Inverse Reinforcement Learning based on Critical State

Kao-Shing Hwang Tien-Yu Cheng Wei-Cheng Jiang

Department of Electrical Engineering
National Sun Yat-sen University

Abstract

Inverse reinforcement learning is tried to search a reward function based on Markov Decision Process. In the IRL topics, experts produce some good traces to make agents learn and adjust the reward function. But the function is difficult to set in some complicate problems. In this paper, Inverse Reinforcement Learning based on Critical State (IRLCS) is proposed to search a succinct and meaningful reward function. IRLCS select a set of reward indexes from whole state space through comparing the difference between the good and bad demonstrations. According to the simulation results, IRLCS can search a good strategy that is similar to experts.

Keywords: Inverse Reinforcement learning, reward function, reward feature.

1. Introduction

In recent years, Inverse Reinforcement Learning (IRL) is widely used and studied. The [1] proposes a Hierarchical Apprenticeship Learning method for quadruped locomotion. In addition, the others researches apply the IRL in the different problems [2, 3]. The original concepts of IRL algorithm are proposed by [4] which is a QP-based algorithm; nevertheless, the methods derive some problems. First of all, it needs to predefine a set of reward indexes or reward features. The FIRL method is introduced to search an appropriate reward features via Fitting Step [5]. However, the method spends more time finding a set of useful reward function. In [6], the dimension reduction methods are proposed to extract useful features by demonstrations. However, it bases on quadratic programing and makes the application inconvenient. Secondly, most IRL problems need to have many demonstrations that are demonstrated by experts. These demonstrations are viewed as correct behaviors. But, incorrect demonstrations should be viewed important information equally. The agent can learn good behaviors by incorrect demonstrations. In this paper, IRLCS algorithm is proposed to search appropriate reward indexes in whole state space by two sets of demonstrations, good trajectories and bad trajectories. IRLCS uses the significant states to form a portable and succinct reward function in short computational time. The efficiency of IRLCS is demonstrated by a simulation problem, speeding car, which is introduced by [5]. The paper is organized as follows. Section 2 introduces reinforcement learning and inverse reinforcement learning. Section 3 introduces the IRLCS algorithm**.** Section 4 summarizes the simulation and analysis results. Section 5 is conclusion about the algorithm.

2. Background

2.1 Reinforcement Learning

Reinforcement learning (RL) is a sub-area of Machine learning. In comparison with supervised learning, reinforcement learning can solve problems without training samples. Instead of training samples, a reward function is defined by the expert to express what the good and bad events are for the agent. In a reinforcement learning process, an agent takes actions in an environment so as to maximize the long-term reward.

2.2 Inverse Reinforcement Learning

Reinforcement learning techniques provide a powerful solution for the decision making problems under uncertainty. In the beginning of a reinforcement learning process, we have to define a reward function that guides the agent towards the goal. Unfortunately it is difficult to define a reward function for a complex problem. In order to solve this problem, inverse reinforcement learning algorithms are proposed [3] [4]. In inverse reinforcement learning algorithm, the reward function is approximated using a linear combination of useful features:

$$R(s) = \boldsymbol{w} \cdot \phi(s) \qquad (1)$$

where $\phi(s) = [\phi_1(s), \phi_2(s), ..., \phi_i(s)]^T$ are predefined basis functions, and l is the number of useful features in the reward function, and $w(s) = [w_1, w_2, ..., w_i]$ are the parameters to be tuned during the learning process. We have to define the useful reward feature factor and provide example traces in the beginning of an inverse reinforcement learning process. The goal of the inverse reinforcement learning is finding a reward function by which agent can learn a good strategy that close to the expert strategy.

3. Proposed method

3.1 Apprenticeship Learning

Apprenticeship learning [4], or apprenticeship via inverse reinforcement learning (AIRL), is a concept in the field of Artificial Intelligence and Machine learning, developed by Pieter Abbeel and Andrew Ng. AIRL deals with Markov decision process without explicitly given a reward function, but where instead we can observe an expert demonstrating the task that we want to learn to perform.

3.2 Inverse Reinforcement Learning via Orthogonal Projection

Reward function, the most succinct representation of the designer's intention, needs to be provided beforehand in reinforcement learning processes. However, it is difficult to design an appropriate reward function in order to fit a complex problem. An inverse reinforcement learning (IRL) algorithm is useful when we solve a MDP without reward functions.

3.2.1 Reward Index

To search a reward function through IRL, we approximate the reward function using a linear combination of useful features. In equation (1) we desire to search a policy by which navigating a mobile robot to the goal. There are two dimensions in this case; the distance and the angle. As Fig. 1, each dimension we can divide into three blocks, so that we define the reward features as $\boldsymbol{F}=[d_0,d_1,d_2,\theta_0,\theta_1,\theta_2]$. If the state $\boldsymbol{s}$ is (d_0,θ_1), the basis vector $\phi(s)=[1,0,0,0,1,0]^T$. Nevertheless, the aforementioned definition is not appropriate to some special situations. For instance, there are obstacles in the environment, and furthermore there is an obstacle between robot and goal. As shown in Fig. 2, $\boldsymbol{F}=[d_0,d_1,d_2,\theta_0,\theta_1,\theta_2,r_0,r_1,\psi_0,\psi_1]$ .In this situation, the nearest distance to goal is not a good state. On the other hand, one dimension is relative to another dimension. In order to solve the problem, we can define the set of reward indexes K as $\{K|K\subseteq \boldsymbol{S}\}$, where $\boldsymbol{S}$ represents the whole state space. An example is shown as follows:

$$K=\{(d_0,\theta_1,r_0,\varphi_0),(d_0,\theta_1,r_1,\varphi_0),(d_2,\theta_2,r_1,\varphi_0)\}$$
$$\phi(s)=[0,1,0]^T, if\ \ s=(d_0,\theta_1,r_1,\varphi_0)$$
$$\phi(s)=[0,0,1]^T, if\ \ s=(d_2,\theta_2,r_1,\varphi_0)$$
$$\phi(s)=[0,0,0]^T, if\ \ s=(d_0,\theta_0,r_1,\varphi_1)$$

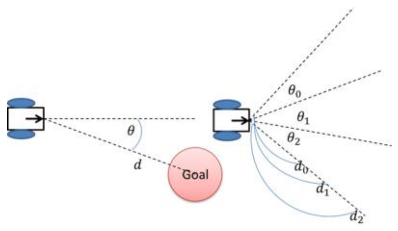

Fig. 1: Mobile robot navigation.

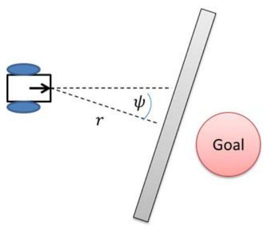

Fig. 2: Obstacle in the environment.

3.2.2 Iteration Algorithm

Orthogonal projection IRL algorithm is shown in Fig. 3. Instead of reward functions, example traces is provided beforehand, so that we can quantize the designer's intention as reward functions. Besides, a set of useful reward indexes is needed in the beginning of the iteration algorithm. In order to approach the example traces, we search reward functions cyclically. We estimate the distance to example traces via index expectations. The derivation of index expectations is shown as follows:

$$E_{s_0\sim I}\left[V^{\pi}(s_0)\right]=\omega\cdot E\left[\sum\nolimits_{t=0}^{\infty}\gamma^t\phi(s_t)\middle|\pi\right] \qquad (2)$$
$$\mu(\pi)=E_{s_0\sim I}\left[\sum\nolimits_{t=0}^{\infty}\gamma^t\phi(s_t)\middle|\pi\right]\in R^k \qquad (3)$$

where $\boldsymbol{I}$ is the initial-state distribution, from which the start state $\boldsymbol{s}_0$ is chosen; $\gamma(0<\gamma\leq 1)$ is discount factor. A policy π is a mapping from states to probability distributions over actions. In RL, a state value $V(\boldsymbol{s})$ is used to evaluate sum of rewards from an initial state to terminal state as depicted in equation (2). Equation (3), we define index expectations of a trajectory which start from an initial state $\boldsymbol{s}_0$ through a policy $\pi^{(i)}$ as equation (4). If there is more than one trajectory, the index expectations are computed by equation (5), where m is the number of trajectories. The equation (6) is a Euclidean norm. It is a distance between the expert's trajectory and the trajectory through the current policy. If the distance is less than or equal to a constant threshold, the current policy is good enough and the reward functions by which search a good policy will be outputted. On the contrary, we compute a new set of reward weights through equation (7) to get close to the expert's example traces. If the iterative process has not been terminated, a new iteration point is chosen by equation (8).

$$\mu(\pi)=\sum\nolimits_{t=0}^{\infty}\gamma^t\phi(s_t) \qquad (4)$$
$$\mu(\pi)=\frac{1}{m}\sum\nolimits_{j=1}^{m}\sum\nolimits_{t=0}^{\infty}\gamma^t\phi(s_t^{(j)}) \qquad (5)$$
$$s^{(j)}=\left\|\mu_E-\mu(\pi^{(j-1)})\right\|_2 \qquad (6)$$
$$\boldsymbol{w}^{(i)}=\mu_E-\mu(\pi^{(j-1)}) \qquad (7)$$
$$\mu^{(i)}=\mu^{(i-1)}+\frac{\left(\mu(\pi^{(i)})-\mu^{(i-1)}\right)^T\left(\mu_E-\mu^{(i-1)}\right)}{\left(\mu(\pi^{(i)})-\mu^{(i-1)}\right)^T\left(\mu(\pi^{(i)})-\mu^{(i-1)}\right)}\left(\mu(\pi^{(i)})-\mu^{(i-1)}\right) \qquad (8)$$

The IRL algorithm mentioned above can solve a Markov decision process without reward functions. However, it is hard to define reward indexes in a complex dynamic environment. We proposed a reward index construction method that can be used in a complicated problem.

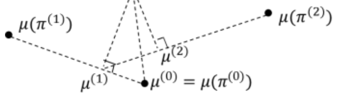

Fig. 3: Three iterations through orthogonal projection.

3.3 Reward Index Construction

In this section, we define the states visited by the good and bad demonstrations as the source domain, the set of critical states is the target domain. Then, we can choose significant states through observing the difference between the good and bad demonstrations, and we regard those significant states as reward indexes. We proposed a method that selects reward indexes from state space via impurity function. The entropy function expresses the degree of mess in a system. This measure is based on the concept of entropy in information theory. Entropy function is shown as follows:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_n p(x_i) \quad (9)$$

where $\boldsymbol{X}$ is a set that $\boldsymbol{X} = [b_0, b_1, b_2, ..., b_m]$, x_i is a subset of $\boldsymbol{X}$. n is the number of subsets. We define $0\log 0 \equiv 0$.

3.3.1 Visit frequency of states

We assume that a set of reward indexes is a subset of state space. Furthermore, we find out these critical states via the difference between the good and bad demonstrations. There is a straightforward factor, visit frequency of states, which can be used in searching critical states. For instance, one state is often visited by a good policy, and it is seldom visited by a bad policy. We can take this state as a good state. On the contrary, the state will be a bad state. If one state we need to approach or avoid, like the above-mentioned two cases, we take the state as a critical state. Besides, if the visit frequency of state is similar in good and bad demonstrations, the state is an insignificant state. We compute the state-impurity factor by equation (10), which presents a lopsided degree in a certain state.

$$f_s(s) = -p_{sG}(s) \log_2 p_{sB}(s) \log_2 p_{sB}(s) \quad (10)$$

where P_s is a visit probability, the subscript G means a good demonstration, and B means a bad demonstration.

3.3.2 Visit frequency of state-action pair

In addition to observe the state distributions as mentioned above, we observe the visit frequency of state-action pairs to search critical states. There are two conditions we will take a state as a critical state. First, while the action-impurity factor of the state is low, the state is a critical state. The action-impurity factors that calculated via entropy function as shown in (11) indicate degree of mess in one state.

$$H(s) = -\sum_{i=1}^{n} p(s,a) \log_n p(s,a) \quad (11)$$

In this case, we substitute the size of action space into n, and $p(\boldsymbol{s}, a)$ is the probability of choosing action a in a state $\boldsymbol{s}$.

The second condition that take a state as a critical state while one state in which the good policy is different from the bad policy. We proposed a method to quantize this situation. First, we calculate a total trajectory through accumulating the visit frequency of state-action pairs demonstrated by the good and bad policy. Second, we compute action-impurity factor of total trajectory. While one state in which the good policy is different from the bad policy, action-impurity factor of total trajectory will be much greater than the good policy one. Besides, if the action-impurity factor of good policy is high, there is an uncertain policy in this state. In this situation, it is an insignificant state. To sum up, we compute action-impurity ratio by equation (12), which quantizes the difference between the good and bad policy in a certain state.

$$f_a(s) = \frac{-\sum_{i=0}^{|A|} p_{aSun}(s,a_i) \log_{|A|} p_{aSum}(s,a_i) + \varepsilon}{-\sum_{i=0}^{|A|} P_{aG}(s,a_i) \log_{|A|} P_{aG}(s,a_i) + \varepsilon}, \quad (12)$$

$$\varepsilon = 0.001$$

where |A| is size of action space, P_a ($\boldsymbol{s}$, a_i) is probability of choosing action a_i in a state $\boldsymbol{s}$, the subscript G means a good policy, and Sum means summation of good and bad policy. The constant ε is a smoothing factor to avoid denominator equal to zero.

3.4 Inverse Reinforcement Learning Based on Critical State

Based on the above concept, we propose an algorithm, Inverse Reinforcement Learning based on Critical State (IRLCS), which is able to do self-organization and search an appropriate reward function through the good and bad demonstrations. In the beginning of the algorithm, we have to provide two example traces, the good demonstration $\boldsymbol{D_G}$ and the bad demonstration $\boldsymbol{D_B}$. In the same environment, it may not only one goal. For example, in a car driving problem, sometimes we want to avoid collision, and sometimes want to drive as fast as possible. Therefore, the good demonstration is relevant to the objective of task. On the contrary, the bad demonstration we can do a random operation. Moreover, we may do a bad operation to cause the mission to fail intentionally. In our algorithm, we check the entire state space to search for critical states. In Step (2.1), we count the visit frequency of state $\boldsymbol{s}_i$ in $\boldsymbol{D_G}$ and $\boldsymbol{D_B}$. Besides, the sum of two counters is assigned to a total counter $\boldsymbol{C_{Sum}}$. If $\boldsymbol{C_{Sum}}(\boldsymbol{s}_i)$ is equal to zero, we regard the state $\boldsymbol{s}_i$ as an insignificant state, because the state has never been visited by all demonstrations. Next, we can compute a state-impurity factor through Step (2.3) and Step (2.4). The number of actions chosen in a certain state is a factor to search critical states. Therefore, we compute action-impurity ratio $f_a(\boldsymbol{s}_i)$ through Step 2.6 and Step 2.7. In Step 2.8 and Step 2.9, we determine whether state $\boldsymbol{s}_i$ is a critical state. There are two thresholds provided beforehand; $TH_s(0 \leq TH_s \leq 1)$ and $TH_a(1 \leq TH_a)$. The smaller TH_a, the more strict condition is. If TH_s is equal to zero, the state is a critical state while one of demonstrations has never visited the state. On the contrary, the larger TH_a, the more strict condition is. We can adjust these two parameters to control the desired amount of a critical states and the accuracy of the reward function. Finally, in Step (3), we can use the IRL method introduced in section 3.2 to search the appropriate reward function based on critical states. Roughly, the IRL function used in IRLCS is the same as the method introduced in section 3.2. In Step 4.4, Q-Learning algorithm is applied to search a policy $\pi^{(i)}$. The policy used to select an action in the action space is epsilon greedy. There are a parameter, ε, means exploration probability. In exploration, we randomly choose an action from action space. Otherwise, we take a greedy action that causes a maximum state-action value in current state.

```
$\boldsymbol{S}=\{\boldsymbol{s}_0, \boldsymbol{s}_1,\dots, \boldsymbol{s}_n\}$   $\boldsymbol{A}=\{a_0, a_1,\dots, a_n\}$  $K$: set of critical states
1.  Initialize:
        $\boldsymbol{D_G}$ : set of state-action pairs in a good demonstration
        $\boldsymbol{D_B}$ : set of state-action pairs in a bad demonstration
2.  for $i$ = 0 to $|\boldsymbol{S}|$ - 1 step 1
    2.1  Count the visit frequency of state $\boldsymbol{s}_i$ in $\boldsymbol{D_G}$ and $\boldsymbol{D_B}$, $\boldsymbol{s}_i \in \boldsymbol{S}$
             $\boldsymbol{C}_G(\boldsymbol{s}_i)$ : the visit frequency of state $\boldsymbol{s}_i$ in $\boldsymbol{D_G}$
             $\boldsymbol{C}_B(\boldsymbol{s}_i)$ : the visit frequency of state $\boldsymbol{s}_i$ in $\boldsymbol{D_B}$
             $\boldsymbol{C}_{sum}(\boldsymbol{s}_i) \leftarrow \boldsymbol{C}_G(\boldsymbol{s}_i)+\boldsymbol{C}_B(\boldsymbol{s}_i)$
    2.2  if $\boldsymbol{C}_{sum}(\boldsymbol{s}_i) = 0$ then
    2.3  Calculate the visit probability of $\boldsymbol{s}_i$ in demonstrations
             $P_{sG}(\boldsymbol{s}_i)=\boldsymbol{C}_G(\boldsymbol{s}_i)/\boldsymbol{C}_{Sum}(\boldsymbol{s}_i)$,  $P_{sB}(\boldsymbol{s}_i)=\boldsymbol{C}_B(\boldsymbol{s}_i)/\boldsymbol{C}_{Sum}(\boldsymbol{s}_i)$
    2.4  Calculate the state-impurity factor, $f_s(\boldsymbol{s}_i)$,by (10)
    2.5  Count the visit frequency of state-action pairs $(\boldsymbol{s}_i, a_j)$ in $\boldsymbol{D_G}$ and $\boldsymbol{D_B}$, $\boldsymbol{s}_i \in \boldsymbol{S}$, $a_j \in \boldsymbol{A}$
         for $j = 0$ to $|\boldsymbol{A}|$ - 1 step 1
             $\boldsymbol{C}_G(\boldsymbol{s}_i, a_j)$ : the visit frequency of $(\boldsymbol{s}_i, a_j)$ in $\boldsymbol{D_G}$
             $\boldsymbol{C}_B(\boldsymbol{s}_i, a_j)$ : the visit frequency of $(\boldsymbol{s}_i, a_j)$ in $\boldsymbol{D_B}$
             $\boldsymbol{C}_{sum}(\boldsymbol{s}_i,a_j) \leftarrow \boldsymbol{C}_G(\boldsymbol{s}_i,a_j)+\boldsymbol{C}_B(\boldsymbol{s}_i,a_j)$
         end for
    2.6  Calculate the probability of choosing action $a_j$ in a state $\boldsymbol{s}_i$.
         for $j = 0$ to $|\boldsymbol{A}|$ - 1 step 1
             $P_{aG}(\boldsymbol{s}_i,a_j)=\boldsymbol{C}_G(\boldsymbol{s}_i,a_j)/\sum_{j=0}^{|A|}\boldsymbol{C}_G(\boldsymbol{s}_i,a_j)$,
             $P_{aSum}(\boldsymbol{s}_i,a_j)=\boldsymbol{C}_{Sum}(\boldsymbol{s}_i,a_j)/\sum_{j=0}^{|A|}\boldsymbol{C}_{Sum}(\boldsymbol{s}_i,a_j)$
         end for
    2.7  Calculate the action-impurity ratio, $f_a(\boldsymbol{s}_i)$,by (12)
    2.8  if $f_s(\boldsymbol{s}_i) \le TH_{s'}$ then $\boldsymbol{s}_i$ insert in $K$
    2.9  else if $f_a(\boldsymbol{s}_i) \le TH_{a'}$ then $\boldsymbol{s}_i$ insert in $K$
    end for
3.  Search a reward function $R(\boldsymbol{s})$ by inverse reinforcement learning algorithm
    3.1  $R(\boldsymbol{s}) \leftarrow$ inverse reinforcement learning($\boldsymbol{D_G}$, $K$)
```

Fig. 4: Inverse Reinforcement Learning based on Critical State algorithm.

4. Simulation

In this paper, we tested the efficiency of the IRLCS algorithm proposed through a speeding car environment which has been used to test the usability of the renowned IRL algorithms [5]. The simulation environment and the key performance index are illustrated in the following.

4.1 Environment

Fig. 4 is the simulation scenario that we test the IRLCS algorithm. The task is to navigate a car on a three-lane highway. All vehicles, except for the agent's marked in blue, are moving at a speed, said level 1, and appear from the top of the screen randomly. The agent can drive at speeds 1 to 4, and can move one lane left or right. There are five actions, shift agent right, speed up, shift agent left, speed down, do nothing. The objectives of this task are that driving as fast as possible, but sincerely taking account of collision-avoidance and speeding-avoidance. There are eight dimensions in the state space of this task:

1. There is a vehicle at most n car-length in front of the agent. There are equivalent features for checking for cars in three lanes. Length n is in the range from 0 to 3 car-lengths.
2. There is a vehicle at most n car-length behind the agent. There are equivalent features for checking for cars in three lanes. Length n is in the range from 0 to 2 car-lengths.
3. Four speeds at which the agent can drive.
4. Three lanes the agent can occupy.

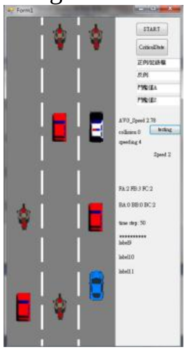

Fig. 4: Speeding car simulation.

Please do not add page numbers to this style; page numbers will be added by the publishers.

In this simulation, there are two additional rules:

1. The lawful driver prefers to drive fast, but does not exceed speed 2 in the right lane, or speed 3 in the others lane.
2. If the agent is speeding, the out-law speed will not calculate in the average speed.

4.3 Simulation results

The results are described in two parts. There are "collision-avoidance behavior" and "driving-as-fast-as-possible behavior". Since the algorithm may learn different policies based on the same set of the critical states due to the simulation scenarios are randomly generated. Each learned policy was executed 50 times as a trail to show the average. We took totally 10 policies for trials and calculated the mean and standard deviation for each kind of parameter settings.

4.3.1 Collision-avoidance behavior

To verify the validity of the threshold values, we provide different kind of parameters to aggregate different critical states. In addition, we compare these settings with the reward indexes constructed through whole state space. The simulation parameters are shown in Table 1. In Fig. 5, the horizontal axis represents different reward indexes, and the vertical axis represents how many time steps in a collision state. The first result that has 148 critical states is the most stringent parameter setting. It is difficult to demonstrate a bad trajectory without any good state, so overly stringent parameter settings may cause that some significant states are filtered. The situation that we use whole state space as reward indexes as shown in item whole ***S***. The item whole ***S*** has more complete information because of there are some significant states in state space which are never visited in example traces. However, the item whole ***S*** contains a lot of insignificant states. It leads item whole ***S*** to a worse result and costs more computational time. According to the results of collision time steps, the item cs154 is the best parameter setting. It is even better than item whole ***S*** which has full information.

Table 1 Simulation parameters in case 1.

Item	cs148	cs150	cs151	cs154
CS	148	150	151	154
TH_S	0	0	0.3	0.6
TH_A	100	1.1	1.1	1.1

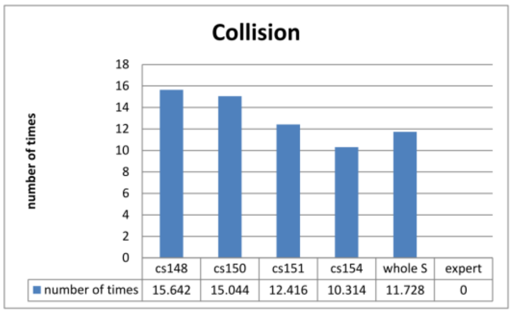

Fig. 5: Collision frequency in case 1.

Fig. 6 represents average speed in 10 trials. The item cs154 has the highest mean value and the lowest standard deviation. These results confirm the effectiveness of a succinct reward function based on critical states.

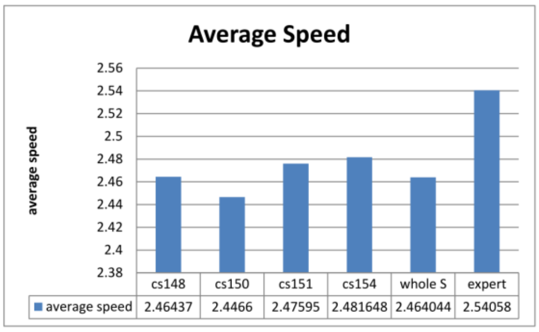

Fig. 6: Average speed in case 2.

4.3.2 Driving-as-fast-as-possible behavior

As mentioned in previous section, we compare different sets of critical states with the reward indexes constructed via whole state space. The simulation parameters are shown in Table 2. According to the simulation results in this case, though the cs134 and cs139 results through IRLCS algorithm is worse than the item whole ***S*** in collision frequency test, it is better in average speed test, as shown in Fig.7 and Fig.8. In addition, according to a "driving as fast as possible" behavior, the policy through IRLCS is more close to the expert's policy because of the high average speed. The result of expert's average speed is lower than item cs134, cs139 and whole ***S*** because of the reaction time of the human operator.

Table 2 Simulation parameters in case 2.

Item	cs132	cs134	cs139
CS	132	134	139
TH_S	0	0	0.5
TH_A	100	1.1	1.1

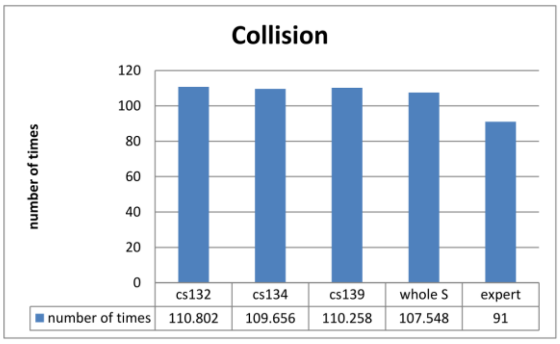

Fig. 7: Collision frequency in case 2.

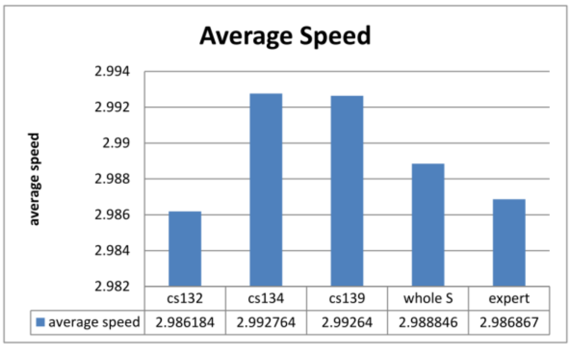

Fig. 8: Average speed in case 2.

5. Conclusions

According to the simulation results in section 4, IRLCS that we proposed is able to find a useful and succinct reward function to learn a behavior demonstrated by expert. The reward index construction methods only use the good trajectories that are demonstrated by experts. IRLCS not only considers the good demonstrations, but also uses the bad demonstrations. Furthermore, the policy through IRLCS is more close to the expert's policy than the policy via IRL algorithm based on whole state space. Moreover, with the dimension increases, the computational cost will be exponential amplification. In summary, IRLCS algorithm can find a useful reward function in low computational time.

References

[1] J. Kolter, P. Abbeel and A. Ng, Hierarchical apprenticeship learning with application to quadruped locomotion, *Advances in Neural Information Processing Systems*, vol. 20, 2008.

[2] P. Abbeel, A. Coates and A. Ng, Autonomous helicopter aerobatics through apprenticeship learning, *International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[3] P. Abbeel, D. Dolgov, A. Ng, and S. Thrun, Apprenticeship learning for motion planning with application to parking lot navigation, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1083-1090, 2008.

[4] P. Abbeel and A. Ng, Apprenticeship learning via inverse reinforcement learning, in *Proceedings of the 21st international conference on Machine learning*, 2004.

[5] S. Levine, Z. Popovic and V. Koltun, Feature construction for inverse reinforcement learning, *Advances in Neural Information Processing Systems*, 2010.

[6] S.-Y. Chen, H. Qian, J. Fan, Z.-J. Jin and M.-L. Zhu, Modified reward function on abstract features in inverse reinforcement learning, *Journal of Zhejlang University - Science C*, vol. 11, no. 9, pp. 718-723, 2010.