ant colony optimization algorithm for path planning of robot in dynamic environment," *International Journal of Information Technology*, vol. 12, no. 3, pp. 78–88, 2006.
- [7] M. A. P. Garcia, O. Montiel, O. Castillo, R. Sepúlveda, and P. Melin, "Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation," *Applied Soft Computing*, vol. 9, no. 3, pp. 1102–1110, 2009.
- [8] C. Lopez-Franco, J. Zepeda, N. Arana-Daniel, and L. Lopez-Franco, "Obstacle avoidance using PSO," in *Proceedings of the 9th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE '12)*, pp. 1–6, Mexico City, Mexico, September 2012.
- [9] H.-Q. Min, J.-H. Zhu, and X.-J. Zheng, "Obstacle avoidance with multi-objective optimization by PSO in dynamic environment," in *Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC '05)*, vol. 5, pp. 2950–2956, IEEE, Guangzhou, China, August 2005.
- [10] S. Dutta, "Obstacle avoidance of mobile robot using PSO-based neuro fuzzy technique," *International Journal of Computer Science and Engineering*, vol. 2, no. 2, pp. 301–304, 2010.
- [11] W.-G. Han, S.-M. Baek, and T.-Y. Kuc, "Genetic algorithm based path planning and dynamic obstacle avoidance of mobile robots," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 3, pp. 2747–2751, Orlando, Fla, USA, October 1997.
- [12] J. Tu and S. X. Yang, "Genetic algorithm based path planning for a mobile robot," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '03)*, vol. 1, pp. 1221–1226, September 2003.
- [13] Y. Zhang and J. Wang, "Obstacle avoidance for kinematically redundant manipulators using a dual neural network," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 34, no. 1, pp. 752–759, 2004.
- [14] X. Wang, V. Yadav, and S. N. Balakrishnan, "Cooperative UAV formation flying with obstacle/collision avoidance," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 4, pp. 672–679, 2007.
- [15] J. Godjevac, "A learning procedure for a fuzzy system: application to obstacle avoidance," in *Proceedings of the International Symposium on Fuzzy Logic*, pp. 142–148, Zurich, Switzerland, 1995.
- [16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Boston, Mass, USA, 1998.
- [17] J. Michels, A. Saxena, and A. Y. Ng, "High speed obstacle avoidance using monocular vision and reinforcement learning," in *Proceedings of the 22nd International Conference on Machine Learning (ICML '05)*, pp. 593–600, ACM, Bonn, Germany, 2005.

- [18] C. Ye, N. H. C. Yung, and D. Wang, "A fuzzy controller with supervised learning assisted reinforcement learning algorithm for obstacle avoidance," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 33, no. 1, pp. 17–27, 2003.
- [19] B.-Q. Huang, G.-Y. Cao, and M. Guo, "Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance," in *Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC '05)*, vol. 1, pp. 85–89, August 2005.
- [20] C. F. Touzet, "Neural reinforcement learning for behaviour synthesis," *Robotics and Autonomous Systems*, vol. 22, no. 3-4, pp. 251–281, 1997.
- [21] W. D. Smart and L. P. Kaelbling, "Effective reinforcement learning for mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '02)*, vol. 4, pp. 3404–3410, Washington, DC, USA, May 2002.
- [22] K. Maček, I. Petrović, and N. Perić, "A reinforcement learning approach to obstacle avoidance of mobile robots," in *Proceedings of the 7th International Workshop on Advanced Motion Control (AMC '02)*, pp. 462–466, IEEE, Maribor, Slovenia, July 2002.
- [23] U. Muller, J. Ben, E. Cosatto et al., "Off-road obstacle avoidance through end-to-end learning," in *Advances in Neural Information Processing Systems*, pp. 739–746, 2005.
- [24] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.
- [25] C. L. Nehaniv and K. Dautenhahn, "Like me?-measures of correspondence and imitation," *Cybernetics & Systems*, vol. 32, no. 1-2, pp. 11–51, 2001.