Hide and Seek Using Function-Based Q learning

Yu-Hui Chen, Wen-Hao Kao, Mu-Lin Tsai

Abstract - We want to simulate the situation of the police chase in reality, and implement it in a 30×30 map using function-based Q learning to train two robots.

I. INTRODUCTION

In the reality, police chase is an important task in maintaining public order, so it is necessary to find a efficient way to fix this task. To fix this task , we try to use a simple map to simulate the police chase situation. In this simulation, we have the following objects a lot of pressure plates (to imitate monitor) , two robots (to play the police role and the thief role respectively) and two points in the map will be set up as the outlets (the only way of the thief to escape from the police is arriving one of the outlets) . The simulation will base on Q-learning and we let two robots learn how to escape from the other and how to catch the other by the Q-learning algorithm, respectively. The section II will explain the problem formulation and solutions in detail.

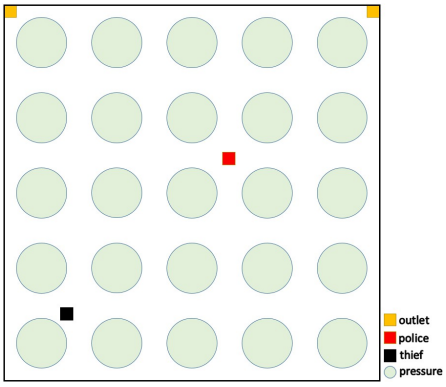

Fig.1 Environment

II. PROBLEM FORMULATION AND SOLUTIONS

The goal of this project is to train two turtle bot call police and thief respectively, playing hide and seek by function-based Q learning. As Fig.1 shows, there are two robots in the environment. The initial positions of two robots are random except the two outlets, and there are 25 pressure plates on the given map. When the thief moves to points that in the range pressure plates(monitor) can detect, the police will get the information of current predict position of the thief, then the police can take action based on the information. The speed of the police is two times faster than the thief, since if the police has the same speed with the thief, chasing the thief will be very difficult for the police, so we set the police in higher speed. If the thief is arrested by the police or the thief arrives one of the outlets successfully, then a round of hide and seek will be finished. We will train two robots for many rounds to find the optimal solutions, based on Q learning,

$$Q(s,a) = w^T f = \sum_{i=1}^{n} w_i f_i$$

where Q is the Q function approximated by some features f_i and w_i is the weighting vectors,

$$w_i \leftarrow w_i + \alpha[R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)]\frac{\partial Q(s,a)}{\partial w_i}$$

we find weighting vectors w_i by gradient descent where a is action, s is state and a' is action at next state s', α is learning rate and γ is discount. [1]

First we have to give the police and the thief features(f), action(a), reward(R), learning rate(α), and discount(γ), and the initial Q-values of all pairs of the state and the action($Q(s,a)$). After the initial condition and parameters are set up, the first thing

is letting two robots, the police and the thief, take action considering the initial condition, then, by the approach of gradient descent, we can update the weighting of all features for the police and the thief after a turn is finished(The police and the thief move once respectively is a complete turn). After all the weighting vectors are updated, we can calculate the new Q-function values of all pairs of the state and the action Q(s,a) and update them. Repeat the above steps(take action, update weighting, update Q-value) will improve the action mode of both the police and the thief, and they will act reasonably. But if we choose the parameters badly(especially for reward and feature), then the training result will not be good and hence they can't move like what we want. So, giving these parameters is the hardest and most important thing in function-based Q learning, we need to try many possible case to make Q learning effective.

III. FUNCTION-BASED Q LEARNING

In function-based Q learning we give the police six features $f1$: constant, $f2$: the minimum distance to the two outlets, $f3 : x$ coordinate, $f4 : y$ coordinate, $f5$: the vertical distance from the police to the line between the thief and the outlet (Fig.2), $f6$: the distance of the police and the predict thief position.

The action of the police is move forward, move to left front, move to right front, move backward and stay. The reward of the police is 10 if police catch thief, -10 if thief escape, -0.1 go to the next state. The learning rate and the discount are 0.001 and 0.9 respectively.

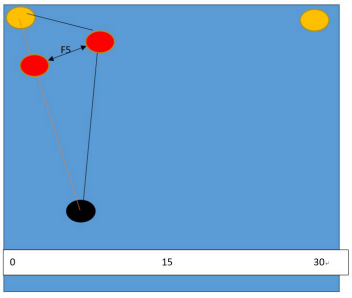

Fig.2 f5 of police

Then give the thief six features $f1$: constant, $f2$: the minimum distance to the two outlets, $f3 : x$ coordinate, $f4 : y$ coordinate, $f5$: the number of times that stay on the pressure plate, $f6$: the absolute value of the radian between the orientation of the thief and the line from the thief to the outlet (Fig.3).

The action of the thief is move forward, move to left front, move to right front and move backward. The reward of the thief is -10 if police catch thief, 10 if thief escape, -0.1 go to the next state. The learning rate and the discount are 0.001 and 0.9 respectively.

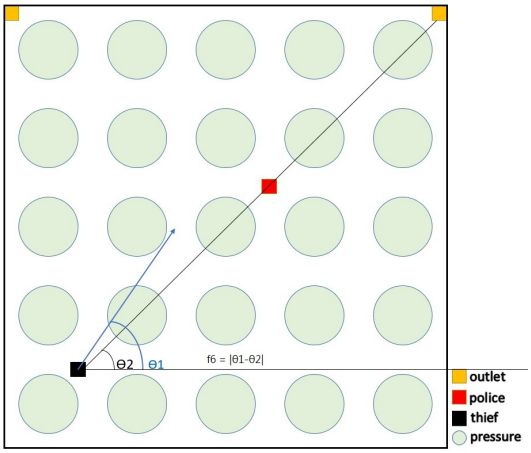

Fig.3 f6 of thief

IV. SIMULATION

We use python and gazebo to simulate this project, the difference between python and gazebo is that gazebo closer to reality. In gazebo, after training six times we get the police weighting (Fig.4) and the thief weighting (Fig.5). However, after finishing the 7th train, the weighting of the thief diverge to infinity, this result does not meet our expectations since we can get the converge weighting of the thief when we use python to simulate. After checking our code repeatedly, we still don't know where the problem is. Then we try to change the reward, action or the learning rate, the weighting still remain divergent. Although we just train six times, in Fig.4 we can figure out that which feature is more important.

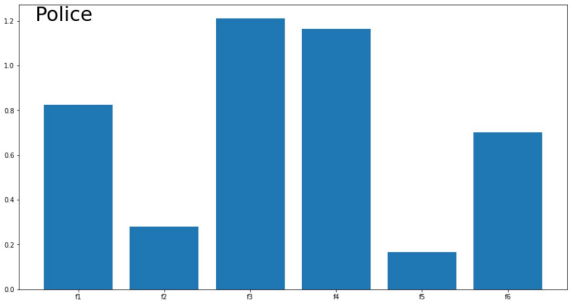

Fig.4 police weighting

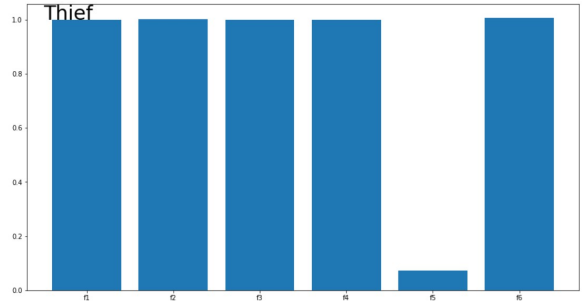

Fig.5 thief weighting

V. FUTURE WORKS

1. Bayes theory [2]
   In our current training results, the police tend to wait for the thief at the end, rather than ambush and intercept the thief based on the location of the thief. We think this is because the police are not effective in predicting the location of the thief, so we want to fix this problem. In order to enhance the police's detection capabilities, we will use Bayes filter to calculate the forecast result.

2. DQN
   In function-based Q learning, the hardest thing is to find the effective features. So when we don't know which features are valid, we want to adopt the DQN(Deep Q Network) model (Fig.6). In this way, DQN can find the features by itself make sure that the features are effective.

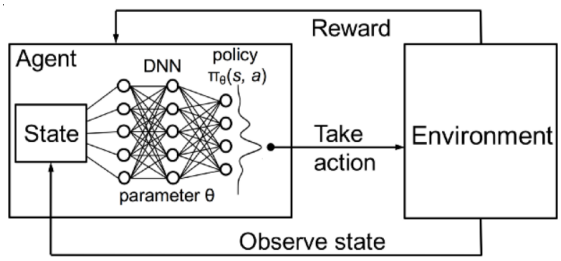

Fig.6 DQN [3]

3. Obstacle setup
   At present, our experiment has no obstacles in the map. In other words, there is a big gap between us and the situation in reality. It can be seen that the effect of implementation in reality will definitely be bad. In order to be implemented in reality in the future, we must let the robot know how to avoid obstacles.

VI. REFERENCE

[1] Kuo-Shih Tseng, "Search for People Using Kernel-Based Reinforcement Learning"

[2] Kuo-Shih Tseng, "Bayes Theorem Over Time"

[3] Hongzi Mao, Mohammad Alizadeh, Ishai Menachey, Srikanth Kandulay, "Resource Management with Deep Reinforcement Learning"

[4] Mohamed K. Gunady1, Walid Gomaa1, Ikuo Takeuchi, "Multi-Agent Task Division Learning in Hide-and-Seek Games"