Robotics and Autonomous Systems 84 (2016) 1–14

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Neural inverse reinforcement learning in autonomous navigation

Chen Xia *, Abdelkader El Kamel

Research Center CRIStAL, UMR CNRS 9189, École Centrale de Lille, 59651 Villeneuve d'Ascq, France

HIGHLIGHTS

- A novel model-free algorithm, Neural Inverse Reinforcement Learning, is introduced in performing autonomous navigation tasks.
- A nonlinear neural policy representation is proposed to establish the mapping between the state and action spaces.
- Computer-based expert demonstrations are supplemented to learning robots when human experts are not available in some extreme navigation tasks.
- A refinement operation based on maximum a posteriori is designed to pretreat the demonstrations from suboptimal experts.
- This method can easily deal with large state spaces and generalize unvisited states in demonstrations.

ARTICLE INFO

Article history:
Received 21 October 2015
Accepted 10 June 2016
Available online 23 June 2016

Keywords:
Inverse reinforcement learning
Learning from demonstration
Neural network
Autonomous navigation
Markov decision processes
Dynamic environments

ABSTRACT

Designing intelligent and robust autonomous navigation systems remains a great challenge in mobile robotics. Inverse reinforcement learning (IRL) offers an efficient learning technique from expert demonstrations to teach robots how to perform specific tasks without manually specifying the reward function. Most of existing IRL algorithms assume the expert policy to be optimal and deterministic, and are applied to experiments with relatively small-size state spaces. However, in autonomous navigation tasks, the state spaces are frequently large and demonstrations can hardly visit all the states. Meanwhile the expert policy may be non-optimal and stochastic. In this paper, we focus on IRL with large-scale and high-dimensional state spaces by introducing the neural network to generalize the expert's behaviors to unvisited regions of the state space and an explicit policy representation is easily expressed by neural network, even for the stochastic expert policy. An efficient and convenient algorithm, Neural Inverse Reinforcement Learning (NIRL), is proposed. Experimental results on simulated autonomous navigation tasks show that a mobile robot using our approach can successfully navigate to the target position without colliding with unpredicted obstacles, largely reduce the learning time, and has a good generalization performance on undemonstrated states. Hence prove the robot intelligence of autonomous navigation transplanted from limited demonstrations to completely unknown tasks.

1. Introduction

A rapid development of the robotic techniques has witnessed the advent of modern robots in large numbers. More and more robots are designed and produced to assist or replace our human beings to perform complicated control operations and planning tasks among plenty of domains. However, we are aware of the fact that designing reliable control systems for autonomous robot tasks is usually a complicated process, even for people specialized in programming robots. The number of uncertain situations which a robot may face and the wide spectrum of behaviors it may have to perform make the job of robot programming rather difficult. This sort of manually programming is generally an expensive as well as intense time-consuming process. Rather than pre-programming a robot for all the behaviors, it would be more useful if the robot could learn such tasks by themselves.

Recent researches has brought machine learning approaches into the community of robotics in order to improve the robot autonomous ability based on accumulated experiences. These artificial intelligent methods can be computationally less expensive than classical ones and can largely ease the burden on human designers.

In this paper, we are interested in designing intelligent mobile robots. The autonomous navigation ability is thus one of the fundamental skills, which requires a robot to traverse from some start location to a goal location in the meanwhile to avoid any potential collisions. Along the path, it must maintain knowledge

* Corresponding author.
E-mail addresses: chen.xia@centraliens-lille.org (C. Xia), abdelkader.elkamel@ec-lille.fr (A. El Kamel).

http://dx.doi.org/10.1016/j.robot.2016.06.003
0921-8890/

of its own position, observe the surrounding environment, plan actions to achieve its goal, and finally execute these actions. All decisions must be made in real time, and the robot may be faced with situations beyond the imagination of human designers. This makes it impractical to manually devise a policy for all the possible tasks that the robot may have to perform.

Machine learning techniques has been successfully applied to the state-of-the-art self-driving cars. Thrun et al. provided a comprehensive survey of Stanley, the winning robot of the 2005 DARPA Grand Challenge [1]. This robotic car was a milestone in the quest for self-driving cars. The pervasive use of machine learning techniques in outdoor obstacle detection and terrain mapping, both ahead and during the race, made Stanley robust and precise. However, since the race environment was static, Stanley was unable to navigate in traffic. Two years later, Junior, a robotic vehicle capable of navigating urban in-traffic environments autonomously was developed and won second place in the 2007 DARPA Urban Challenge [2]. The robot successfully demonstrated merging, intersection handling, parking lot navigation, lane changes and autonomous U-turns.

Reinforcement learning (RL) [3] is one of the machine learning methods that offers a powerful tool for constructing adaptive and intelligent systems. In the framework of RL, the learner is a decision-making agent that takes actions in an environment and receives an reinforcement signal for its actions in trying to accomplish a task. The signal, well known as reward (or penalty), evaluates an action's outcome, and the agent seeks to learn to select a sequence of actions, i.e. a policy, that maximize the total accumulated reward over time. Significant applications of reinforcement learning to enable the learning ability of autonomous systems can be found in [4–8]. Kober et al. also summarized the reinforcement learning techniques applied in robotics in [9].

The goal of a reinforcement learning agent is to collect as many rewards as possible, and an informative reward function becomes a fundamental assumption that a successful RL algorithm counts on. This type of the evaluation of robot behaviors always needs to be provided beforehand. However, in practice, defining the reward function can itself be a challenge because an informative reward function may be very hard to specify and exhaustive to tune for large and complex problems [10]. Inverse Reinforcement Learning (IRL) arose due to the curiosity of if a learning agent can discover the underlying rewards from a bunch of demonstrated examples of a desired behavior.

Rather than directly mimicking the expert with some supervised learning approach, IRL consists in learning a reward function under which the policy demonstrated by the expert is optimal. IRL is originally introduced in [11], where the authors addressed three learning problems: IRL in finite state spaces, IRL in infinite state spaces and IRL from sampled trajectories. In practice, it is easier to get samples from an expert. However, the authors also noted that the IRL problem is ill-posed. In fact, there exists a series of reward functions, including constant functions, that may lead to the same optimal policy. Abbeel and Ng then introduced a new indirect learning approach, named apprenticeship learning [12], where the learning is less concerned about the actual reward function, and the objective is to recover a policy that is close to the demonstrated behavior. It is assumed that the reward is a sum of weighted state features, and finds a reward function to match the demonstrator's feature expectations. This method may not explicitly recover the expert's reward function, but still output a policy that attains the performance close to that of the expert. This method is then implemented in helicopter control [13].

The maximum margin planning (MMP) algorithm [14] uses similar ideas, a linearized-features reward, where the learner attempts to find a policy that make the provided demonstrations look better than other policies by a margin, and minimizes a cost function between observed and predicted actions by a subgradient descent. Ratliff et al. extends the maximum margin planning and developed the LEARCH algorithm [15], and applied it to outdoor autonomous navigation. The idea of MMP also inspired the structured classification based inverse reinforcement learning (SCIRL) [16,17], where the authors use only sampled trajectories to reduce IRL to a structured classification problem and do not need to solve the direct RL process that many existing IRL algorithms require. Similarly, the dynamic policy programming was adopted in [18] in order to estimate the reward and the state value function without solving the MDP.

The Bayesian inverse reinforcement learning approaches [19,20] use probability distribution to tackle with the ill-posed problem. They assume that the demonstrator samples state–action sequences from a prior distribution over possible reward functions, and calculates a posterior on the reward function using Bayesian inference.

Similar to Bayesian IRL, the maximum entropy algorithm [21] use an MDP model for calculating a probability distribution on the state-action pairs. Maximum entropy IRL focuses on the distribution over trajectories rather than pure actions. Later on, based on the maximum entropy framework, the relative entropy inverse reinforcement learning algorithm using policy iteration is proposed in [22]. It indirectly employs knowledge of the environment and minimizes the relative entropy between the empirical distribution of the trajectories under a baseline policy and the distribution of the trajectories under a policy that matches the reward features of the demonstrations. A stochastic gradient descent is used to minimize the relative entropy.

Qiao and Beling proposed a Gaussian processes model and use preference graphs to represent observations of decision trajectories [23]. Levine et al. present a probabilistic algorithm for nonlinear inverse reinforcement learning and they use Gaussian process model to learn the reward as a nonlinear function [24].

Our research focuses on improving robot learning ability in autonomous navigation tasks via inverse reinforcement learning. An autonomous navigation task is a typical large-scale state-space problem. The demonstrators can only cover a small subset of the state spaces, and thus solving the generalization of state space is a key issue. Most inverse reinforcement learning algorithms use a linear feature-based state representations instead of directly using states in order to generalize on undemonstrated states and do not give an explicit policy representation in large-scale spaces.

In this paper, we present an efficient and convenient learning algorithm, neural inverse reinforcement learning (NIRL), and apply it to autonomous robot navigation tasks. We represent the states using a linear combination of state and action features and adopt an artificial neural network to generalize the expert's actions to unvisited regions of the state space. By this means, we propose an explicit nonlinear policy representation. The maximum margin method is applied to learn the reward function. The IRL algorithms in the literature generally assume that a model of the transition is known, which is unrealistic in an unpredictable navigation problem. Our method, on the contrary, is model-free. Experiments are conducted on simulated autonomous robot navigation, and the results show that a mobile robot using our approach can successfully accomplish an autonomous navigation task without colliding with unpredicted obstacles. NIRL largely reduces the learning time, and has a good generalization performance on undemonstrated states. Therefore, NIRL is proved to be a reliable and robust learning algorithm for endowing the mobile robots the autonomy and the intelligence.

The rest of the paper is organized as follows. The next section describes the fundamental background that we use in this paper. Section 3 describes the autonomous mobile robot model. The proposed nonlinear neural policy representation is given in

Section 4, and Section 5 presents the complete neural inverse reinforcement learning algorithm. Section 6 describes two kind of demonstrations, optimal and sub-optimal, in the context of autonomous navigation. In Section 6, experiments are conducted to show the simulation results of the proposed method. Section 7 draws the final conclusions.

2. Background

This section covers the theoretical background that is necessary for understanding the remaining of this paper.

2.1. Markov decision processes

Inverse reinforcement learning is formulated within the framework of Markov decision processes (MDPs) without a reward function, denoted by MDP\R. An MDP describes a sequential decision problem in which an agent must choose the sequence of actions that maximizes some reward-based optimization criterion. Formally, a finite-state, infinite horizon MDP is defined as a tuple $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma\}$ where:

- $\mathcal{S} = \{s_1, \ldots, s_n\}$ is a finite set of *n states* that represents the dynamic environment,
- $\mathcal{A} = \{a_1, \ldots, a_p\}$ is a set of *p actions* that could be executed by an agent,
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longmapsto [0, 1]$ is a *transition probability function*, or *transition model*, where $\mathcal{T}(s, a, s')$ stands for the state transition probability upon applying action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ leading to state in state $s' \in \mathcal{S}$, i.e. $\mathcal{T}(s, a, s') = P(s' \mid s, a)$,
- $r : \mathcal{S} \times \mathcal{A} \longmapsto \mathbb{R}$ is a *reward function* with absolute value bounded by $R_{\max}$; $r(s, a)$ denotes the immediate reward incurred when action $a \in \mathcal{A}$ is executed in state $s \in \mathcal{S}$,
- $\gamma \in [0, 1)$ is a *discount factor*.

A *policy* is a mapping $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, where $\pi(s, a)$ is the probability pf choosing action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$.

For a fixed policy, the *value of a state* is defined by

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s\right], \tag{1}$$

where $(s_t, a_t)_{t \geq 0}$ is the sequence of random state–action pairs generated by executing the policy π. The function $V^{\pi} : \mathcal{S} \to \mathbb{R}$ is called the *value function* under the policy π.

The associated *action–value function* $Q^{\pi}(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ under the policy π is defined by

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a\right]. \tag{2}$$

The purpose of the agent is to find an *optimal policy* π^* that maximizes the expected total discounted reward over all states. The *optimal value function* is defined by $V^*(s) = \sup_{\pi} V^{\pi}(s)$, while the *optimal action–value function* is defined by $Q^*(s, a) = \sup_{\pi} Q^{\pi}(s, a)$.

A policy π is optimal for a MDP $\mathcal{M}$ if and only if for all $s \in \mathcal{S}$:

$$\pi(s) \in \arg\max_{a \in \mathcal{A}} Q^{\pi}(s, a). \tag{3}$$

V^* and Q^* satisfy the following *Bellman Equations* for all $s \in \mathcal{S}, a \in \mathcal{A}$:

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a), \tag{4}$$

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V^*(s'). \tag{5}$$

2.2. Inverse reinforcement learning

Inverse reinforcement learning addresses the problem of recovering the unknown reward function for a given Markov decision problem for which the optimal policy matches the demonstrations. The agent observes N_T trajectories: $\mathcal{D}_E = \{\tau^{(i)}\}_{1 \leq i \leq N_T}$, and each trajectory $\tau^{(i)}$ consists of L_i state–action pairs: $\tau^{(i)} = \{(s_t^{(i)}, a_t^{(i)})\}_{1 \leq t \leq L_i}$. Therefore, the expert demonstrations is composed of M state–action pairs:

$$\mathcal{D}_E = \{(s_1, a_1), (s_2, a_2), \ldots, (s_M, a_M)\},$$

where $M = \sum_{i=1}^{N_T} L_i$. IRL is aimed at finding a reward function r whose corresponding optimal policy π^* matches the actions taken in $\mathcal{D}_E$. We assume that the reward function is a linear combination of K feature vectors f_k with weights θ_k:

$$\forall (s, a) \in \mathcal{S} \times \mathcal{A} : \quad r(s, a) = \theta^{\top} f(s, a) = \sum_{k=1}^{K} \theta_k f_k(s, a). \tag{6}$$

The features indirectly represent the perceived state of environment. Then, Eq. (2) can be rewritten as follows:

$$\begin{aligned} Q^{\pi}(s, a) &= \theta^{\top} \mu^{\pi}(s, a), \\ \text{with } \mu^{\pi}(s, a) &= \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t f(s_t, a_t) \mid s_0 = s, a_0 = a\right]. \end{aligned} \tag{7}$$

The term μ^{π} is called the feature expectations of the policy π, and they completely determine the expected sum of discounted rewards for acting according to that policy [12]. The expectation of one feature f_i under the policy π can thus be expressed by

$$\mu_i^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t f_i(s_t, a_t) \mid s_0 = s, a_0 = a\right]. \tag{8}$$

If two policies share the same feature expectations, then they will have the same value function whatever reward $R = \theta^{\top} f$:

$$\mu^{\pi_1} = \mu^{\pi_2} \Longrightarrow \theta^{\top} \mu^{\pi_1} = \theta^{\top} \mu^{\pi_2} \Longrightarrow Q^{\pi_1} = Q^{\pi_2}. \tag{9}$$

Therefore, inverse reinforcement learning can be transformed to minimizing a distance between the feature expectations of the expert μ_E and those of the learning agent μ^{π}.

3. Model of autonomous mobile robots

The mobile robot model used in this paper is built according to the mobile robot system Robotino® from Festo Didactic,[1] as shown in Fig. 1. It is based on an omnidirectional drive assembly, which enables the system to roam freely. The Robotino® is equipped with a total of nine infrared sensors arranged around its base at an angle of 40° to each other, and each distance sensor reads out a voltage level whose value depends on the distance to a reflective object.

Therefore, our mobile robot model is assumed to have three wheels, one in front and two in back. The robot is equipped with nine distance sensors to determine the distances of objects within its surroundings. The sensors are arranged at an angle of 40° to each other, and each sensor can cover a scope of detection of 40°, as shown in Fig. 2(a). $(S_i)_{\{1 \leq i \leq 9\}}$ are the nine sensors and $(R_i)_{\{1 \leq i \leq 9\}}$ representing the corresponding scopes of detection.

The robot navigation environment consists of its target and the obstacles, as shown in Fig. 2(b). The initial and target positions are predefined to the robot, and the robot will try to reach the target in a collision-free path in spite of the presence of static or moving obstacles.

d_{r-t} is the distance between the robot and the target, and d_{r-o} is the distance between the robot and the obstacle.

[1] http://www.festo-didactic.com/int-en/services/robotino/.

4. Nonlinear neural policy representations

A policy connects every state with a corresponding action. If we are dealing with a large-scale state space, an explicit policy representation would be hard to describe, and most of previous work omit the policy representation. With a perfect policy representation, the learning process could be much more efficient.

An artificial neural network (NN) can approximate any functions in any accuracy and has a good generalization performance. Thus, we propose an explicit policy representation by incorporating the neural network framework, and this representation can ease the learning by IRL.

4.1. State and action spaces

In order to represent a state of environment, we first denote a numerical danger level of 0–7 for each sensor reading. The bigger the level is, the more possible the robot collides with an obstacle. A level of 0 means no obstacle detected in this sensor region and 7 means a collision. Only seven sensors excluding $S5$ and $S6$ are used in the state representation. The reason why we eliminate them is that the experimental results showed a better performance if not considering $S5$ and $S6$ as state features. Thus, the information of sensors at time instant t are stored in a 7-element row vector $D_t = [d_1, d_2, d_3, d_4, d_7, d_8, d_9]^\top$, where d_i is the ith sensor degree of danger.

Besides sensor readings, a state should include two more parameters. One is the target region, $V_t \in \{1, 2, \ldots, 9\}$, represented by robot sensor regions in Fig. 2(a). The other one is an indicator $I_t \in \{0, 1\}$ that determines if the robot has detected the target. $I_t = 1$ means the target shows up in the robot sensors.

When a sensor detects an obstacle, a robot should distinguish if this obstacle is an object that can be bypassed or a wall that cannot be bypassed. Hence, we introduce a vector $U_t \in \{0, 1\}^{7\times 1}$ to indicate if a wall is detected ($U = 1$) or not ($U = 0$) for the seven sensors in D_t.

Therefore, a state can be defined by four groups of features, totally 16 features, and is expressed by the function $\phi : \mathcal{S} \to \mathbb{R}^{16\times 1}$:

$$\phi(s_t) = \begin{bmatrix} D_t \\ V_t \\ I_t \\ U_t \end{bmatrix}. \tag{10}$$

Since the robot needs to deal with different environments, and the robot position coordinates have no direct relation with the target and the obstacles in different environments, it is not necessary to include them as part of the state features. The robot localization can be fixed by many methods and is not the focus of this paper, so we assume the robot know its position relative to its start position.

The action space is defined by six robot moving actions:

$$\mathcal{A} = \{u_1, u_2, u_3, u_4, u_5, u_6\}.$$

Among them, five are basic moving actions: move forward (u_1), turn left at 30° (u_2), turn left at 60° (u_3), turn right at 30° (u_4), turn right at 60° (u_5) and one emergency action: move backward (u_6). The six actions are based on the robot orientation. A turn at 90° is not defined because a sharp turn will bring danger to a real vehicle. Moving backward (u_6) is regarded as an emergency action that can be executed when no path is available in front of the robot and whatever turns cannot avoid obstacles.

4.2. Neural policy representation

In general, given a set of n actions, one state at time instant t has a k-feature representation $\phi(s_t) = \{\phi_i\}_{1\le i\le k}$. We can then extend this state feature by a state–action feature representation $f(s_t, a_t)$ at time instant t. It is a row vector of size $k \times n$:

$$f(s_t, a_t) = [\nu_1 \quad \nu_2 \quad \cdots \quad \nu_n] \tag{11}$$

where $\nu_i \in \mathbb{R}^{1\times k}$ is a row vector with the definition:

$$\nu_i = \begin{cases} \phi(s_t)^\top, & \text{if } a_t = u_i \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

The state–action features $f(s_t, a_t)$ would be used to express the reward function.

Then, we denote a function $\mathcal{G} : \mathcal{S} \to \mathbb{R}^{n\times 1}$ a three-layer neural network to link each state s_t with all its corresponding action values $Q(s_t, u)$, i.e.,

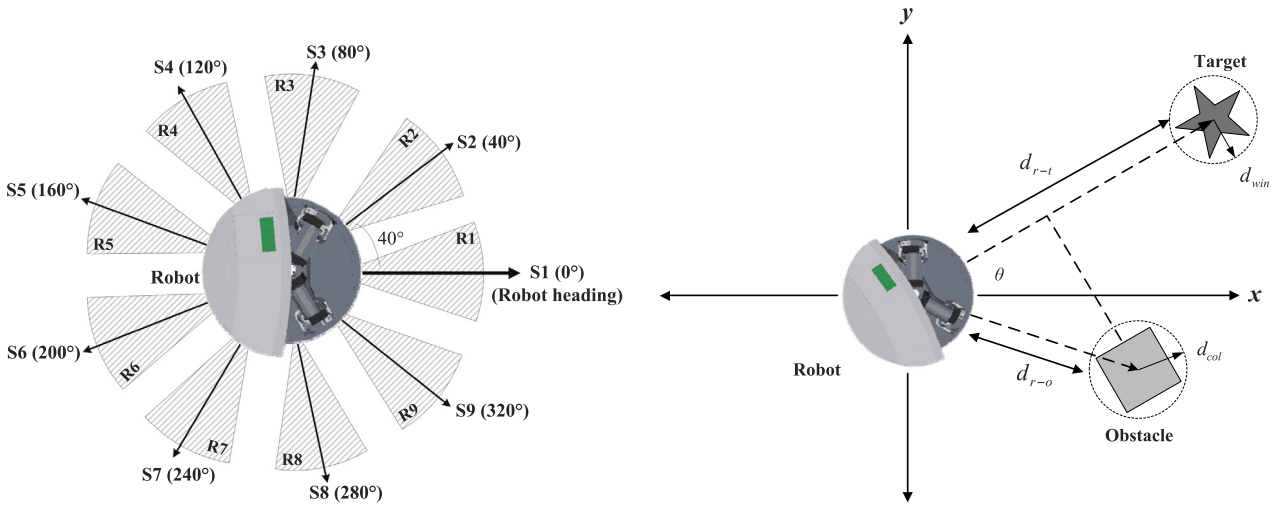
$$\mathcal{G}(s_t) = \begin{bmatrix} Q(s_t, u_1) \\ Q(s_t, u_2) \\ \vdots \\ Q(s_t, u_n) \end{bmatrix}, \tag{13}$$

where $n = |\mathcal{A}|$. The architecture of the neural policy representation is shown in Fig. 3.

Generally, the state features $\phi(s)$ is sent to the input layer and the network outputs a vector of action values. Thus, the network has k input units, L_h hidden units and n output units. From now on, we consider the bias units as parts of the input and hidden layers, and they are set to 1. Hence, the input of the network is now $x = \{\phi_i\}_{0\le i\le k}$ where $\phi_0 = 1$ is the bias unit, and the same goes for the hidden layer. The weight $W^{(1)}$ is used to connect the input layer and the hidden layer, and similarly, the weight $W^{(2)}$ links the hidden layer and the output layer:

$$\begin{aligned} W^{(1)} &= \begin{bmatrix} \omega^{(1)}_{1,0} & \cdots & \omega^{(1)}_{1,k} \\ \vdots & \ddots & \vdots \\ \omega^{(1)}_{L_h,0} & \cdots & \omega^{(1)}_{L_h,k} \end{bmatrix} \in \mathbb{R}^{L_h\times(k+1)}, \\ W^{(2)} &= \begin{bmatrix} \omega^{(2)}_{1,0} & \cdots & \omega^{(2)}_{1,L_h} \\ \vdots & \ddots & \vdots \\ \omega^{(2)}_{n,0} & \cdots & \omega^{(2)}_{n,L_h} \end{bmatrix} \in \mathbb{R}^{n\times(L_h+1)}. \end{aligned} \tag{14}$$

Fig. 1. Robotino®: a mobile robot system for education and research.



(a) The mobile robot model with nine sensors and their corresponding sensing areas.

(b) The mobile robot environment with a target and an obstacle.

Fig. 2. The autonomous mobile robot model.

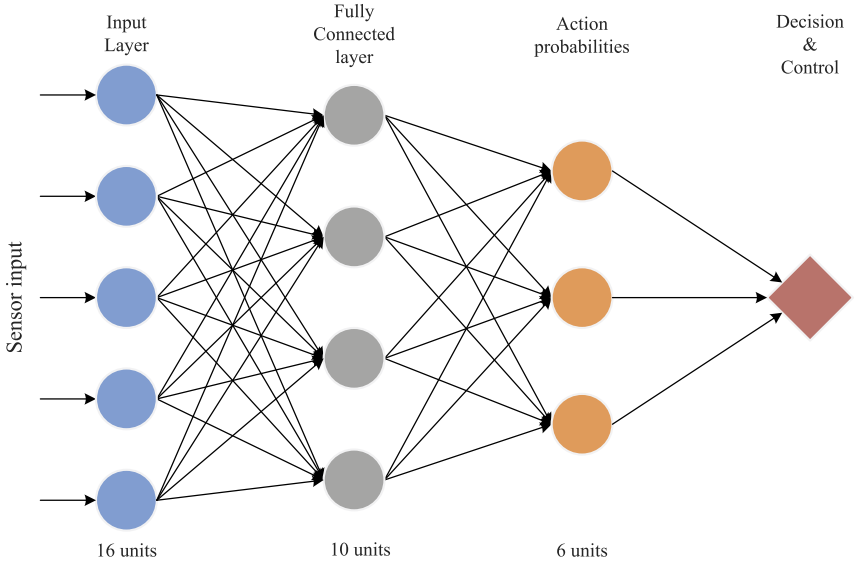


Fig. 3. The nonlinear neural policy representation in autonomous navigation.

Keeping the weights $w^{(1)}$ and $w^{(2)}$ unchanged, we can calculate $g(s_t)$ using the feedforward neural network (FFNN) with the parameter $\Omega = \{W^{(1)}, W^{(2)}\}$:

$$\begin{aligned} z &= w^{(1)} \cdot x = w^{(1)} \cdot \begin{bmatrix} 1 \\ \phi_t \end{bmatrix}, \\ h &= \mathrm{sig}(z), \\ g_\omega(s_t) &= \mathrm{sig}\left(w^{(2)} \cdot \begin{bmatrix} 1 \\ h \end{bmatrix} \right) \end{aligned} \tag{15}$$

where $\phi_t = \phi(s_t) \in \mathbb{R}^{k\times 1}$, and $\mathrm{sig}(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function, applied in both hidden and output units.

We also denote

$$g_\omega(s_t, a_t) = [g_\omega(s_t)]_i = Q(s_t, u_i), \quad \text{with } u_i = a_t.$$

4.3. Stochasticity of policy

The stochastic policy is designed based on the neural representation, and is expressed by a (parametric) Boltzmann probability distribution $\pi_\Omega(s, a) = P(a|s; \Omega)$ that stochastically selects action a in state s according to NN weight Ω:

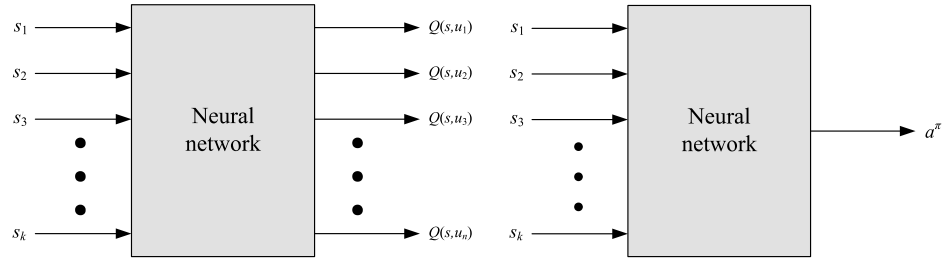
$$\pi_\Omega(s_t, a_t) = P(a_t \mid s_t; \Omega) = \frac{\exp(\eta g_\Omega(s_t, a_t))}{\sum_{u\in\mathcal{A}} \exp(\eta g_\Omega(s_t))}. \tag{16}$$

η is the Boltzmann parameter that controls the stochasticity of action selection. If η is low, all the choices have similar values, the policy outputs more stochastically. On the contrary, if η is high, the action Q-values differ and the action with the highest Q-value is preferred to be picked. Thus, $P(a_t|s_t) \propto \exp(\eta Q(s_t, a_t)) > 0$.

Once the policy stops updating, the robot may start navigation by taking greedy action selection that selects actions according to the following equation is optimal:

$$a_t^\pi = \pi_\Omega(s_t, a_t) = \arg\max_{u\in A} g_\Omega(s_t). \tag{17}$$

Since the neural network can approximate any function, our proposed method is greatly suitable for large state spaces. One may consider the neural policy representation as a black box which contains a nonlinear function between input features and output state–action values, see Fig. 4(a).



(a) Multiple outputs. (b) Single output.

Fig. 4. A black box structure of neural network.

Also, if we integrate the action selection strategy inside the black box, we may choose the selected action as the only output, see Fig. 4(b).

5. Neural inverse reinforcement learning

Inverse reinforcement learning algorithms normally require a model of environment and solve iteratively MDPs in order to optimize the reward function. They also assume the reward function to be linear in the state features, and update all states to obtain a new policy in solving MDPs. The NIRL that we proposed, however, is a model-free method, and with the adoption of the proposed nonlinear neural policy representation, NIRL only needs to update a subset of state–action space to achieve the generalization of unvisited regions, which largely reduces the learning time.

In our method, we do not assume the expert to be optimal. After obtaining the set of sub-optimal demonstration examples, we may pretreat the dataset via maximum a posteriori estimation (MAP) before proceeding them by NIRL.

5.1. Suboptimal demonstration refinement via maximum a posteriori estimation

When an expert demonstrates one specific task, it is not guaranteed that he is always making decisions optimally. For instance, the expert may take different actions for a same state. This can be interpreted as the expert policy is assumed to be stochastic, even the actions may be stochastic. For example, we desire to steer the car to the left at 30°, however, due to the slippery road, the actual steering angle turns out to be 60°. In these cases, we would like to pretreat the examples from the expert via maximum a posteriori estimation and finally refine them to be near-optimal.

Formally, given the expert examples $\mathcal{D}_E = \{(s_t, a_t)\}_{1\le t\le M}$, there exist some state–action pairs that the same state correspond to multiple actions. The MAP is expressed by

$$\hat{a}(s_t) = \arg\max_a P(s_t \mid a)P(a). \tag{18}$$

The likelihood $P(s_t \mid a)$ can be obtained by counting the frequency of (s_t, a) in $\mathcal{D}_E$. The prior $P(a)$ can be of any predefined form. If we do not know the prior, we can assume $P(a)$ is uniformly distributed. In this case, we have $P(s_t \mid a)P(a) \propto P(s_t \mid a)$, then $\hat{a}(s_t) = \arg\max_a P(s_t \mid a)$. A MAP estimation is reduced to a maximum likelihood (ML) estimation. Simply, for one particular state in all expert samples, the most likely action choice is the mode of all the actions corresponding to that state.

After the pretreatment of suboptimal examples, we can apply NIRL on them.

5.2. Model-free maximum margin planning

NIRL is designed to learn a reward function from the expert demonstrations in the form of multiple trajectories. Assume that the reward $r(s, a) = \theta^\top f(s, a)$, and given a set of demonstrations $\mathcal{D}_E = \{\tau_E^{(i)}\}_{1\le i\le N_T}$ with each trajectory $\tau_E^{(i)} = \{(s_t^{(i)}, a_t^{(i)})\}_{1\le t\le L_i}$, we denote the estimated value for the expert feature expectations μ_E by

$$\mu_E = \frac{1}{N_T}\sum_{i=1}^{N_T}\sum_{t=1}^{L_i}\gamma^{(t-1)} f(s_t^{(i)}, a_t^{(i)}). \tag{19}$$

The maximum margin planning method (MMP) [14] is applied to find a policy π that has feature expectations close to those of the expert, that is, $\|\mu^\pi - \mu_E\| < \epsilon$. It iteratively constructs a set of policies until a policy is found for which the distance between its value and the estimate of the expert's policy (using the expert's feature expectations) is smaller than a given threshold. If such a policy is not found, a new policy is generated as the optimal policy corresponding to a reward function for which the expert does better by a margin than any of the policies previously found by the algorithm.

We use the MMP in a model-free context, and we compare the values of the expert in every state in the recorded trajectories to the ones of learned policy in the same state. The total reward of the expert policy has to be bigger than the rewards of any other learned policy by a loss function $l(s, a) \ge 0$:

$$\begin{aligned} &\min_\theta \frac{1}{2}\|\theta\|_2^2 + C\sum_{i=1}^{N_T}\xi^{(i)} \\ &\text{s.t. } Q_\theta^{\pi_E}(s_t^{(i)}, a_t^{(i)}) + \xi^{(i)} \ge \max_\pi Q^\pi(s_t^{(i)}, a_t^{(i)}) + l(s_t^{(i)}, a_t^{(i)}), \quad \forall\pi \end{aligned} \tag{20}$$

where $Q^\pi(s_t^{(i)}, a_t^{(i)})$ is an estimated Q-value of the learning agent in the state $s_t^{(i)}$, and is calculated by using our neural policy representation in Section 4.2. $Q_\theta^{\pi_E}(s_t^{(i)}, a_t^{(i)})$ is the Q-value of the expert policy, calculated by Eq. (7) with the expert feature expectations in Eq. (19). The loss function $l(s, a)$ is defined to be 0 if the state–action pair of the expert can be found by the learned policy, otherwise $l(s, a) = 1$. A slack variable $\xi^{(i)}$ is given for each expert trajectory in order to allow constraint violations for a penalty, and it is supposed to be small. Hence the optimization problem is simplified by minimizing the objective function:

$$\begin{aligned} J(\theta) = &\sum_{i=1}^{N_T}\sum_{t=1}^{L_i}\max_\pi\left(Q^\pi(s_t^{(i)}, a_t^{(i)}) + l(s_t^{(i)}, a_t^{(i)})\right) \\ &- Q_\theta^{\pi_E}(s_t^{(i)}, a_t^{(i)}) + \frac{\lambda_1}{2}\|\theta\|_2^2, \end{aligned} \tag{21}$$

where $\lambda_1 \ge 0$ is a empirical constant to balance the penalties on constraint violations and the desire for a small weight vector. $J(\theta)$ can be minimized by a subgradient descent method proposed in [14,15], i.e., $\theta \leftarrow \theta - \sigma_1\frac{\partial J(\theta)}{\partial\theta}$, where $\sigma_1 \in [0, 1]$ is a step-size parameter. After obtaining θ, we can calculate the reward function $r_\theta(s, a)$ using Eq. (6).

5.3. Neural policy iteration

The nonlinear neural policy needs to be updated now. We execute the current learned policy π with the help of $r_\theta(s, a)$ in the context of RL. We first update the Q-values of the current policy Q^π by the on-policy SARSA algorithm. The update rule is

$$\hat{Q}^\pi(s_t, a_t) = Q^\pi(s_t, a_t) + \alpha[r_\theta(s_t, a_t) + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t)] \quad (22)$$

where $Q^\pi(s_t, a_t)$ is calculated by the neural policy representation, and $\hat{Q}^\pi(s_t, a_t)$ are the new (updated) Q-values in the same state and action. α is the learning rate, set between 0 and 1. Setting it to 0 means that the Q-values are never updated, hence nothing is learned. Setting a high value means that learning can occur quickly.

γ is the discount factor with a range of 0 and 1. If γ is close to 0, the robot will tend to consider immediate reward. On the contrary, if γ approaches 1, the robot will take more future reward into account.

Then, we update the nonlinear neural policy iteratively through the backpropagation algorithm. That is realized by minimizing the neural network error $J(\Omega)$ via updating the weights $W^{(1)}$ and $W^{(2)}$. The network's error is the difference between its output for a given input and a target value, what the network is expected to output. The inputs are the state features and the outputs are the state–action values. The Q-values updated by RL algorithms are treated as the target values. $J(\Omega)$ is the cross-entropy cost function and defined as follows:

$$J(\Omega) = -\frac{1}{N}\sum_{t=1}^{N}\Big[\hat{Q}^\pi(s_t, a_t) \circ \log Q^\pi(s_t, a_t) + (1 - \hat{Q}^\pi(s_t, a_t)) \circ \log(1 - Q^\pi(s_t, a_t))\Big] + \frac{\lambda_2}{2N}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(w_{j,i}^{(l)}\right)^2. \quad (23)$$

N is the total number of state–action pairs generated by the learned policy π. L is the total number of layers in network, and s_l is the number of units in layer l, excluding bias unit. The second term of $J(\Omega)$ is the regularization term aimed at avoiding overfitting or underfitting, and λ_2 is regularization parameter, optimized by using cross-validation of state–action pairs.

The gradient descent method is adopted to minimize the cost $J(\Omega)$ by

$$W^{(l)} \leftarrow W^{(l)} - \sigma_2 \frac{\partial J(\Omega)}{\partial W^{(l)}}, \quad \Omega = \{W^{(1)}, W^{(2)}, \ldots\}$$

where $\sigma_2 \in [0, 1]$ is a step-size parameter.

5.4. The algorithm for neural inverse reinforcement learning

Given a set of demonstrations, if the expert is suboptimal, we should first decide if a MAP pretreatment is needed. Then after the refinement, we calculate the expert's feature expectations using Eq. (19).

The NIRL algorithm iteratively repeat three major steps:

1. Estimate the feature expectations of current learned policy,
2. Find current reward function using MMP,
3. Solve the MDP and calculate current optimal policy with respect to the reward function.

The complete NIRL algorithm for finding an optimal policy is presented in Algorithm 1.

Algorithm 1 The NIRL algorithm

Input: the expert's feature expectations μ_E.

1: Randomly generate an initial neural policy $\pi^{(1)}$, represented by its weights $\Omega^{(1)} = \{W^{(1)^{(1)}}, W^{(2)^{(1)}}\}$.
2: Set $i = 1$.
3: **while** $i > 0$ **do**
4: Execute the current policy $\pi^{(i)}$ and generate a sequence of state–action pairs $\zeta^{(i)} = \{(s_t^{(i)}, a_t^{(i)})\}$, and compute $\mu^{(i)} = \mu(\Omega^{(i)})$.
5: Minimize the objective function $t^{(i)} = \min_\theta J(\theta)$ in Eq. (21) s.t. $\|\theta\|_2 \leq 1$, and let $\theta^{(i)}$ be the value that attains this minimum.
6: **if** $t^{(i)} \leq \epsilon$ **then**
7: Terminate.
8: **end if**
9: Compute the rewards $r(s_t^{(i)}, a_t^{(i)}) = (\theta^{(i)})^\top f(s_t^{(i)}, a_t^{(i)})$, and using the SARSA algorithm, update the Q-values of the sequence $\zeta^{(i)}$.
10: Apply the neural policy iteration to compute the current optimal neural weights $\Omega^{(i+1)} = \{W^{(1)^{(i+1)}}, W^{(2)^{(i+1)}}\}$, which represents the new policy $\pi^{(i+1)}$.
11: Set $i = i + 1$.
12: **end while**

Output: a series of NN weights $\Omega^{(1)}, \Omega^{(2)}, \ldots$.

In the algorithm, the expert feature expectations μ_E are represented by its estimate value $\hat{\mu}_E$, i.e., $\mu_E = \hat{\mu}_E$.

The outputs of NIRL are also a series of policies:

$$\{\pi^{(1)}, \pi^{(2)}, \pi^{(3)}, \pi^{(4)}, \pi^{(5)}, \ldots\}.$$

We use these policies to predict the state–action pairs in the demonstrations $\mathcal{D}_E$, and we choose the policy that achieves the highest prediction accuracy as the optimal policy of NIRL.

Once learned, the NIRL can recover the reward for the current state space, and can predict the reward for any unseen state space within the domain of the features.

6. Expert demonstrations

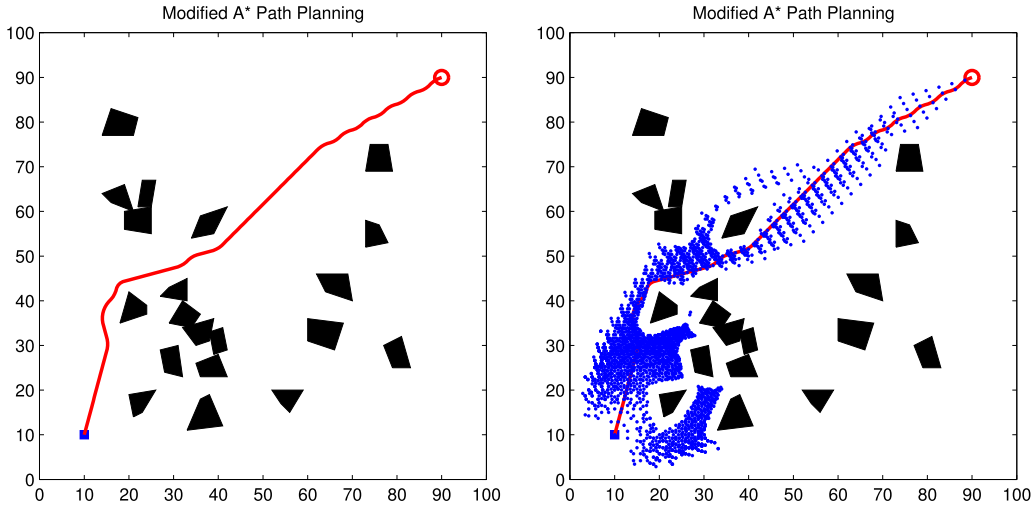
Two types of demonstrations are used in this paper. The first one is the human expert demonstrations. A human expert choose the best policy but may still not an optimal one. The second one is the computer expert demonstrations. A computer expert can always give optimal policies, and therefore a modified A* algorithm is applied to generate optimal policies. Another advantage of computer expert is that in some hazardous situations, human demonstrations cannot be available for the robot, so learning from computer experts bring less harm. All the demonstration examples can be gathered either by the modified A* algorithm, or by the human expert, and NIRL algorithm learns the reward function. Finally a MDP solver finds a policy mapping from input of world states to output of robot actions.

6.1. Computer-based expert demonstrations

The computer-based expert demonstrations are realized by the modified A* algorithm.

The modified A* algorithm has proven its robustness in robot navigation tasks, however this global path planning method is built upon a knowledge of the whole map which is unrealistic in real applications. The idea is how to transplant the optimal approach to future robot navigation tasks where only local information about the map is accessible.

A* is a well known best-first shortest path finding algorithm, and is highly used in robotics. It is used to find a path from a



(a) Modified A* planning. (b) Modified A* with all nodes in the closed list (blue dots).

Fig. 5. Modified A* algorithm result. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

given start node to a target node. It searches the whole map area and finds each possible path from the start node to reach the target node and then gives the shortest path. Unlike most A* algorithms use rectangle grids for the world representation, this paper proposes a modified A* based on robot actions. Since a robot can choose 6 possible actions, one node in modified A* have 6 potential neighbor nodes according to actions. Without restriction of grid representation, the planning becomes more practical.

This paper uses the classic representation of the A* algorithm as the evaluation function $F(x)$:

$$F(x) = G(x) + H(x). \tag{24}$$

$G(x)$ is the path cost function which calculates the actual total cost of the path to reach the current node from the start node. G cost is accumulated by the total distance traveled by all the past moving steps.

$H(x)$ is the heuristic function which estimates the cost of the path from the current node to the goal node. H cost is simply taken as the direct distance from the current position to the target regardless of obstacles.

Each node is represented by a structure array with the following fields: current position; heading; H value; G value; F value; parent position; action.

Take *start node* as example: current position is the robot starting coordinates; heading is the robot initial heading; H value is the distance to the target; G value is 0; F value is the sum of H and G values; it has no parent position and no action to take.

Only current position is recorded into the open list and the closed list. Other field values can be called on by searching the node with the corresponding current position.

The advantage of this modified A* algorithm is that the path finding is not restricted to a grid world. This method shows more flexibility and can be adapted to any real robot applications with any robot action spaces.

In Fig. 5, the start point and the target are, respectively, placed at (10, 10) (a solid blue rectangle) and (90, 90) (a hollow red circle). The optimal path is planned by the modified A* algorithm, and represented by the red path in Fig. 5(a). The blue dots alongside the red path in Fig. 5(b) showed us all the world states visited by our algorithm. Without dividing the environment into grids, the proposed algorithm has successfully computed an optimal path.

In Fig. 6, we compared our method (in red solid curve) and the conventional A* algorithm (in green dashed curve). For the same navigation environment, the conventional A* used 48.1936 s to plan a path of 124.3638 m, while our algorithm took 23.2288 s to give out a path of 120.2053 m. We could tell that this modified method produced a shorter and more optimal path, and specifically saved much planning time. In sum, our method is more practical. The expert's policy π_E corresponds to the optimal deterministic policy.

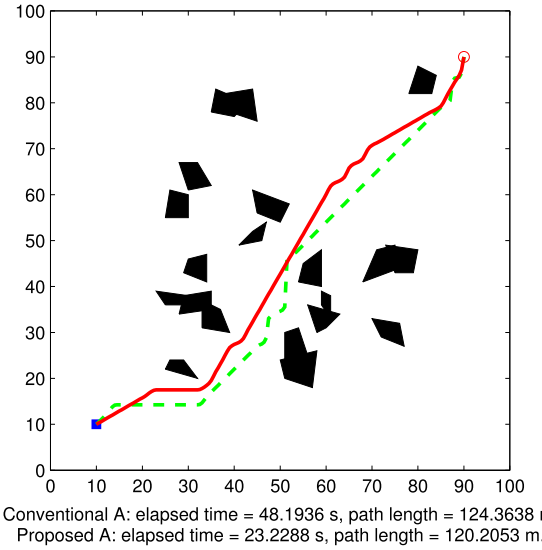
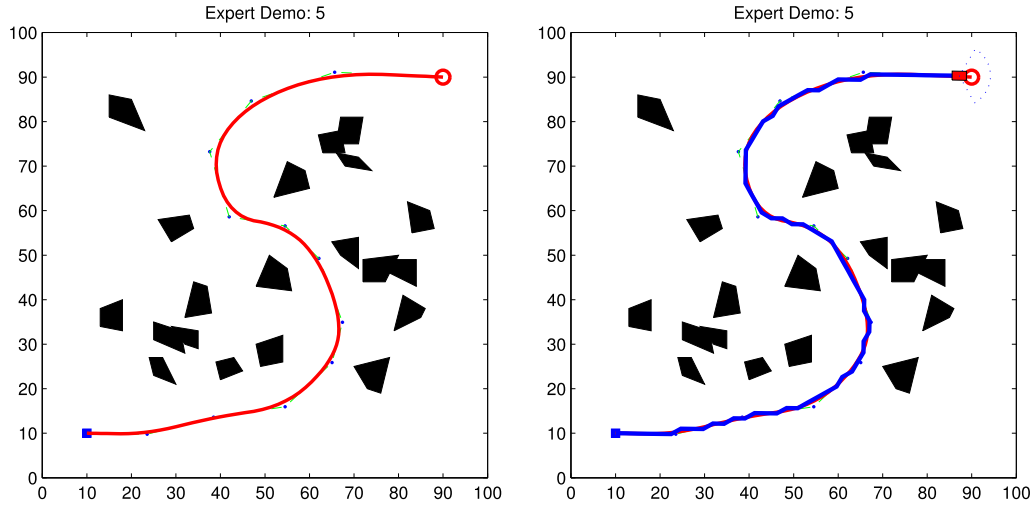


Fig. 6. Pathfinding results using modified A* (red solid) and conventional A* (green dashed) algorithms. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

6.2. Human demonstrations

A human expert can also demonstrate their navigating strategy by themselves. An expert can control the learning robot by teleoperation or directly show the robot how to navigate. We model the human demonstration in Fig. 7.

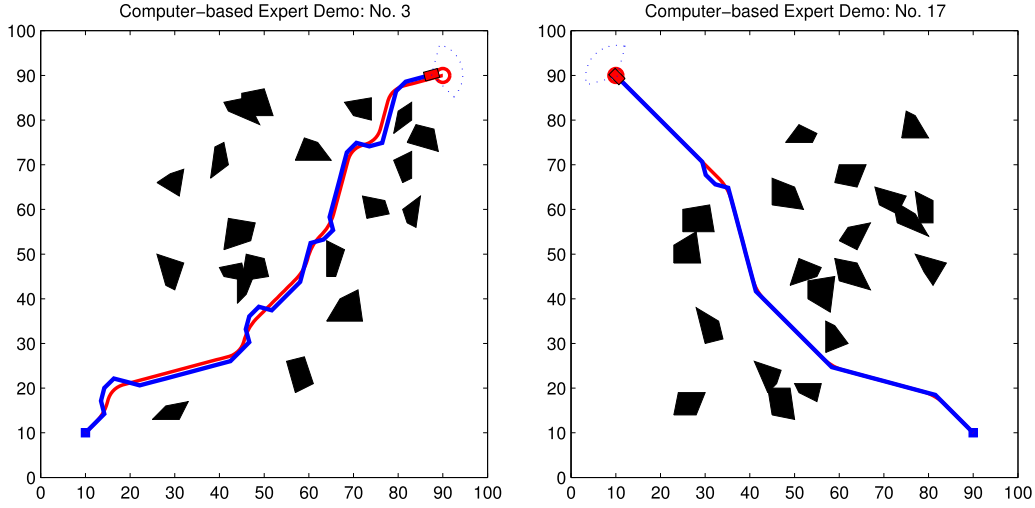
The human expert first gave some key points in the navigating map (represented by the tiny blue points in Fig. 7(a)), and then a smooth trajectory (in red) passing the key points is generated through interpolation techniques. The trajectory is regarded as the planned path for the robot. Afterwards, the robot follows the path to collect state–action pairs and leaves a blue trail in Fig. 7(b). Obviously, a human expert may not always deliver an optimal policy.



(a) A red path is given by a human expert.

(b) The robot follows the planned path to collect data and leaves a blue trail.

Fig. 7. A human expert demonstration. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) A trajectory with several stochastic actions.

(b) A trajectory with little stochastic actions.

Fig. 8. Two expert demonstrations with stochastic actions. The expert paths are shown in red lines, and the robot trajectories are shown in blue lines. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

7. Experimental results

The mobile robot is represented by a rectangle-shaped robot. It is equipped with 9 sensors to observe the surrounding environment, as shown in Fig. 2(a). The environment map has a size of 100 m $\times$ 100 m. The obstacles are randomly scattered in the environment and the robot has no prior knowledge of their numbers, sizes and positions. The initial position of the robot is placed at (10, 10) and the target position, a red circle in the map, is found at (90, 90). The velocity of the robot is fixed at 2 m/s. The mission of the robot is to start from the initial position and to find an optimal path to arrive at the target position without any collision with any obstacles. If no obstacles are detected, the robot is designed to move towards the target.

The neural network adopted in the experiments has three layers: 16 in the input layer, 10 in the hidden layer and 6 in the output layer. The inputs and outputs are presented above in Section 4.1. Some experiment parameters are selected as follows:

- Boundary distance: $d_{bou} = 10$ m,
- Warning distance: $d_{warn} = 5$ m,
- Learning rate of SARSA: $\alpha = 0.8$,
- Learning rate of BPNN: $\sigma = 0.4$,
- Discount factor: $\gamma = 0.65$.

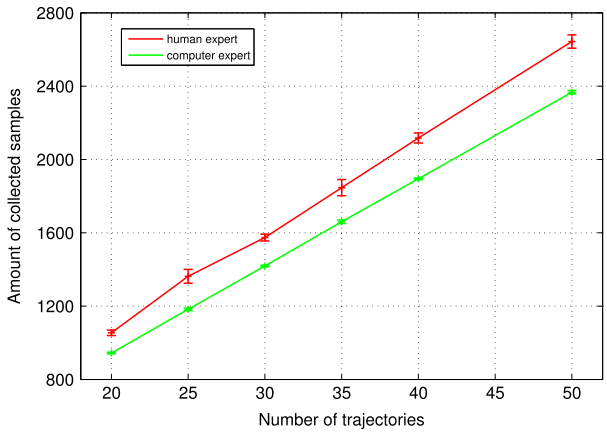
7.1. Comparison of two types of demonstrations

When robot follows the expert trajectories, the actions that the robot take may be stochastic, that is, a suggested action may lead to a different action according a probability $P(\hat{a} \mid a)$, see Fig. 8.

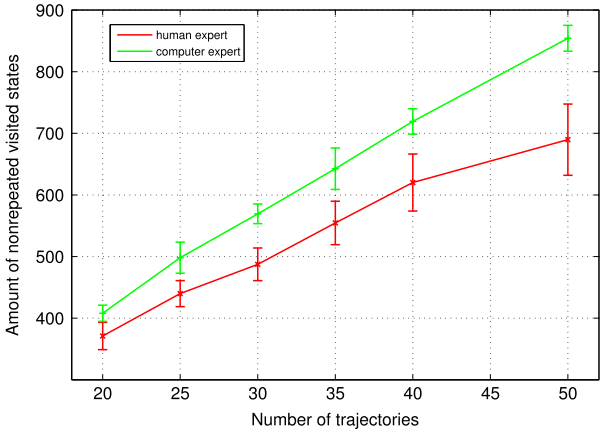
In Fig. 8(a), we can see a number of stochastic actions, while in Fig. 8(b), there exists only one stochastic action.

In order to show the relationship between number of demonstrations and collected samples, different numbers of demonstrations were tested: 20, 25, 30, 35, 40 and 50. Recall that one sample is one state–action pair. Each number is conducted 10 times. The average results are shown in Fig. 9(a). Averagely, one trajectory could contain 52–55 samples.

The human's policies π_E correspond to the sub-optimal stochastic policies. Due to the stochasticity, in the same states, there may exist many action possibilities. We concern how many different states have been visited during the demonstrations,



(a) The average amount of collected samples in different numbers of trajectories.



(b) The average amount of nonrepeated visited states in different numbers of trajectories.

Fig. 9. A comparison between human and computer-based experts.

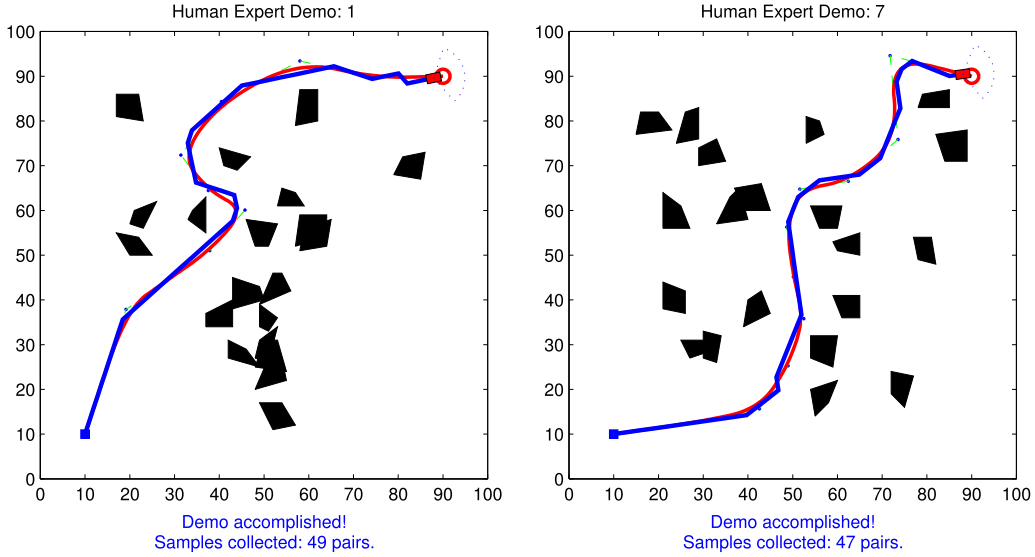


Fig. 10. Human expert demonstrations. The red paths are chosen by the human expert, and the blue paths are traveled by the robot to collect data. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

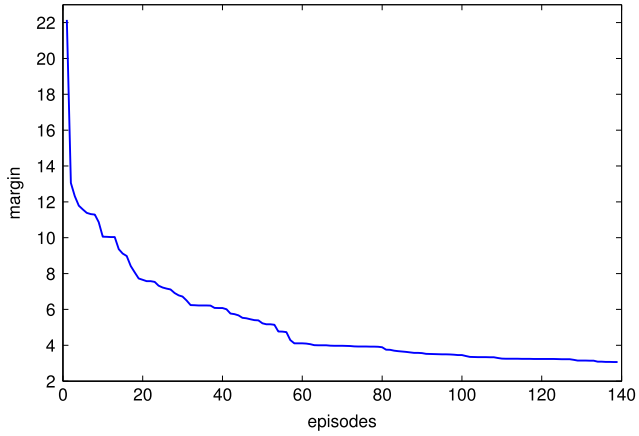


Fig. 11. Changes of margin during the learning episodes.

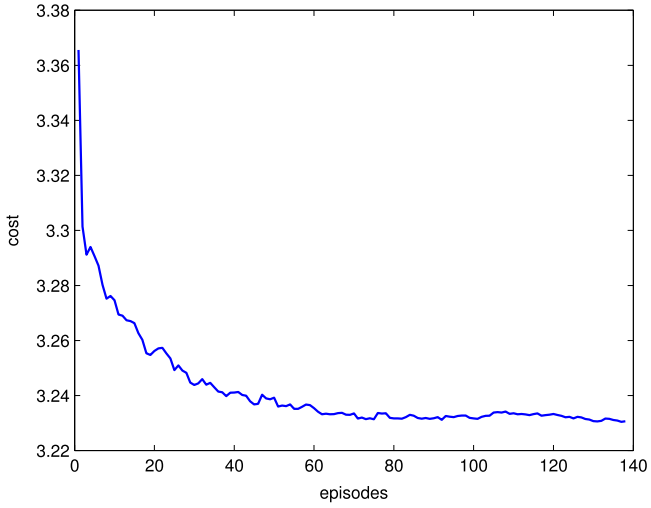


Fig. 12. Changes of NN cost during the learning episodes.

therefore, we gave out the relation between the amount of nonrepeated visited states and the number of demonstrations, as shown in Fig. 9(b).

Averagely, one trajectory could contain 13–19 nonrepeated visited states.

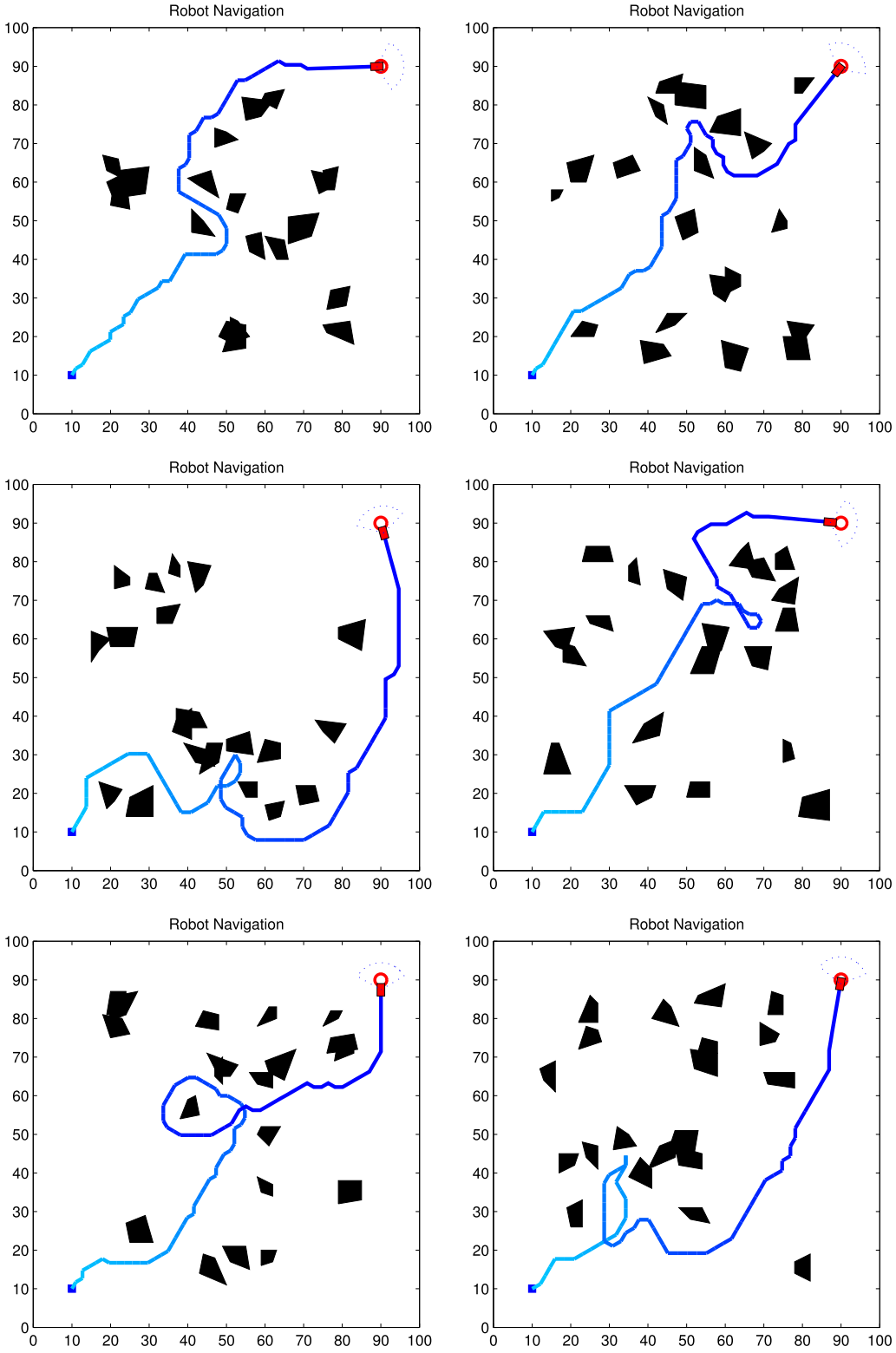


Fig. 13. Robot navigation results in different environments.

7.2. Neural inverse reinforcement learning

In this paper, 30 demonstrations are generated both by human expert and computer expert. All of them have different configurations of random obstacle positions. The advantage of different maps is that the robot can always face new challenge and help gather more state–action combination. Some of the human expert demonstrations are presented in Fig. 10.

It is observed that the computer-based demonstrations can always provide optimal paths, and this will greatly improve the robot learning performance compared to human teachers who cannot guarantee the quality of their demonstrations.

In each iteration of NIRL, an episode of 10 trajectories is generated using the current policy and all the sequences of state–action pairs are collected that are used for updating the policy. In each trajectory, the robot tries to navigate in different

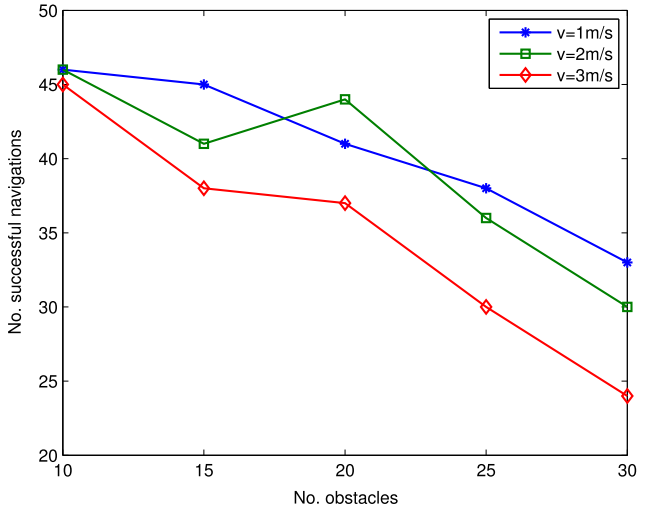


Fig. 14. Comparison of numbers of successful navigations in the environments of different numbers of obstacles.

environments in order to assure the diversity of state–action pairs. A new episode will be started in the following three situations:

- The robot finds a collision free path to the target;
- The robot collides with an obstacle or the map borders;
- The robot runs out the moving steps.

The max-margin method is applied to learn the reward function by minimizing the margin function $J(\theta)$ in Eq. (21).

Fig. 11 showed the change of the margin in MMP during the learning episodes. We can tell the margin was stably diminishing towards 3. Hence, the margin between a learned policy was gradually approaching the expert policy, and this is exactly the learning objective.

The nonlinear neural policy is trained for the state–action pairs collected under the current policy. The goal is to minimize the cost $J(\Omega)$ in Eq. (23) by iteratively updating the weights $W^{(1)}$ and $W^{(2)}$. The result is shown in Fig. 12.

In the first 20 episodes, the cost decreases fast and then becomes smaller stably. Finally after 140 episodes, the cost converges and stays around 3.23. Once the training terminates, $W^{(1)}$ and $W^{(2)}$ stop updating and the learning process from demonstration is accomplished. It is now the time to examine the robot learning achievement.

7.2.1. Robot navigation in new unknown environments

In the navigation process, the weights $W^{(1)}$ and $W^{(2)}$ have converged to their optimal values. The mobile robot has learned how to behave in front of obstacles in unexpected environments. The neural policy is adopted to choose the best fit action for robot behaviors. Some of the navigation results are shown in Fig. 13.

It is observed that the paths that the robot chose may not be the optimal ones due to a lack of complete knowledge of environment map, but they are still fully acceptable and perfect enough to meet our expectation, especially in such complicated and unstructured environments.

A successful navigation is one in which the robot succeeds in arriving at the target position. We tried 100 times of autonomous navigation by using various number of expert demonstrations. We compare the rate of success with both computer expert demonstrations (CE) and human expert demonstrations (HE) (see Table 1).

We observe that both types of demonstrations have proved to be reliable instructors to the robot, and the computer expert did slightly better. Also more demonstrations may not achieve a higher rate of success, because more demonstrations bring more stochastic in expressing the policies.

Table 1
Rate of success in different numbers of demos.

Number of demos	25	30	40	50
Rate of success of CE	90%	91%	89%	87%
Rate of success of HE	82%	85%	84%	84%

In order to test the robustness of the proposed algorithm, we gave the robot six completely new navigation missions using the same weights with three different robot velocities, namely 1, 2 and 3 m/s. 50 runs were conducted for each velocities. The results are shown in Fig. 14.

With the increase of obstacles presented in the surrounding environment, the robot becomes harder to navigate successfully. It is a normal situation that a bigger speed represents a need for quicker and more accurate reaction to the environment changes, especially in an completely unknown environment. Despite that, we can see that at least the robot could manage half of the navigation tasks and in most cases, the robot did pretty good.

Therefore, the above experiments have proven the feasibility and the stability of the proposed NIRL algorithm in different navigation environments and in different robot velocities.

7.3. Analysis of the weight changes

We analyzed the changes of neural policy weights $W^{(1)}$ and $W^{(2)}$ during the learning process using NIRL. The change in one episode is the difference between the weights before the learning and the ones after learning, i.e., $\|W^{(l)}_{\text{new}} - W^{(l)}\|_1$. The results are shown in Fig. 15(a) and (b). The X-axis represents the number of episode and the Y-axis is the weight change.

The changes of the weight $W^{(1)}$ fluctuated strong in the beginning episodes and then went quiet around 0 after 20 episodes. The overall trend of weight changes showed that the weight $W^{(1)}$ was turning stable as the learning process carried on and $W^{(1)}$ converged to its optimal value.

In the meantime, the changes of weight $W^{(2)}$ fluctuated more fiercely through all the learning process even though the changes went small after 60 episodes. That is because $W^{(2)}$ is directly related to the errors between the real outputs and the expected values in the neural network. Since the expected values are calculated in Eq. (22), and in each learning episode, the robot needs to deal with a different environment, there are always errors between the real outputs and the expected values. Therefore, it is reasonable that the weight $W^{(2)}$ is more difficult to converge to an optimal value. However, the overall trend still told us that $W^{(2)}$ was converging during the learning process, and the experiments showed that it did not affect the robot learning performance.

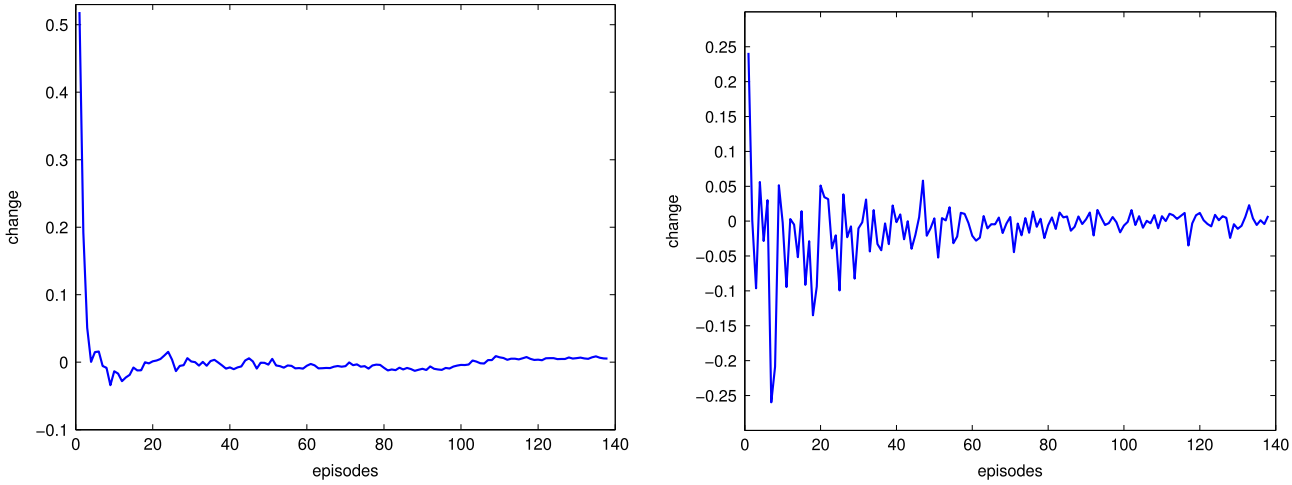
The changes of weights showed that NIRL converged to the optimal values after approximately 80 learning episodes, and the learned policy is close enough to the expert policy.

7.4. Autonomous navigation in dynamic environments

We tested the NIRL algorithm in dynamic environments. The results are shown in Fig. 16. In the environment, the black solid objects are static obstacles, and the purple rectangle ones are moving obstacles, whose moving directions are unpredictable.

There are 10 static obstacles and 10 moving obstacles. When the robot detected no obstacles around, it headed to the destination. From the experiment, we can see that at time 13, the robot met a static obstacle on its left and a moving obstacle in front of it, and the robot successfully avoided them and finally arrived at the destination.

Therefore, NIRL has proven its feasibility in the application of autonomous navigation in dynamic environments.



(a) Changes of $W^{(1)}$ in learning process.

(b) Changes of $W^{(2)}$ in learning process.

Fig. 15. Changes of neural policy weights during the learning process.

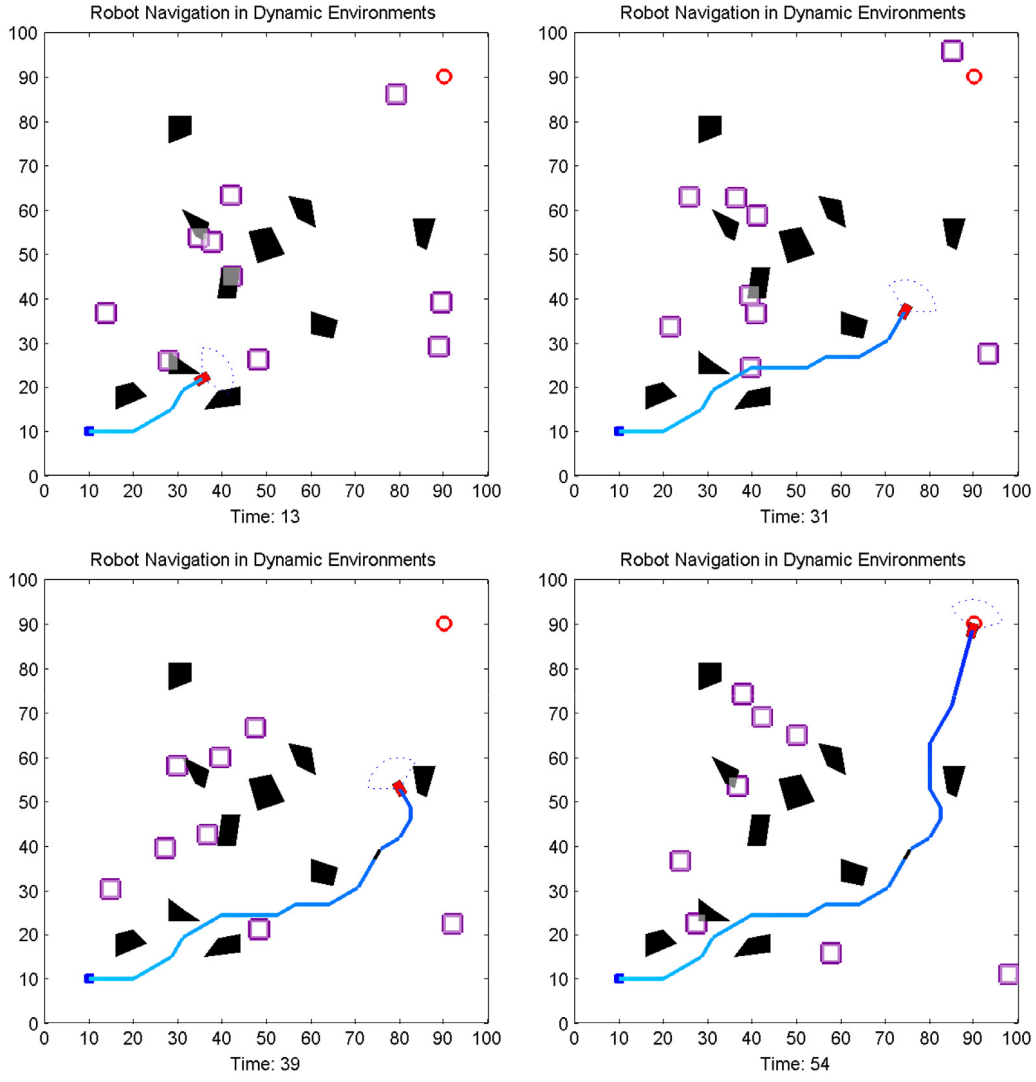


Fig. 16. Autonomous robot navigation in a dynamic environment using NIRL.

7.5. Discussions

We compare the NIRL algorithm with three major existing methods. The first one is the apprenticeship learning (AL) in [12], the baseline of the family of inverse reinforcement learning. The second one is the relative entropy inverse reinforcement learning (RE), proposed in [22], a popular IRL algorithm using the theory of entropy. The last one is the structured classification inverse

Table 2
Comparison of the rate of success versus different IRL methods.

	NIRL	AL	RE	SCIRL
Average rate (%)	83	66	78	79
Best rate (%)	94	82	87	88
Worst rate (%)	69	43	61	60

reinforcement learning (SCIRL), presented in [16], a recent work using max-margin method for a batch model-free learning algorithm. We applied these methods on the setting of autonomous navigation.

We fixed the robot velocity at 2 m/s and the number of obstacles at 20. We conducted 25 runs for each learning method, and one run is composed of sufficient training given expert trajectories and 100 tests of executing autonomous navigation tasks. For all the methods, we implemented the same computer-based expert, and we calculated the rate of success based on the 100 tests in one run. We compared the results from the runs having the best and the worst rates, and also averaged the rates from all 25 runs. The comparison is shown in Table 2.

From the results, we can see that NIRL achieved the best rate of success. RE and SCIRL performed almost equally and slight worse than our NIRL algorithm. AL, as a pioneer algorithm, did the worst.

Therefore, the NIRL algorithm had a better performance in autonomous navigation tasks over the other three methods.

8. Conclusion

This paper presents an intelligent robot learning method, that the robot tries to understand an expert's behaviors in executing one specific task by means of learning the underlying rewards from the expert demonstrations. This method is under the framework of inverse reinforcement learning. We developed our method, the neural inverse reinforcement learning algorithm. We first proposed a nonlinear neural policy representation by incorporating an artificial neural network. This explicit policy representation made the learning algorithm easy to implement. NIRL is also a method that does not assume the expert to be optimal. We proposed to apply maximum a posteriori to pretreat the suboptimal samples and later use the refined samples onto NIRL. Then, we adopted the maximum margin method to learn the reward through minimizing the maximum margin between the expert policy and the learned policy. Finally, we updated the nonlinear neural policy using the learned reward function.

The experiment results show the feasibility and the robustness of the proposed NIRL algorithm, and the robot can complete autonomous navigation tasks safely in an unpredicted dynamic environment. The robot not only can imitate what the expert behaves but also understand why the expert behaves in that way through learning the reward function. Future work will be concentrated on extending NIRL to the continuous state and action spaces.

References

[1] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, et al., Stanley: The robot that won the DARPA grand challenge, J. Field Robot. 23 (9) (2006) 661–692.

[2] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, et al., Junior: The stanford entry in the urban challenge, J. Field Robot. 25 (9) (2008) 569–597.

[3] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT press, 1998.

[4] M.A.K. Jaradat, M. Al-Rousan, L. Quadan, Reinforcement based mobile robot navigation in dynamic environment, Robot. Comput.-Integr. Manuf. 27 (1) (2011) 135–149.

[5] Y.-H. Wang, T.-H.S. Li, C.-J. Lin, Backward q-learning: The combination of SARSA algorithm and q-learning, Eng. Appl. Artif. Intell. 26 (9) (2013) 2184–2193.

[6] C. Xia, A. El Kamel, A reinforcement learning method of obstacle avoidance for industrial mobile vehicles in unknown environments using neural network, in: Proceedings of 2014 International Conference on Industrial Engineering and Engineering Management, IEEM, 2015, pp. 671–675.

[7] A. El-Fakdi, M. Carreras, Two-step gradient-based reinforcement learning for underwater robotics behavior learning, Robot. Auton. Syst. 61 (3) (2013) 271–282.

[8] Z. Miljković, M. Mitić, M. Lazarević, B. Babić, Neural network reinforcement learning for visual control of robot manipulators, Expert Syst. Appl. 40 (5) (2013) 1721–1736.

[9] J. Kober, J.A. Bagnell, J. Peters, Reinforcement learning in robotics: A survey, Int. J. Robot. Res. (2013) 237–285.

[10] S. Zhifei, E.M. Joo, A review of inverse reinforcement learning theory and recent advances, in: 2012 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2012, pp. 1–8.

[11] A.Y. Ng, S.J. Russell, Algorithms for inverse reinforcement learning, in: International Conference on Machine Learning, ICML, 2000, pp. 663–670.

[12] P. Abbeel, A.Y. Ng, Apprenticeship learning via inverse reinforcement learning, in: Proceedings of the Twenty-First International Conference on Machine Learning, ACM, 2004, p. 1.

[13] P. Abbeel, A. Coates, A.Y. Ng, Autonomous helicopter aerobatics through apprenticeship learning, Int. J. Robot. Res. (2010).

[14] N.D. Ratliff, J.A. Bagnell, M.A. Zinkevich, Maximum margin planning, in: Proceedings of the 23rd International Conference on Machine Learning, ACM, 2006, pp. 729–736.

[15] N.D. Ratliff, D. Silver, J.A. Bagnell, Learning to search: Functional gradient techniques for imitation learning, Auton. Robots 27 (1) (2009) 25–53.

[16] E. Klein, M. Geist, B. Piot, O. Pietquin, Inverse reinforcement learning through structured classification, in: Advances in Neural Information Processing Systems, 2012, pp. 1007–1015.

[17] B. Piot, M. Geist, O. Pietquin, Learning from demonstrations: Is it worth estimating a reward function? in: Machine Learning and Knowledge Discovery in Databases, Springer, 2013, pp. 17–32.

[18] E. Uchibe, K. Doya, Inverse reinforcement learning using dynamic policy programming, in: 2014 Joint IEEE International Conferences on Development and Learning and Epigenetic Robotics, ICDL-Epirob, IEEE, 2014, pp. 222–228.

[19] D. Ramachandran, E. Amir, Bayesian inverse reinforcement learning, IJCAI 51 (2007) 2586–2591.

[20] M. Lopes, F. Melo, L. Montesano, Active learning for reward estimation in inverse reinforcement learning, in: Machine Learning and Knowledge Discovery in Databases, Springer, 2009, pp. 31–46.

[21] B.D. Ziebart, A.L. Maas, J.A. Bagnell, A.K. Dey, Maximum entropy inverse reinforcement learning, in: AAAI, 2008, pp. 1433–1438.

[22] A. Boularias, J. Kober, J.R. Peters, Relative entropy inverse reinforcement learning, in: International Conference on Artificial Intelligence and Statistics, 2011, pp. 182–189.

[23] Q. Qiao, P.A. Beling, Inverse reinforcement learning with Gaussian process, in: American Control Conference, ACC, 2011, IEEE, 2011, pp. 113–118.

[24] S. Levine, Z. Popovic, V. Koltun, Nonlinear inverse reinforcement learning with Gaussian processes, in: Advances in Neural Information Processing Systems, 2011, pp. 19–27.

Chen Xia received his B.S. degree in Mathematics and Applied Mathematics, and M.S. degree in Electronics and Communication Engineering from Beihang University, China, in 2010 and 2013, respectively. He also held a M.S. degree in Automatic Control and Electrical Systems from École Centrale de Lille, France, in 2012. He is currently a Ph.D. student in the Research Center in Computer Science, Signal and Automatic Control of Lille (CRIStAL—UMR CNRS 9189) at École Centrale de Lille. His research interests include robot learning, reinforcement learning and its applications on robotics, intelligent control systems and machine vision.

Abdelkader El Kamel received the Engineering Diploma, Master Diploma from École Centrale de Lille, France, both in 1990, and the Ph.D. degree in 1994. He is currently a full professor at École Centrale de Lille. From 2008 to 2013, he was in charge of the Computer-Telecommunication-Robotics Department at Ecole Centrale de Pékin, China. He is appointed as a permanent visiting professor in major engineering/business schools in Tunisia, and a visiting professor in Chili, Canada, India and Japan. He has been an invited speaker for more than 25 plenary lectures or tutorials in international conferences and is on the program committee for about 150 IEEE SMC & CSS, IFAC and several other conferences. His major research interests include intelligent systems monitoring, complex systems analysis & control, computational intelligence & optimization. Applying intelligent technologies, virtual reality & optimization techniques in transportation systems and mobile cooperative robots are among the current focus of work.