Neural Inverse Reinforcement Learning with a Nonlinear Policy Representation*

Chen Xia[1*] and Abdelkader El Kamel[1]

Abstract— **Inverse reinforcement learning (IRL) is designed to teach robots to perform specific tasks without manually specifying the reward function. Many existing IRL algorithms proved efficiency while assuming the expert policy to be optimal and deterministic, and were applied to experiments with limited state spaces. This paper proposes a new nonlinear policy representation with help of neural nets to establish a solid mapping between large-scale and high-dimensional state spaces and action spaces. By applying IRL to train this policy representation, we present an efficient algorithm, Neural Inverse Reinforcement Learning (NIRL), which allows the demonstrations to be non-optimal and stochastic. Experiments are conducted on simulated autonomous navigation tasks, and the results show that a mobile robot can learn efficiently from non-optimal human demonstrations, and prove a good generalization performance on undemonstrated states.**

I. INTRODUCTION

Robotics has developed to a level that a smart control system is a necessity, however, designing such reliable systems is usually difficult. Manually programming a robot for every specific control strategy is widely considered as an expensive and intensely time-consuming process. Compared with pre-programming a robot for all behaviors, it is preferable that a robotic system could learn to performed tasks by themselves.

As a result, machine learning approaches are playing an increasingly important role in improving the robot autonomous ability based on accumulated experiences. These methods can be computationally less expensive than classical ones and can largely ease the burden on human designers. Reinforcement learning (RL) [1] is such a learning method for constructing intelligent systems. Under the RL framework, the learner is a decision-making agent that takes actions in an environment and receives a reward as feedback for evaluating the action outcome. Then the agent repeats this process to progressively refine the inner control strategy. Previous studies have demonstrated the successful applications of RL in robotics [2], [3], [4], [5]. However, designing an informative reward function, a crucial part when applying RL, poses a great challenge and is often exhaustive to tune for large and complex problems. Inverse reinforcement learning (IRL) [6] arose and was aimed at discovering the underlying rewards from demonstrated examples of a desired behavior. Abbeel and Ng then developed the original IRL into a new indirect learning approach [7], where the learner cared less about the actual reward function, but tried to output a policy that attains the performance close to that of the expert. This method was later implemented in helicopter control [8].

The maximum margin planning (MMP) algorithm [9] assumes a linearized-features reward and 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 [10], and applied it to outdoor autonomous navigation. The idea of MMP also inspired the structured classification based inverse reinforcement learning (SCIRL) [11], [12], where the authors use only sampled trajectories to reduce IRL to a structured classification problem.

Generally, 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 giving an explicit policy representation in large-scale spaces. Also, reliable examples in those methods need to be optimal. In this paper, we present a new and efficient learning algorithm, neural inverse reinforcement learning (NIRL). We propose an explicit policy representation by adopting modern neural nets to establish a nonlinear mapping between states and actions, and thus to generalize the expert's actions to unvisited regions. We modify the maximum margin method into a model-free version in order to learn the reward function. The proposed NIRL is tested on simulated autonomous robot navigation, which is a typical large-scale state-space problem. The results show that a mobile robot using NIRL can successfully accomplish an autonomous navigation task without colliding with unpredicted obstacles. NIRL is robust and has a good generalization performance on undemonstrated states.

The rest of the paper is organized as follows. The next section describes the fundamental background. Section III describes the proposed nonlinear neural policy representation, and the complete NIRL algorithm is given in Section IV. In Section V, experiments are conducted to show the simulation results of the proposed method, and the final section draws the conclusions.

II. BACKGROUND

This section covers the theoretical background used in this paper.

[1]Chen Xia and Abdelkader El Kamel are with Research Center CRIStAL, UMR CNRS 9189, École Centrale de Lille, 59651 Villeneuve d'Ascq, France.
chen.xia@centraliens-lille.org,
abdelkader.elkamel@ec-lille.fr.
*C. Xia is the correpsonding author.

Preprint submitted to 2016 IEEE Conference on Robotics and Biomimetics.
Received July 24, 2016.

A. Markov Decision Processes

Inverse reinforcement Learning is formulated within the framework of Markov decision processes (MDPs) without a reward function, denoted by MDP\R. A finite-state MDP is defined as a tuple $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma\}$ where: $\mathcal{S}$ is a finite set of states that represents the dynamic environment; $\mathcal{A}$ is a set of actions that could be executed by an agent; $\mathcal{T}$ is a transition probability function, 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 $s' \in \mathcal{S}$; r is a reward function; and $\gamma \in [0, 1)$ 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

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

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 id defined by

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,|\, s_0 = s, a_0 = a\right]. \quad (1)$$

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 $s \in \mathcal{S}$:

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

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

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

B. 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: $\mathcal{D}_E = \{(s_1, a_1), (s_2, a_2), ..., (s_M, a_M)\}$, 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 in $\mathcal{D}_E$. We assume that the reward function is a linear combination of K feature vectors f_k with weights θ_k:

$$\forall (s, a): \quad r(s, a) = \theta^{\top} f(s, a) = \sum_{k=1}^{K} \theta_k f_k(s, a). \quad (2)$$

Eq. (1) can then be rewritten as:

$$Q^{\pi}(s, a) = \theta^{\top} \mu^{\pi}(s, a), \qquad \text{with}$$
$$\mu^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t f(s_t, a_t) \,|\, s_0 = s, a_0 = a\right]. \quad (3)$$

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 [7].

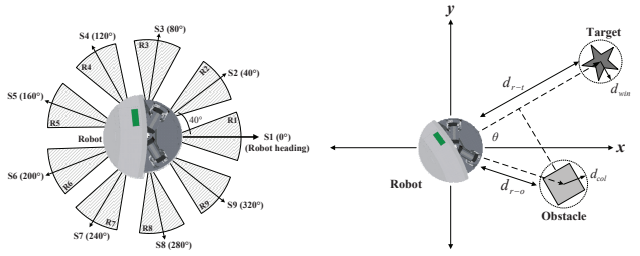
If two policies share the same feature expectations, then they will have the same value function whatever reward $R = \theta^{\top} f$:

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

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}.

C. Mobile Robot Model

This paper uses Robotino® as the mobile robot model, which has three wheels, one in front and two in back. The robot is equipped with nine infrared sensors arranged around its base at an angle of 40° to each other, and each sensor can cover a scope of detection of 40°, as shown in Fig. 1a. $(S_i)_{\{1 \leq i \leq 9\}}$ are the nine sensors and $(R_i)_{\{1 \leq i \leq 9\}}$ representing the corresponding scopes of detection.



(a) The mobile robot model with nine sensors and sensing areas. (b) The mobile robot surrounding with the target and an obstacle.

Fig. 1: The autonomous mobile robot model.

The robot navigation environment consists of its target and the obstacles, as shown in Fig. 1b. The robot knows its initial and target positions, and tries to reach the target in a collision-free path despite of the presence of 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.

Preprint submitted to 2016 IEEE Conference on Robotics and Biomimetics.
Received July 24, 2016.

III. 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. We propose an explicit policy representation by incorporating the neural nets framework, and this representation can ease the learning by IRL.

A. State and Action Spaces

In order to represent a environment state, we first denote a numerical danger level of 0 to 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. Experimental results showed a state represented by seven sensors excluding $S5$ and $S6$ led a better performance. 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 i-th sensor degree of danger.

Besides sensor readings, a state should include two more parameters. One is the target region, $V_t \in \{1, 2, ..., 9\}$, represented by robot sensor regions in Fig. 1a. 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 (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 $\phi : \mathcal{S} \to \mathbb{R}^{16\times 1}$:

$$\phi(s_t) = [D_t \quad V_t \quad I_t \quad U_t]^\top$$

Since the robot needs to deal with different environments, and the robot position coordinates have no direct relation with the target and obstacles in different environments, it is not necessary to include them as part of the state features. 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.

B. 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 $f(s_t, a_t)$ at time instant t. It is a row vector of size $k \times n$:

$$f(s_t, a_t) = [\nu_1 \quad \nu_2 \quad \dots \quad \nu_n]. \tag{4}$$

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

$$\nu_i = \begin{cases} \phi(s_t)^\top, & \text{if } a_t = u_i \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

The state-action features $f(s_t, a_t)$ would be used to expressed 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.,

$$\mathcal{G}(s_t) = \begin{bmatrix} Q(s_t, u_1) \\ Q(s_t, u_2) \\ \vdots \\ Q(s_t, u_n) \end{bmatrix}, \tag{6}$$

where $n = |\mathcal{A}|$. The architecture of the neural policy representation is shown in Fig. 2.

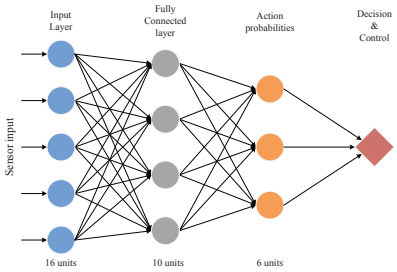


Fig. 2: The nonlinear neural policy representation.

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.

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

$$\begin{aligned} z &= w^{(1)} \cdot x = w^{(1)} \cdot \begin{bmatrix} 1 \\ \phi_t \end{bmatrix}, \\ h &= \text{sig}(z), \\ \mathcal{G}_\omega(s_t) &= \text{sig}\left(w^{(2)} \cdot \begin{bmatrix} 1 \\ h \end{bmatrix} \right). \end{aligned} \tag{7}$$

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

We also denote

$$\mathcal{G}_\omega(s_t, a_t) = \left[\mathcal{G}_\omega(s_t)\right]_i = Q(s_t, u_i), \quad \text{with } u_i = a_t.$$

Preprint submitted to 2016 IEEE Conference on Robotics and Biomimetics.
Received July 24, 2016.

C. Stochasticity of Policy

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

$$\pi_\Omega(s_t,a_t) = P(a_t \mid s_t;\Omega) = \frac{\exp(\eta \mathcal{G}(s_t,a_t))}{\sum_{u\in\mathcal{A}} \exp(\eta \mathcal{G}(s_t))}. \quad (8)$$

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:

$$a_t^\pi = \pi_\Omega(s_t,a_t) = \arg\max_{u\in A} \mathcal{G}(s_t). \quad (9)$$

Since the neural network can approximate any functions, our proposed method is greatly suitable for large state spaces. One may consider the neural policy representation as a black box which containing a nonlinear function between input features and output state-action values.

IV. Neural Inverse Reinforcement Learning

Inverse reinforcement learning algorithms normally require a model of environment, and 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.

A. Suboptimal Demonstration Refinement via Maximum a Posteriori Estimation

When an expert demonstrates one specific task, 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.

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

$$\hat{a}(s_t) = \arg\max_a P(s_t \mid a)P(a). \quad (10)$$

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.

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

B. 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

$$\mu_E = \frac{1}{N_T}\sum_{i=1}^{N_T}\sum_{t=1}^{L_i} \gamma^{(t-1)} f(s_t^{(i)}, a_t^{(i)}). \quad (11)$$

The maximum margin planning method (MMP) [9] 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 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 $l(s,a) \ge 0$:

$$\min_\theta \frac{1}{2}\|\theta\|_2^2 + C\sum_{i=1}^{N_T}\xi^{(i)}, \qquad \text{s.t.} \quad \forall\pi :$$
$$Q_\theta^{\pi_E}(s_t^{(i)}, a_t^{(i)}) + \xi^{(i)} \ge \max_\pi Q^\pi(s_t^{(i)}, a_t^{(i)}) + l(s_t^{(i)}, a_t^{(i)}). \quad (12)$$

where $Q^\pi(s_t^{(i)}, a_t^{(i)})$ is an estimated Q-value of the agent in $s_t^{(i)}$, and is calculated by using our neural policy representation in Section III-B. $Q_\theta^{\pi_E}(s_t^{(i)}, a_t^{(i)})$ is the Q-value of the expert policy, calculated by (3) with the expert feature expectations in (11). 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

Preprint submitted to 2016 IEEE Conference on Robotics and Biomimetics.
Received July 24, 2016.

objective function:

$$J(\theta) = \sum_{i=1}^{N_T} \sum_{t=1}^{L_i} \max_{\pi} \left(Q^{\pi}(s_t^{(i)}, a_t^{(i)}) + l(s_t^{(i)}, a_t^{(i)}) \right) - Q_{\theta}^{\pi_E}(s_t^{(i)}, a_t^{(i)}) + \frac{\lambda_1}{2} \|\theta\|_2^2, \tag{13}$$

where $\lambda_1 \geq 0$ 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 [9], [10], 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 (2).

C. 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:

$$\hat{Q}^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) + \alpha[r_\theta(s_t, a_t) + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) - Q^{\pi}(s_t, a_t)]. \tag{14}$$

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:

$$J(\Omega) = -\frac{1}{N} \sum_{t=1}^{N} \left[\hat{Q}_t^{\pi} \circ \log Q_t^{\pi} + (1 - \hat{Q}_t^{\pi}) \circ \log (1 - Q_t^{\pi}) \right] + \frac{\lambda_2}{2N} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(w_{j,i}^{(l)} \right)^2, \tag{15}$$

where $\hat{Q}_t^\pi = \hat{Q}^\pi(s_t, a_t)$ and $Q_t^\pi = Q^\pi(s_t, a_t)$. 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, 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

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

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

D. 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 (11).

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

Require: 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 (13) 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**

Ensure: 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

Preprint submitted to 2016 IEEE Conference on Robotics and Biomimetics.
Received July 24, 2016.

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.

V. Experimental results

The navigation tasks require the robot to start from an initial position, and plan in real-time a path while avoiding any potential collisions to arrive at a goal place. It is impractical to manually program a control policy for accomplishing such complicated tasks.

The mobile robot is represented by a rectangle-shaped robot, and it is equipped with 9 sensors to observe the surrounding environment, as shown in Fig. 1a. The 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 nonlinear neural policy has three layers: 16 units in the input layer, 10 in the hidden layer and 6 in the output layer. The inputs and outputs are presented above in Section III-A. Some experiment parameters are selected as follows:

- Learning rate of SARSA: $\alpha = 0.8$,
- Learning rate of BPNN: $\sigma = 0.4$,
- Discount factor: $\gamma = 0.65$,

A. Neural Inverse Reinforcement Learning

In this paper, 30 demonstrations are generated by human 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. 3.

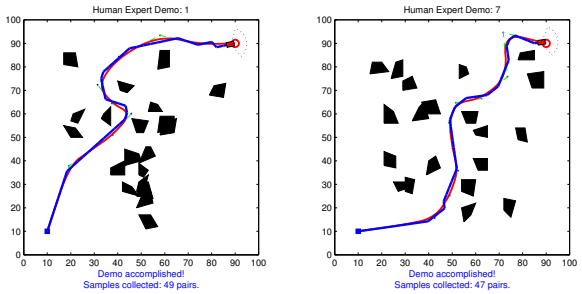
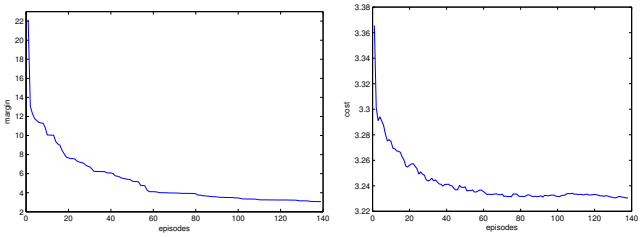


Fig. 3: Human expert demonstrations. The red paths are chosen by the human expert, and the blue paths are travelled by the robot to collect data.

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 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 (13).



(a) Changes of margin during the learning episodes.

(b) Changes of NN cost during the learning episodes.

Fig. 4: Changes of parameters.

Fig. 4a 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 Equation (15) by iteratively updating the weights $W^{(1)}$ and $W^{(2)}$. The result is shown in Fig. 4b.

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.

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. 5.

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.

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 m/s, 2 m/s and 3 m/s. 50 runs were conducted for each velocities. The results are shown in Fig. 6.

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

Preprint submitted to 2016 IEEE Conference on Robotics and Biomimetics.
Received July 24, 2016.

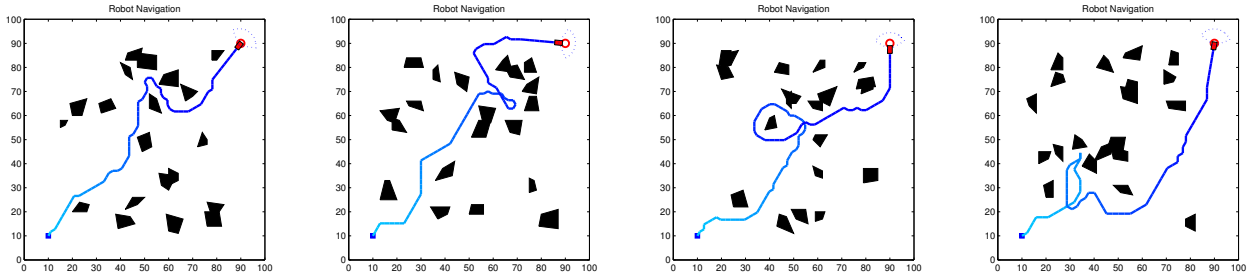


Fig. 5: Robot navigation results in four different environments.

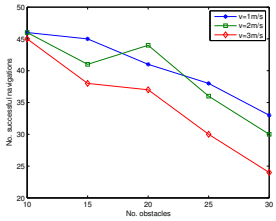


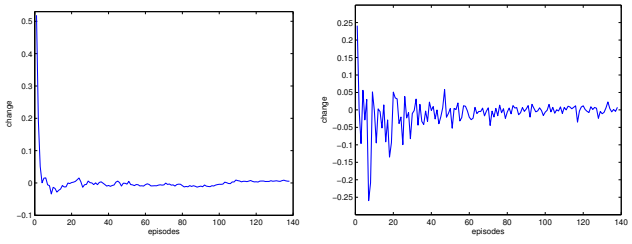
Fig. 6: Comparison of numbers of successful navigations in the environments of different numbers of obstacles.

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.

B. 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. 7a and Fig. 7b. The X-axis represents the number of episode and the Y-axis is the weight change.



(a) Changes of $W^{(1)}$ in learning process. (b) Changes of $W^{(2)}$ in learning process.

Fig. 7: Changes of neural policy weights during the learning process.

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 (14), 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.

C. Autonomous Navigation in Dynamic Environments

We tested the NIRL algorithm in dynamic environments. The results are shown in Fig. 8. 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.

D. Discussions

We compare the NIRL algorithm with three major existing methods. The first one is the apprenticeship learning (AL) in [7], the baseline of the family of inverse reinforcement learning. The second one is the relative entropy inverse reinforcement learning (RE), proposed in [13], a popular IRL algorithm using the theory of entropy. The last one is the structured classification inverse reinforcement learning (SCIRL), presented in [11], a recent work using max-margin method for a batch model-free learning algorithm. We applied these methods on the setting of autonomous navigation.

Preprint submitted to 2016 IEEE Conference on Robotics and Biomimetics.
Received July 24, 2016.

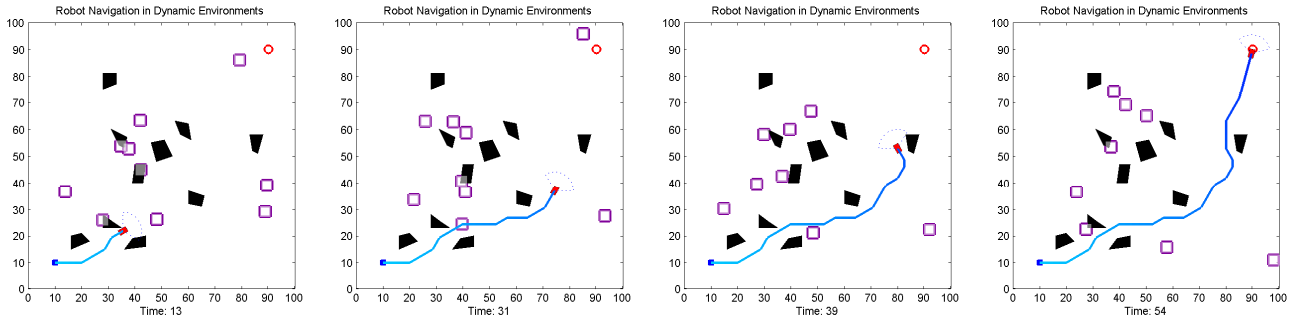


Fig. 8: Autonomous robot navigation in a dynamic environment using NIRL. (displayed in different time steps.)

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 I.

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

TABLE I: Comparison of the rate of success versus different IRL methods.

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.

VI. CONCLUSIONS

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] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.

[2] Y.-H. Wang, T.-H. S. Li, and C.-J. Lin, "Backward q-learning: The combination of sarsa algorithm and q-learning," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 9, pp. 2184–2193, 2013.

[3] C. Xia and A. El Kamel, "A reinforcement learning method of obstacle avoidance for industrial mobile vehicles in unknown environments using neural network," in *Proceedings of the 21st International Conference on Industrial Engineering and Engineering Management 2014*, 2015, pp. 671–675.

[4] A. El-Fakdi and M. Carreras, "Two-step gradient-based reinforcement learning for underwater robotics behavior learning," *Robotics and Autonomous Systems*, vol. 61, no. 3, pp. 271–282, 2013.

[5] Z. Miljković, M. Mitić, M. Lazarević, and B. Babić, "Neural network reinforcement learning for visual control of robot manipulators," *Expert Systems with Applications*, vol. 40, no. 5, pp. 1721–1736, 2013.

[6] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Machine Learning (ICML), International Conference on*, 2000, pp. 663–670.

[7] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1.

[8] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, 2010.

[9] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 729–736.

[10] N. D. Ratliff, D. Silver, and J. A. Bagnell, "Learning to search: Functional gradient techniques for imitation learning," *Autonomous Robots*, vol. 27, no. 1, pp. 25–53, 2009.

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

[12] B. Piot, M. Geist, and 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.

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

Preprint submitted to 2016 IEEE Conference on Robotics and Biomimetics.
Received July 24, 2016.