Comparison of Several Machine Learning Techniques in Pursuit-Evasion Games

Jacky Baltes and Yongjoo Park

Centre for Image Technology and Robotics
University of Auckland,Auckland
New Zealand
j.baltes@auckland.ac.nz
http://www.citr.auckland.ac.nz/~jacky

Abstract. This paper describes the results of an empirical evaluation comparing the performance of five different algorithms in a pursuit and evasion game. The pursuit and evasion game was played using two robots. The task of the pursuer was to catch the other robot (the evader). The algorithms tested were a random player, the optimal player, a genetic algorithm learner, a k-nearest neighbor learner, and a reinforcement learner. The k-nearest neighbor learner performed best overall, but a closer analysis of the results showed that the genetic algorithm suffered from an exploration-exploitation problem.

1 Introduction

This paper describes the results of an empirical evaluation comparing the performance of three different machine learning algorithms in a pursuit and evasion game.

Because of the inherent difficulty of writing control programs for robots, many researchers have used machine learning techniques to find efficient methods for controlling a robot in the real world. Many of these approaches use machine learning techniques as function optimization. In [?], we describe the use of reinforcement learning to learn the control function of a mobile robot for path tracking. Most of this and similar other approaches focus on acquisition of efficient implementation for low level reasoning.

More recently, there have also been attempts in the RoboCup community to use machine learning to acquire higher level reasoning methods. For example, Stone describes the use of reinforcement learning to learn a subset of three against two keep away soccer [?].

Miller and Cliff argue that pursuit and evasion games are the next logical step in the investigation of robust, intelligent, adaptive behavior. The first step is the avoidance of static obstacles and the achievement of static goals [?].

The quality of the learned solution can often not easily be judged, since the optimal solution for a given problem is often not known. One reason for choosing a pursuit and evasion game in this work is the fact that the optimal strategy is known for games in an infinitely sized playing field. The quality of the machine learning methods can thus be compared to the optimal performance possible.

The sizes and kinematics of the game were based on the physical robots available in our lab. Those robots are based on remote controlled cars with a length of 18 cm and a width of 10cm. The maximum turn rate of the pursuer was limited to 15cm and its maximum speed to 20cm/s. The maximum turn rate of the evader was limited to 25cm and its maximum speed to 10cm/s.

Figure 1 shows a small example of a game in progress.

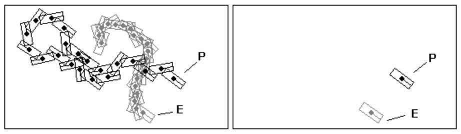

Fig. 1. Example of the Pursuit Evasion Game. The time history is shown on the left, the final state on the right. In this game, the pursuer was successful, since it managed to move within lethal range of the evader.

Section 2 gives a brief introduction into the theory of differential games and leads to the description of the optimal strategy for infinite playing fields in section 3. The machine learning techniques used (Genetic algorithms, k-nearest neighbor, and reinforcement learning) are described briefly in section 4. The empirical evalution of the learned strategies is shown in section 5. The paper concludes with looking at future work (Section 6).

2 Related Work

In this work, we focus our attention on a pursuit and evasion game, the so-called “homicidal chauffeur game” [?].This game takes place on a parking lot. In this game, both pursuer and evader are seen as non-holonomic cars. The pursuer has a higher speed, but a larger turn radius. The evader is more maneuverable, that is, it has a smaller turn radius, but is slower.

The kinematic equations for the pursuer and the evader are shown in Fig. 2. The equations model a unicycle with a limit on the maximum turn cycle.

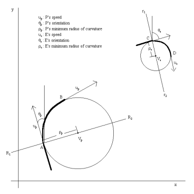
u_p : P's speed
θ_p : P's orientation
ρ_p : P's minimum radius of curvature
u_e : E's speed
θ_e : E's orientation
ρ_e : E's minimum radius of curvature
y
x
r_1
r_2
θ_e
C
D
ρ_e
V_e
u_e
u_p
B
θ_p
u_p
ρ_p
V_p
R_1
R_2
A

Fig. 2. Kinematic Equations in the Pursuit Evasion Game

The control inputs ϕ_p and ϕ_e are normalized from $-1 \ldots +1$. For example, when $\theta_p = -1$, the pursuer executes a full left turn and for $\theta_e = +1$, the evader executes a full right turn.

3 Optimal Strategy

The kinematic equations of the pursuer and evader need to be simplified to develop optimal strategies for the pursuer and evader. We establish a relative frame of reference, which puts the pursuer at the origin and the evader's position relative to the pursuer is determined. Furthermore, the velocity of the evader and the minimum radius of curvature are expressed relative to that of the pursuer. This results in the following coordinate system:

$$\begin{aligned}
x &= (x_e - x_p) cos\theta_p - (y_e - y_p) sin\theta_p \\
y &= (x_e - x_p) sin\theta_p - (y_e - y_p) cos\theta_p \\
\dot{x} &= v_e sin\theta_e - y\phi_p \\
\dot{y} &= v_e cos\theta_e - 1 + x\phi_p
\end{aligned}$$

The game terminates when the evader is within the *lethal range* of the pursuer, where the lethal range is defined as $x^2 + y^2 < \beta$.

Given this representation, the UP, NUP, and BUP can be visualized as shown in Fig. 3.

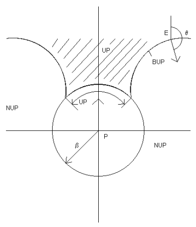

Fig. 3. Boundary of the Usable Portion

The optimal strategy for the pursuer can be derived mathematically, by realizing that it is the action that will center the evader in the UP. The pursuer should turn towards the evader and move randomly if it is already lined up with the evader.

$$\phi_P = \begin{cases} sgns & \text{if s} \neq 0 \\ \text{else move in a random direction} \end{cases}$$

The optimal control for the evader when E is on the BUP (the circle $x^2+y^2 = \beta$, where β is the lethal range) is to steer towards the NUP and normal to the BUP. If such a move is not possible because it would exceed the maximum turn radius, it should steer as close to the desired direction as possible.

$$\phi_E = cos^{-1}\sqrt{1 - v_E^2}$$

If the evader is either in UP or NUP, the closest point to the BUP needs to be found. Then E should move towards the direction normal to the BUP and closer to the NUP.

4 Learned Strategies

The following subsection provides detail of the three selected machine learning techniques that were used in this research: genetic algorithms, k-nearest neighbor learner, and reinforcement learning.

4.1 Genetic Algorithm

To reduce the implementation time needed for this, we used the Samuel system [**?**]. Samuel has been used previously to learn successfully the rules for evading a missile with a fighter jet.

The behavior of an agent was determined by a set of a maximum of 40 rules. Each rule consisted of a condition part and an action part.

We used the following three conditions in the rules:*range*: the distance between the pursuer and evader, *bearing*: the direction from the pursuer to the evader, and *heading*: the direction of the evader relative to the pursuer. Each conditional part of a rule includes values for upper and lower bounds for these three conditions.

The action part of the rules contain one of nine control values (-1, -0.75, -0.50,...,0.00, 0.25,...,+1) corresponding to different steering angles ϕ.

Given a set of rules, a policy needs to be established to select which rules to execute if no rules satisfy the conditions or if more than one rule satisfies all conditions. This problem is solved by associating a rule strength to each rule and by executing the action suggested by the strongest rule. The rule strength is determined by how well the current state matches the conditions of the rule and how successful the rule has been in the past.

If a rule has been selected, all three condition parts are updated by moving the bounds closer to the current conditions. The strength of a rule is updated using the profit sharing plan as described in [**?**].

We used the following four genetic operators to “stir the gene pool:” rule mutation, rule crossover, rule merging, and plan crossover.

4.2 k-Nearest Neighbor Learner

In this work, we used a simple case-based reasoning approach, the 1-nearest neighbor algorithm. The basic idea is that a database of previous situations (so called cases) is maintained by the agent. Anytime the agent is asked to make a decision, the entry in the database that is most similar to the current situation is retrieved and the action that is suggested by this case is executed.

Each state is defined by the same conditions as were used in the genetic algorithm learner: range, bearing, and heading. The similarity measure is the sum over the normalized distances between features as shown below:

$$\text{Sim}(C_i, C_j) = \sum_{k=1}^{3} \frac{|C_{ik} - C_{jk}|}{\text{Max. Dist}_k}$$

During learning the agent would execute random actions. All situations that were encountered during this exploratory phase were recorded. If the game was successful, that is the evader was not caught, or the pursuer was able to catch the evader, it is considered as a possible case candidate to be added to the database.

Case-based maintenance is a problem in most case-based reasoning systems. There are processing and memory constraints which limit the size of the database. In this work, we used a simple scheme to control the size of the database. The maximum number of cases for each suggested action was limited to 100 cases. This limited the maximum size of the database to 900 cases.

After a learning episode, a new case would be added to the database, if it was not already subsumed by an existing case in the database. If the maximum size of the case base for a specific action was exceeded by the addition of this case, a randomly selected case was removed from the case base.

4.3 Reinforcement Learner

We also implemented a standard Q learner. Both learners used the standard Q learning update rule shown below. However, since the goal of the pursuer and evader are different, different reward formulas were used for the two players.

Update:$R(x) = R(x) + \rho * (R(i) + \delta(Q(y) - R(x)))$
Reward Evader:$10 * t$ if E is caught at time t
Reward Pursuer:$R(E) = 10 * (T - t)$ if P catches E at time t

5 Comparison of Different Strategies

This section discusses the results of evaluating and comparing the performance of different strategies. In our evaluation, we used a playing field 540cm by 450cm. Each game consisted of a maximum of 100 steps. We run 1000 games and computed the average evasion success rate as well as the standard deviation in 10 blocks of 100 games. In each game, the pursuer started in the centre of the playing field and the evader started in a random position, at least some distance away from the pursuer.

Table 1 shows the result of these experiments. The entries to a random mover (R), the optimized strategy (O), the genetic learner (G), the k-nearest neighbor learner (K), and the reinforcement learner (Q). $\text{Sum}_{P/E}$ summarize the score for a particular pursuer and evader respectively. As should be expected, the optimal evader E_O is the best evader and so is the optimal pursuer P_O. Remember that the entries in the table correspond to the evasion success rate and thus lower numbers show a better performance of the pursuer.

	P_R	P_O	P_G	P_K	P_Q	SUM$_E$
E_R	62(4.49)	0(0)	61(6.70)	31(5.21)	60(4.84)	214
E_O	71(4.25)	25(3.22)	78(3.78)	21(1.22)	76(5.43)	271
E_G	62(3.86)	8(2.19)	79(2.26)	12(4.02)	74(2.41)	235
E_K	67(4.28)	4(0.63)	67(4.64)	37(3.36)	78(4.39)	253
E_Q	62(4.20)	0/(0.89)	63(3.79)	42(4.23)	65(4.25)	232
SUM$_P$	324	37	348	143	353	

Table 1. Evasion Success for 1000 Games with 100 Steps/Game. The entry in each cell is the evasion success rate and the standard deviation is given in brackets.

6 Conclusion

The evaluation shows that the k-NN learner performed well compared to the other strategies. Its performance (253) was similar to that of the optimal strategy (271). The performance of the k-NN pursuer was worse than that of the optimal strategy, but it was significantly better than that of the other learned strategies.

It is also interesting to note that the genetic algorithm did learn the best evasion strategy against the optimal pursuer of all the learned strategies. Since the all learners were trained against the optimal strategy, it shows that the genetic algorithm did not use enough exploration during the training phase. Although it outperforms the k-NN by 100% against the optimal pursuer, it performs much worse than the k-NN learner against other strategies. The k-NN learner on the other hand uses only randomized moves during training and therefore is able to more general strategies.

Although the genetic learner and the reinforcement learner learned reasonable good strategies for the evasion part of the game, they did perform worse than the randomized mover as pursuers.