Learning in Pursuit-Evasion Differential Games Using Reinforcement Fuzzy Learning

by

Badr Al Faiya, **B.Sc.**

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs

in partial fulfillment of the requirements for the degree of
Master of Applied Science in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

February 2012

The undersigned recommend to
the Faculty of Graduate and Postdoctoral Affairs
acceptance of the thesis

Learning in Pursuit-Evasion Differential Games Using Reinforcement Fuzzy Learning

submitted by

Badr Al Faiya, **B.Sc.**

in partial fulfillment of the requirements for the degree of
Master of Applied Science in Electrical and Computer Engineering

---

Chair, Professor Howard Schwartz,
Department of Systems and Computer Engineering

Thesis Supervisor, Professor Howard Schwartz

Carleton University

February2012

Abstract

In this thesis, Q-learning fuzzy inference system is applied to pursuit-evasion differential games. The suggested technique allows both the evader and the pursuer to learn their optimal strategies simultaneously. Reinforcement learning is used to autonomously tune the input parameters and the fuzzy rules of a fuzzy controller for both the evader and the pursuer. We focus more on the behaviours and the strategies of the trained evader. The evader is trained to find its optimal strategy from the received rewards during the game. The homicidal chauffeur game and the game of two cars are used as examples of the method. The simulation results of the suggested technique demonstrate that the trained evader is able to learn its optimal strategies. Furthermore, the learning speed is investigated when using Q-learning with eligibility traces in pursuit-evasion differential games.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Reinforcement learning is learning complex behaviors through interactions with an unknown environment without supervision or a teacher. In reinforcement learning, an interacting learner or player receives feedback as rewards or punishments from the environment. The player then learns to perform optimally based on the received rewards. Reinforcement learning has recently been used to tune the parameters of a fuzzy controller.

Fuzzy control offers an alternative way to replace conventional control methods especially when the control process is complex to analyze or when the available sources of information are represented uncertainly. In robotic applications, for example, designing an intelligent robot can achieve high levels of complexity, making it difficult to use conventional control efficiently. Unlike conventional control methods, fuzzy control provides a linguistic method to control a system by interpreting a humans

heuristic knowledge. In recent studies, the combination of fuzzy control and reinforcement learning has been used to train mobile robots in pursuit-evasion differential games.

Generally the solution of differential games or the optimal strategies of the players are not always available to the players or are difficult to obtain. Therefore, the players need to learn their optimal strategies by playing the game. Pursuit-evasion differential games are one of the most important and challenging optimization problems. Several solutions of the pursuit-evasion games have been developed such as the homicidal chauffeur game and the game of two cars. The difficulty of the problem is due to the conflicting aims of the players which try to make the best possible decisions considering that the opponent is doing the same. In a pursuit-evasion game, one pursuer tries to capture an evader in minimum time while the evader tries to escape from the pursuer.

In pursuit-evasion games, both the evader and the pursuer need to learn their optimal strategies simultaneously. The complexity of the game increases when designing an evader that can learn to turn sharply and escape from the pursuer. Most of the studies on the pursuit-evasion game in the literature did not train the evader to escape. The investigation of this thesis focuses on training the evader in pursuit-evasion games. This can be considered as a general problem that can be used to solve other applications such as obstacle avoidance, missile avoidance, tactical air combat and other military and security applications. The optimal solutions of the homicidal chauffeur game and the time-optimal control strategy for the game of two cars are available in the literature. Therefore we can compare our learned strategies with the optimal solutions to evaluate the players' learning process. Furthermore, we will

investigate the learning speed in reinforcement learning.

1.2 Literature Review

The pursuit-evasion differential game is one type of differential game which is played in the continues time domain [13]. In a pursuit-evasion game, a pursuer attempts to capture an evader in minimal time, while the evader tries to avoid capture.

Pursuit-evasion games have been studied intensely for several decades because of their importance as mentioned before. The formal theoretical solution concerning optimal strategies and time-optimal control strategies of the homicidal chauffeur game and the game of two cars are given in [13, 17, 19, 20]. In [13], Isaacs presented a condition for the pursuer to succeed in capturing the evader in the homicidal chauffeur game.

In a pursuit-evasion games, the evader and the pursuer need to learn their optimal strategies by interacting with an unknown environment. In recent studies, fuzzy reinforcement learning has been applied to differential games [6, 7, 9, 10] to train the players to find their best control actions. Reinforcement learning is a learning technique that maps situations to actions to maximize rewards [23].

To the best of our knowledge, most of the work done on the pursuit-evasion differential game in the literature did not use reinforcement learning except the work done by our colleagues Desouky and Schwartz [6] and Givigi et al. [9]. In [6] and [9], the fuzzy reinforcement learning was applied to train the pursuer in a pursuit-evasion differential game with a fuzzy Q(λ)-learning presented in [6] and a fuzzy actor-critic method presented in [9].

Givigi et al. [9] used reinforcement fuzzy learning to tune the output parameters of a fuzzy controller in a pursuit-evasion game when a robot is pursuing another robot which is moving along a straight line. Simulation results in [9] shows that reinforcement learning can be used to teach the pursuer to capture the evader. Desouky and Schwartz [6] developed a Q(λ)-learning fuzzy inference system (QLFIS) technique in which Q-learning is combined with eligibility traces to tune the input and the output parameters of a fuzzy controller. Q-learning is one of the common reinforcement learning techniques [23, 26, 27] which estimates the expected rewards received in the future given the current state-action pair. To use Q-learning with continues state and action spaces, it is impractical to discretize the state space and the action space. Instead, one can use a "function approximator" such as fuzzy systems to represent the continuous state space and action space[5, 8, 10, 11].

Desouky and Schwartz [6] have successfully applied QLFIS to different versions of the pursuit-evasion games to train the pursuer. Simulation results in [6] showed that reinforcement learning can be used to teach the pursuer to capture the evader and minimize the time of capture. This approach will be modified and simulated in this thesis to make both the evader and the pursuer learn from playing the game. We will also investigate the learning speed when using eligibility traces.

1.3 Contributions and Thesis Organization

This thesis attempts to cover some of the points that have not been represented in the research so far. There is a lack of studies on how the evader can learn its optimal strategy by playing the game. Moreover, the capture condition for the game has not

been investigated when training the players. We analyze the pursuit-evasion game for both the evader and the pursuer. The evader is trained to learn its optimal strategy from the received rewards during the game. We focus more on the behaviours and the strategies of the evader. We increase the complexity of the game by training the evader to take the appropriate action whenever the pursuer reaches some threshold distance. The trained evader learns to find this distance and to use an extreme strategy to avoid being captured, or maximize the capture time if the capture must happen.

Two pursuit-evasion differential games are used: the homicidal chauffeur game and the game of two cars. We consider the optimal solutions and the capture conditions of the homicidal chauffeur game to evaluate the learning technique. We first show the simulation results of the optimal solution of the game for comparison. We then apply a learning algorithm to the games to make both the evader and the pursuer learn their optimal strategies simultaneously. The learning technique is based on Q(λ)-learning fuzzy inference system. Furthermore, with a greater complexity of the learning process and the game comes the need to consider the learning speed. We will investigate the use of eligibility traces in Q(λ)-learning and show the difference in the convergence speed when using Q-learning only.

The organization of this thesis is as follows:

- Chapter 2 presents the pursuit-evasion differential games. Two pursuit-evasion games are presented: the homicidal chauffeur game and the game of two cars. The formal results of optimal strategies are presented and simulated for both the pursuer and the evader. Isaacs capture condition for the game and computer

simulation results are discussed.

- Chapter 3 introduces fuzzy systems structure and components such as fuzzy IF-THEN rules, fuzzy sets and membership functions. The capability of fuzzy systems from a function approximation point of view is tested and simulated.

- Chapter 4 presents reinforcement learning including Markov decision processes, temporal-difference learning, and eligibility traces. We investigate the use of eligibility traces in Q(λ)-learning and compare the results when applying Q-learning.

- Chapter 5 describes the Q(λ)-learning fuzzy inference system that will be used to train the players in the homicidal chauffeur differential game and the game of two cars. We apply a learning algorithm for both the evader and the pursuer to learn their optimal strategies simultaneously. We then apply Q-learning without the use of eligibility traces to evaluate the results and compare the results with Q(λ)-learning. Computer simulation and results show that the evader can learn to maximize the capture time and escape from the pursuer. We also show that the use of eligibility traces did not speed up the learning process significantly.

- Chapter 6 concludes this thesis by stating the main contributions and points out the future research directions.

Chapter 2

Differential Games

2.1 Pursuit-evasion Differential Game

Differential games (DG) are a family of dynamic, continuous time games. The pursuit-evasion differential game is one type of differential game. It was originally presented by Isaacs in 1954. Isaacs defined the "Homicidal Chauffeur Problem" in a Rand technical report [12]. The game has been extended to more general pursuit-evasion problems/games [13]. A pursuer or a group of pursuers attempt to capture one or a group of evaders in minimal time while the evaders try to avoid being captured.

In a pursuit-evasion game, a pursuer attempts to capture an evader in minimal time, while the evader tries to avoid capture. The objectives of the pursuer and the evader are opposite to each other. Therefore, the pursuit-evasion game is a two-player zero-sum game. The difficulty of the problem is due to the conflicting aims of the players who try to make the best possible decisions considering that their opponents are doing the same.

We consider a two-player differential game. The players' system dynamics are given by

$$\dot{\bar{x}}(s) = f(\bar{x}(s), \bar{\varphi}(s), \bar{\psi}(s), s), \quad \bar{x}(t_0) = \bar{x}_0 \tag{2.1}$$

where $\bar{x}(s) \in \mathbf{R}^m$ is the state vector of dimension m, function $f(\cdot)$ determines the dynamics of the system, $\bar{\varphi}$ and $\bar{\psi}$ are the strategies played by player 1 and player 2 respectively. The payoff, represented as $P(\bar{\varphi}, \bar{\psi})$, is given in the form

$$P(\bar{\varphi}, \bar{\psi}) = q(t_f, \bar{x}(t_f)) + \int_{t_0}^{t_f} g(\bar{x}(s), \bar{\varphi}, \bar{\psi}, s) ds \tag{2.2}$$

where t_f is the terminal time (or the first time the states $\bar{x}(t)$ intersect a given final condition), $q(\cdot)$ is the payoff at the terminal time, $g(\cdot)$ is the integral payoff and functions $q(\cdot)$ and $g(\cdot)$ are chosen in order to achieve an objective. For a two-player zero-sum game, player 1 uses strategy $\bar{\varphi}$ to maximize the payoff $P(\cdot)$, whereas player 2 uses strategy $\bar{\psi}$ to minimize it. In a two-player zero-sum game, there exist player's optimal strategies $\bar{\varphi}^*$ and $\bar{\psi}^*$ such that

$$P(\bar{\varphi}^*, \bar{\psi}) \leq P(\bar{\varphi}^*, \bar{\psi}^*) \leq P(\bar{\varphi}, \bar{\psi}^*), \quad \forall\, \bar{\varphi},\, \bar{\psi} \tag{2.3}$$

where $P(\bar{\varphi}^*, \bar{\psi}^*)$ is defined as the value function of the game.

In our model, the game terminates when the evader is within the lethal range of the pursuer (capture and terminal time), or when the time exceeds one minute (escape). Players evaluate the current state and then select their next actions. The players' strategies are not shared and therefore each player has no knowledge of the other player's next selected action. We assume that the environment is obstacles-free.

The players are wheeled mobile robots.

The existence of optimal strategies in Eq. (2.3) in the pursuit-evasion differential game is determined by Isaacs condition [1]. The formal results concerning optimal strategies for pursuit-evasion differential games are given in [13, 17, 19, 20]. The homicidal chauffeur game and Isaacs condition for the game is discussed in Sect. 2.1.1. In section Sect. 2.1.2 the game of two cars is presented as a second example of pursuit-evasion differential games. Simulation results of the games are shown in Sect. 2.2.

2.1.1 The Homicidal Chauffeur Problem

The homicidal chauffeur problem is one of the most well-known model problems in the theory of differential games. The homicidal chauffeur problem is presented as the main example of a pursuit-evasion differential game. In our model, a homicidal chauffeur game is played by autonomous robots. The chauffeur (the pursuer P) is a car-like mobile robot and the pedestrian (the evader E) is a point that can move in any direction instantaneously. In Isaacs' homicidal chauffeur differential game, a pursuer aims to minimize the capture time of an evader. The evader's objective is to maximize the capture time and avoid capture.

We assume that the players move at a constant forward speed w_i, $i \in p, e$. The pursuer's speed is greater than the evader's speed, but the evader can move in any direction instantaneously. The motion of the pursuer is constrained by its maximum steering angle such that

$$-(u_{max})_p \leq u_p \leq (u_{max})_p \tag{2.4}$$

where $(u_{max})_p$ is the maximum steering angle.

The maximum steering angle results in a minimum turning radius R_p defined by

$$R_p = \frac{L_p}{\tan(u_{max})} \tag{2.5}$$

where L_p is the pursuer's wheelbase.

The dynamic equations for the pursuer P and the evader E are [13]

$$\begin{aligned} \dot{x}_p &= w_p \cos(\theta_p) \\ \dot{y}_p &= w_p \sin(\theta_p) \\ \dot{\theta}_p &= \frac{w_p}{R_p} u_p \\ \dot{x}_e &= w_e \cos(u_e) \\ \dot{y}_e &= w_e \sin(u_e) \end{aligned} \tag{2.6}$$

where (x, y), w, and θ denote the position, the velocity, and the orientation respectively as shown in Figure 2.1.

The angle difference ϕ between the pursuer and the evader is given as

$$\phi = \tan^{-1}(\frac{y_e - y_p}{x_e - x_p}) - \theta_p \tag{2.7}$$

The relative distance between pursuer and evader is found as

$$d = \sqrt{(x_e - x_p)^2 + (y_e - y_p)^2} \tag{2.8}$$

The capture occurs when the distance $d \leq \ell$ where ℓ is capture radius.

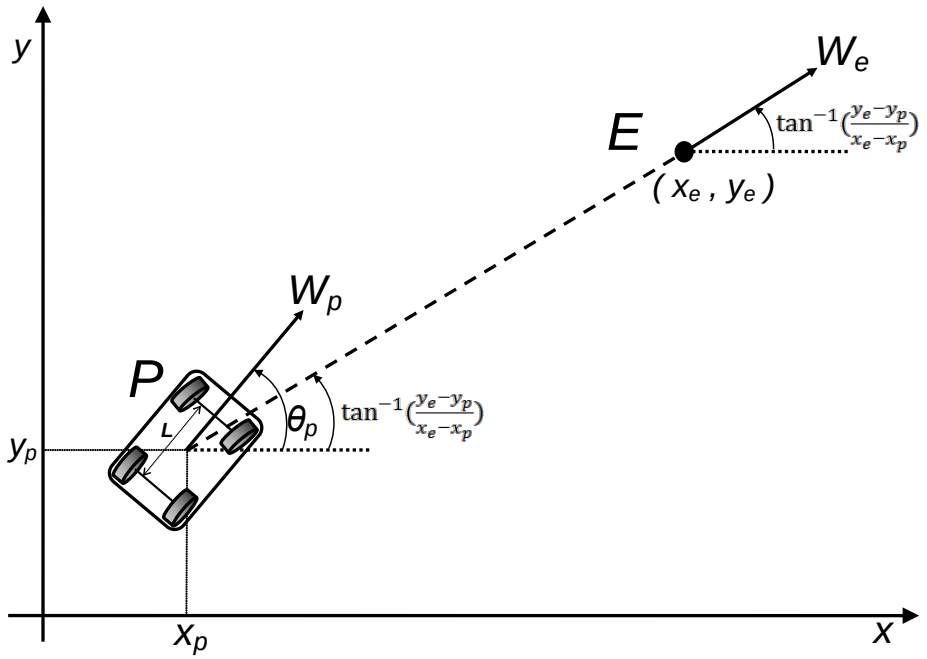


Figure 2.1: Homicidal chauffeur problem

In [13, p. 232-237], Isaacs presented a condition for the pursuer to succeed in capturing the evader. Assuming that the pursuer's speed is greater than the evader's speed, the capture condition is given as

$$l/R_p > \sqrt{1-\gamma^2} + \sin^{-1}\gamma - 1 \tag{2.9}$$

where ℓ/R_p is the ratio of the radius of capture to the minimum turning radius of the pursuer, and $\gamma = w_e/w_p < 1$ is the ratio of the evader's speed to the pursuer's speed. If inequality (2.9) is reversed, E escapes from P indefinitely.

Based on the capture condition in Eq. (2.9) and Isaacs' solution of the problem, the evader's optimal strategy can be obtained by solving the following two problems [13, 19, 20]:

1- When the evader is far enough from the pursuer, the evader's control strategy is to maximize the distance between the evader and the pursuer as follows

$$u_e = \tan^{-1} \frac{y_e - y_p}{x_e - x_p} \tag{2.10}$$

2- When the pursuer approached the evader such that $d \leq R_p$, the evader adopts a second control strategy to avoid capture. The pursuer cannot turn more than a minimum turning radius R_p. Accordingly, the evader will make a sharp turn, *normal* to its direction, and enter the pursuer's non-holonomic constraint region (or called the unreachable region). As shown in Figure 2.2, a non-holonomic wheeled player is constrained to move along the path with a bounded curvature denoted as two dotted circles which is the pursuer's minimum turning radius R given in Eq. (2.5). For the evader to use the advantage of its higher maneuverability, the second control strategy is given as

$$u_{e_{extreme}} = \theta_e \pm \pi/2 \tag{2.11}$$

The pursuer's optimal control strategy is to minimize the distance and capture the evader in minimum time. The pursuer controls its steering angle as follows [1, 4, 17]

$$u_p = \tan^{-1}(\frac{y_e - y_p}{x_e - x_p}) - \theta_p \tag{2.12}$$

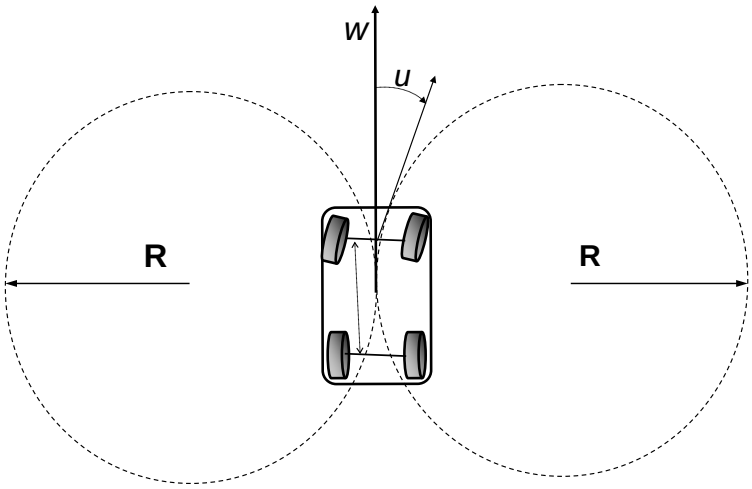


Figure 2.2: The vehicle cannot turn into the circular region defined by its minimum turning radius R.

2.1.2 The Game of Two Cars

The game of two cars includes two car-like mobile robots. Unlike the homicidal chauffeur problem, the evader acts like a car and not like a pedestrian which moves in any direction. The pursuer is faster than the evader but the evader can make sharper turns than the pursuer (more maneuverable) such that

$$\begin{aligned} v_p &> v_e \\ u_{p_{max}} &< u_{e_{max}} \end{aligned} \tag{2.13}$$

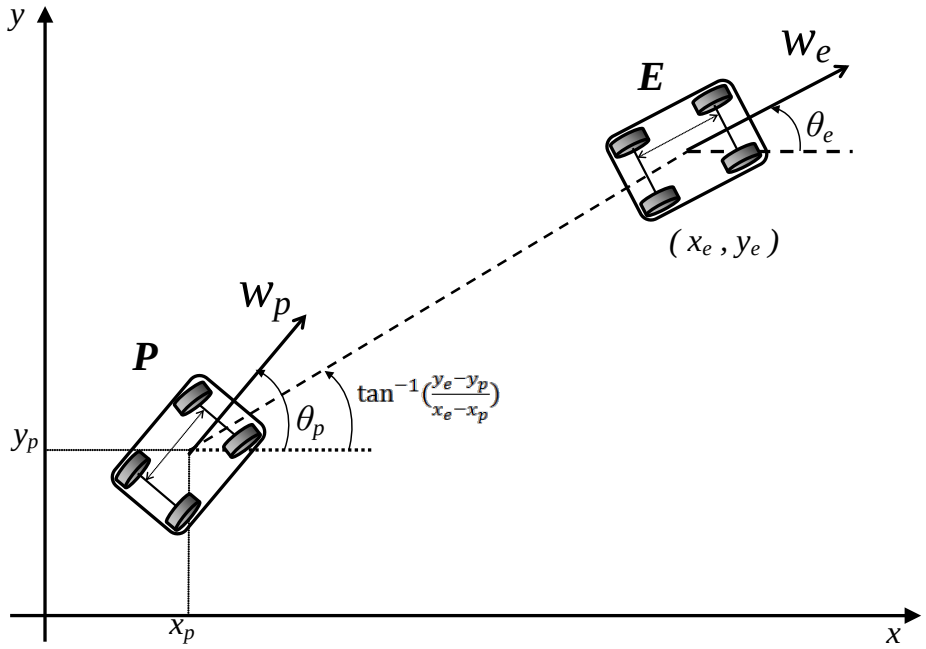


Figure 2.3: The game of two cars

The dynamic equations for the pursuer and the evader are [13, 16]:

$$\begin{aligned} \dot{x}_i &= w_i \cos(\theta_i) \\ \dot{y}_i &= w_i \sin(\theta_i) \\ \dot{\theta}_i &= \frac{w_i}{L_i} \tan(u_i) \end{aligned} \tag{2.14}$$

where $i \in p, e$, and (x_i, y_i), w_i, θ_i, and L_i denote the position, the velocity, the orientation, and the wheelbase of the robot respectively. The parameters and the model are shown in Figure 2.3.

The angle difference ϕ_i between the pursuer and the evader is given as

$$\phi_i = \tan^{-1}(\frac{y_e - y_p}{x_e - x_p}) - \theta_i \tag{2.15}$$

In [17] a time-optimal pursuit-evasion strategy is proposed and derived based on the consideration of the worst-case motion of the evader. The pursuer's strategy is to minimize the distance and the time of capture by controlling its steering angle u_p by Eq. (2.12).

The evader uses two different control strategies:

1- When the evader is far enough from the pursuer, the evader's control strategy is to maximize the distance by controlling its steering angle as

$$u_e = \tan^{-1}(\frac{y_e - y_p}{x_e - x_p}) - \theta_e \tag{2.16}$$

2- When the distance between the pursuer and the evader is less than R_p, the evader adopts a second control strategy to avoid capture. The evader will make a sharp turn $u_{e_{max}}$ to enter the pursuer's non-holonomic constraint region. The evader's second control strategy is given as

$$u_{e_{max}} = \theta_e \pm \pi \tag{2.17}$$

If the capture condition is not satisfied in equation Eq. (2.9), the pursuer will be forced to reroute its path and continue pursuing.

In [17], a predictor is applied for the pursuer to pursue the unpredictable evader's behavior when the evader attempts to intrude into the non-holonomic constraint region of the pursuer. We will not use the predictor in our simulation because we want our evader to learn the optimal strategy itself and avoid capture when the

escape condition is satisfied. Moreover, later in Chap. 5 we will evaluate the learning technique with the theoretical results in this chapter given the capture condition in Eq. (2.9).

2.2 Simulation Results

In this section we play pursuit-evasion differential games for autonomous robots. We use the theoretical solution of the homicidal chauffeur game and and the time-optimal control of the game of two cars described in Sect. 2.1. The capture condition is tested.

We assume that the pursuer is twice as fast as the evader, $w_p = 2m/s$ and $w_e = 1m/s$. The initial position of the pursuer is $(x_p, y_p) = (0, 0)$ and the evader is $(x_e, y_e) = (10, 7)$. The initial orientation of the pursuer is $\theta = 0rad$.

The wheelbase of the pursuer is $L = 0.3m$. The capture radius is $\ell = 0.15m$ which is half the wheelbase of the pursuer $\ell = \frac{L_p}{2}$. The game terminates when the pursuer can touch (or intercept) the evader such that the distance $d \leq 0.15m$, or when the time exceeds 60 sec (escape). We use the kinematic equations of the pursuer and the evader given in Eq. (2.6) and Eq. (2.14). The parameters of the capture condition will be set for two cases as follows.

The homicidal chauffeur game

- Case 1: we first set the maximum steering angle of the pursuer to $-0.5rad \leq u_{p_{max}} \leq 0.5rad$ with $R_p = 0.5491m$. Given the stated parameters of the system, Isaacs capture condition in Eq. (2.9) is not satisfied which means that there exist a strategy for the evader to avoid capture. It can be shown in Figure 2.4(b) that the evader is guaranteed to escape from the

pursuer by making sharp turns when the distance between the two robots $d \leq R_p$, where E_0 and P_0 are the initial positions of the evader and pursuer respectively.

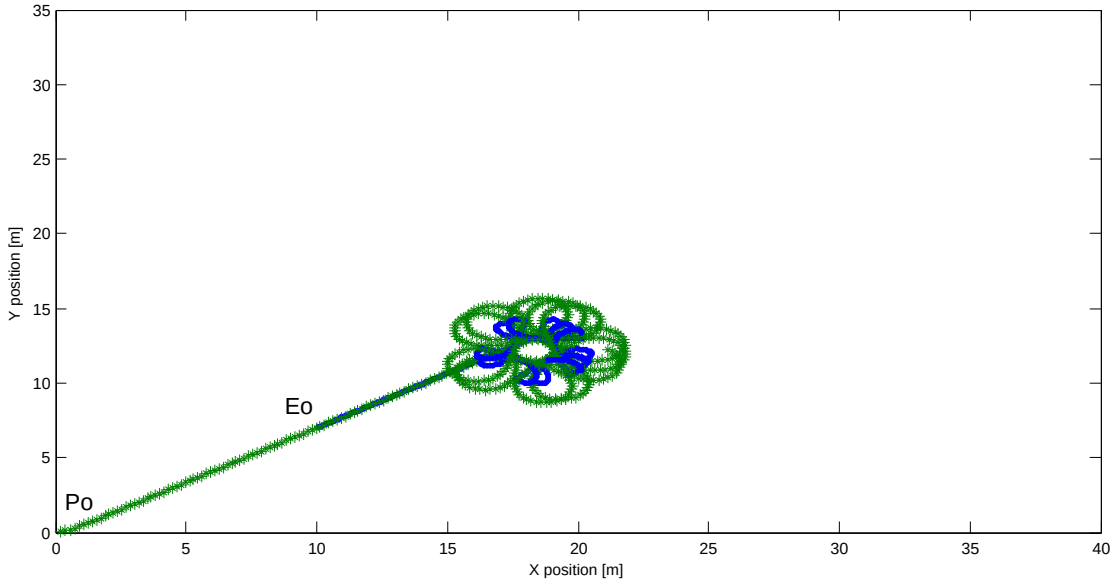
- Case 2: the steering angle of the pursuer is now increased to $-0.7rad \leq u_{p_{max}} \leq 0.7rad$. In this case, the capture is achieved by meeting the capture conditions in Eq. (2.9). Figure 2.4(b) shows that the pursuer can capture the evader with capture time $T_c = 11.90sec$.

The Game of Two Cars

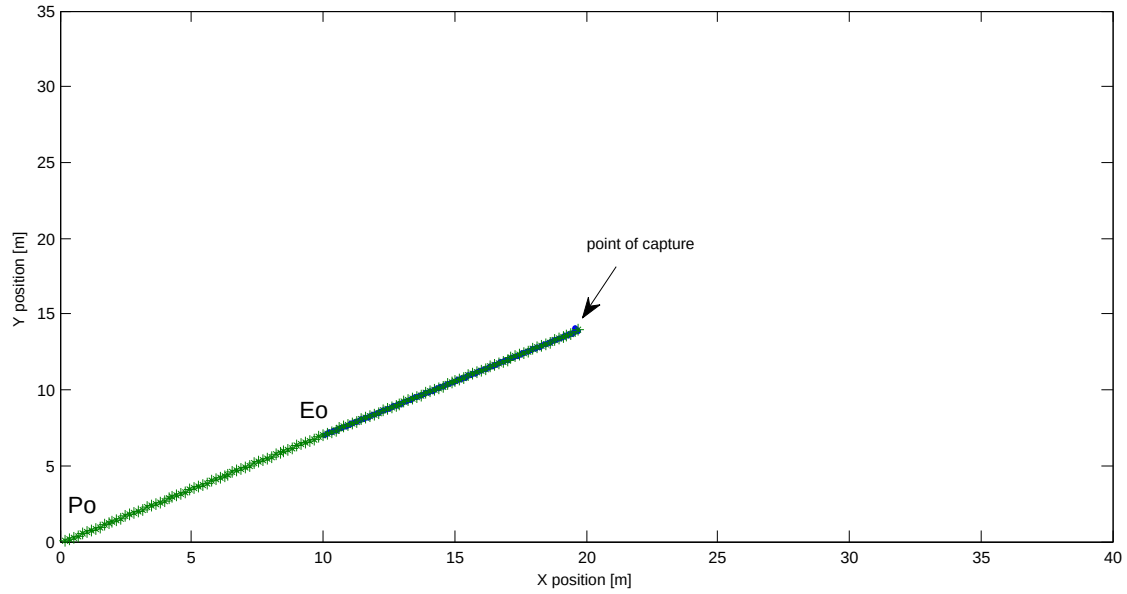
The game of two cars scenario described in Sect. 2.1.2 is simulated for the two cases. Unlike the homicidal chauffeur problem, the evader acts like a car. The maximum steering angle of the evader is $-1\,rad \leq u_{e_{max}} \leq 1\,rad$.

- Case 1: the maximum steering angle of the pursuer is set to $-0.5rad \leq u_{p_{max}} \leq 0.5rad$ with $R_p = 0.5491m$. Similarly, in this case simulation results show that here exist a strategy for the evader to avoid capture. It can be shown in Figure 2.5(a) that the evader escapes from the pursuer by making sharp turns when $d \leq R_p$. The path of the player is different from the homicidal chauffeur because the evader in this game acts like a car.

- Case 2: the steering angle of the pursuer is increased to $-0.7rad \leq u_{p_{max}} \leq 0.7rad$. The capture is again achieved. Figure 2.5(b) shows that the pursuer can capture the evader with capture time $T_c = 12.10sec$.

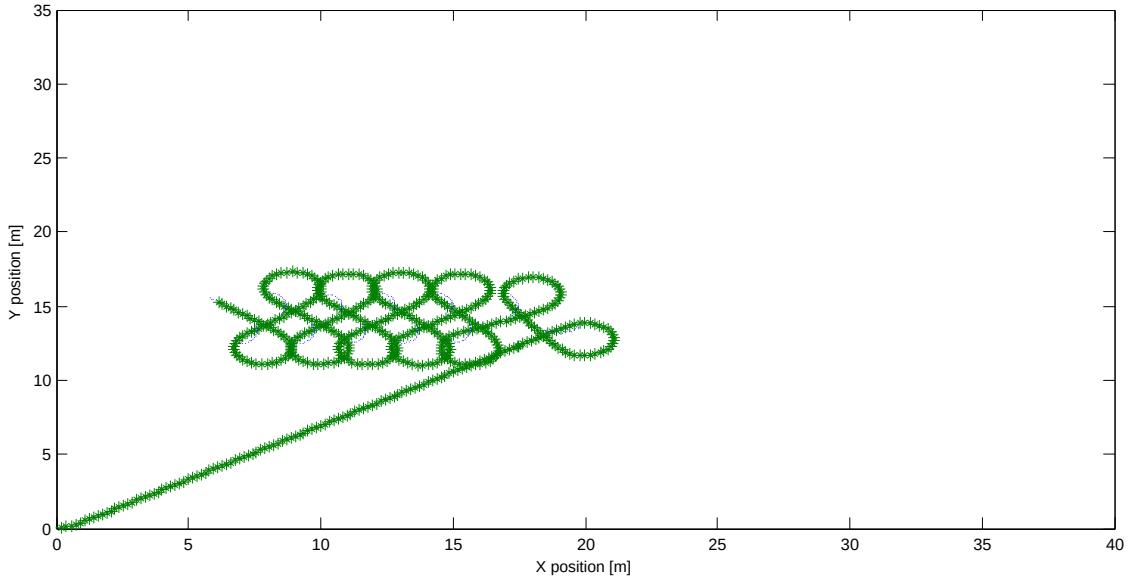


(a) The evader escapes when $u_{p_{max}} = 0.5rad.$

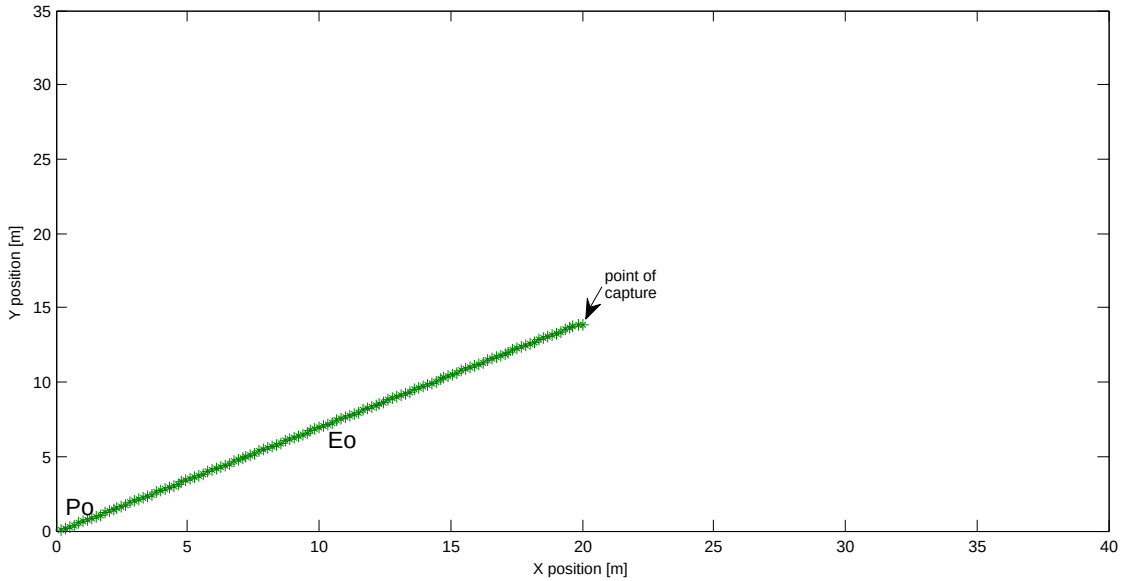


(b) The pursuer captures the evader when $u_{p_{max}} = 0.7rad.$

Figure 2.4: The robots play the optimal strategies in the homicidal chauffeur game for the two cases.



(a) The evader escapes when $u_{p_{max}} = 0.5rad.$



(b) The pursuer captures the evader when $u_{p_{max}} = 0.7rad.$

Figure 2.5: The robots play the optimal strategies in the game of two cars for the two cases.

Chapter 3

Fuzzy Systems

Fuzzy systems have been used in a wide variety of applications in engineering, science, business, medicine, psychology, and other fields [21]. In engineering, for example, some potential application areas include [21]:

- Aircraft/spacecraft: flight control, engine control, avionic systems, failure diagnosis, navigation, and satellite attitude control.
- Robotics: position control and path planning.
- Autonomous vehicles: ground and underwater.
- Automated highway systems: automatic steering, braking, and throttle control for vehicles.

3.1 Fuzzy sets and Fuzzy rules

Fuzzy sets use linguistic labels arranged by membership functions (MF) to perform numerical computation [14]. Fuzzy set theory provides a way of dealing with information linguistically as an alternative to calculus. Fuzzy sets use linguistic labels arranged by membership functions (MF) to perform numerical computation [14].

The universe of discourse X is defined as a collection of elements x which have the same characteristics. A fuzzy set A in X can be denoted by [14]

$$A = \{(x, \mu_A(x)) | x \in X\} \tag{3.1}$$

where $\mu_A(x)$ is the *membership function* (MF) for the fuzzy set A. The membership functions can have values between 0 and 1. The MF maps the elements of the universe of discourse to membership degrees between 0 and 1. If $\mu_A(x)$ only has values of 0 or 1, the fuzzy set A is called a crisp or a classical set.

The interpretations of set operations, such as union and intersection, are complicated in fuzzy set theory due to the graded property of MF's. Zadeh [28] proposed the following definitions for union and intersection operations:

Union $\mu_{A \cup B}(x) = max[\mu_A(x), \mu_B(x)]$

Intersection $\mu_{A \cap B}(x) = min[\mu_A(x), \mu_B(x)]$

where A and B are fuzzy sets.

Membership functions are normally described using graphics. Figure 3.1 shows

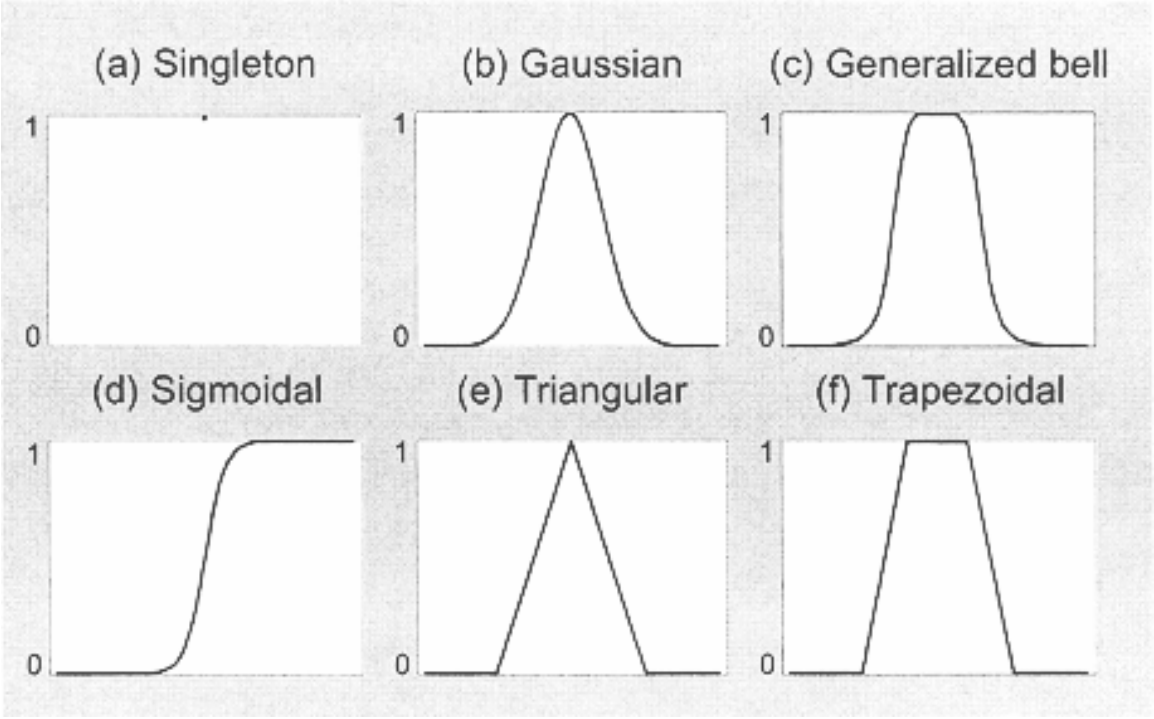


Figure 3.1: Examples of membership functions

various types of membership functions commonly used in fuzzy set theory. The Gaussian membership function for example in Figure 3.1 (b) is given as

$$\mu_A(x) = \exp\left(-(\frac{x-m}{\sigma})^2\right) \tag{3.2}$$

where the Gaussian MF parameters are the mean m and the standard deviation σ.

The trapezoidal membership function has four parameters as shown in figure

3.1 (f). The trapezoidal MF is defined as

$$\mu(x) = \begin{cases} 0 & : \quad x < \alpha \\ \dfrac{x-\alpha}{\beta-\alpha} & : \quad \alpha \leq x < \beta \\ 1 & : \quad \beta \leq x \leq \gamma \\ \dfrac{x-\gamma}{\lambda-\gamma} & : \quad \gamma < x \leq \lambda \\ 0 & : \quad x > \lambda \end{cases} \tag{3.3}$$

Fuzzy IF-THEN rules can effectively model human expertise in an environment of uncertainty and imprecision [14]. Fuzzy IF-THEN rules are defined as

$$\Re_l : if\ x\ is\ A\ then\ y\ is\ B \tag{3.4}$$

where x, y are called *fuzzy* or *linguistic variables.* The sets A and B are fuzzy sets defined in X, Y. "x is A" is called the *antecedent* or *premise*, "y is B" is called the *consequence* or *conclusion.*

The fuzzy IF-THEN rules used in a Takagi – Sugeno (TS) fuzzy system give a mapping from the input fuzzy sets to a linear function in the output [24, 25]. The rules are in the following form

$$\Re_l : IF\ x_1\ is\ A_1^l\ AND\ x_2\ is\ A_2^l\ AND\ ...\ AND\ x_j\ is\ A_j^l\ THEN\ f_l = K_0^l + \ ... \ + K_j^l x_j \tag{3.5}$$

where f_l is the output function of rule l and K_n^l is the consequent parameter.

When f_l is a constant, then we have a zero-order TS fuzzy model [14]. Let K^l be

the constant for the output f_l. The number of rules is determined by the number of inputs and their corresponding MF's. Given 2 inputs and 3 MF's for each input, we need to construct $3^2 = 9$ rules. The rules are given as

$$\Re_1 : IF\ x_1\ is\ A_1\ AND\ x_2\ is\ A_4\ THEN\ f_1 = K^1$$

$$\Re_2 : IF\ x_1\ is\ A_1\ AND\ x_2\ is\ A_5\ THEN\ f_2 = K^2$$

$$\Re_3 : IF\ x_1\ is\ A_1\ AND\ x_2\ is\ A_6\ THEN\ f_3 = K^3$$

$$\Re_4 : IF\ x_1\ is\ A_2\ AND\ x_2\ is\ A_4\ THEN\ f_4 = K^4$$

$$\Re_5 : IF\ x_1\ is\ A_2\ AND\ x_2\ is\ A_5\ THEN\ f_5 = K^5$$

$$\Re_6 : IF\ x_1\ is\ A_2\ AND\ x_2\ is\ A_6\ THEN\ f_6 = K^6$$

$$\Re_7 : IF\ x_1\ is\ A_3\ AND\ x_2\ is\ A_4\ THEN\ f_7 = K^7$$

$$\Re_8 : IF\ x_1\ is\ A_3\ AND\ x_2\ is\ A_5\ THEN\ f_8 = K^8$$

$$\Re_9 : IF\ x_1\ is\ A_3\ AND\ x_2\ is\ A_6\ THEN\ f_9 = K^9$$

Another format for constructing the fuzzy rules is the tabular format as shown in table 3.1.

Table 3.1: Tabular format.

x_1 \ x_2	A_4	A_5	A_6
A_1	K^1	K^2	K^3
A_2	K^4	K^5	K^6
A_3	K^7	K^8	K^9

3.2 Fuzzy Inference Engine

A fuzzy inference engine is used to combine fuzzy IF-THEN rules in the fuzzy rule base into a mapping from a fuzzy set A' in X to a fuzzy set B' in Y. One of the commonly used fuzzy inference engines is called the product inference engine. The product inference engine is also used in our work in later chapters. In this section, the structure of the product inference engine is presented and explained.

We first provide the two operations on fuzzy sets: intersection and union. Assume we have two fuzzy sets A and B defined in the same universe of discourse U, the intersection of these two fuzzy sets is a fuzzy set whose membership function is

$$\mu_{A\cap B}(x) = T(\mu_A(x), \mu_B(x)) = \mu_A(x) \star \mu_B(x) \tag{3.6}$$

where $\star$ is defined as a t-norm operator. Two commonly used t-norm operators are

$$\text{Minimum} : T_{min}(a, b) = \min(a, b) \tag{3.7}$$

$$\text{Algebraic product} : T_{ap}(a, b) = ab \ . \tag{3.8}$$

The union of two fuzzy sets A and B is a fuzzy set whose membership function is given by

$$\mu_{A\cup B}(x) = S(\mu_A(x), \mu_B(x)) = \mu_A(x) \dot{+} \mu_B(x) \tag{3.9}$$

where $\dot{+}$ is denoted as an s-norm operator. Two commonly used s-norm operators

are

$$\text{Maximum} : S_{max}(a,b) = \max(a,b) \tag{3.10}$$

$$\text{Algebraic sum} : S_{ap}(a,b) = a + b - ab\ . \tag{3.11}$$

In the product inference engine, the algebraic product is used for all the t-norm operators and max is used for all the s-norm operators.

To interpret the IF-THEN operation, one can use Mamdani implication. In the Mamdani implication, a fuzzy if-then rule can be considered as a binary fuzzy relation give by

$$\mu_R(x,y) = \mu_{A\times B}(x,y) = \mu_{A\to B}(x,y) = \mu_A(x) \star \mu_B(y) \tag{3.12}$$

where $A \to B$ is used to interpret the fuzzy relation. If Eq. (3.8) is used as the t-norm operator $\star$ in (3.12), then (3.12) is called the Mamdani's product implication. In fuzzy logic, the generalized modus ponens is defined as

premise 1(rule):	if x is A then y is B
premise 2(fact):	x is A'
conclusion:	y is B'

Based on the generalized modus ponens, the fuzzy set B' is inferred as

$$\mu_{B'}(y) = \sup_{x\in X} T\big[\mu_{A'}(x), \mu_{A\to B}(x,y)\big] \tag{3.13}$$

where $T[\cdot]$ denotes the t-norm operator and sup denotes the greatest element in the set.

In the individual rule-based inference, each fuzzy IF-THEN rule generates an

individual output fuzzy set and the whole output of the fuzzy inference engine is the combination of all the individual output fuzzy sets. In the product inference engine, we combine the individual output fuzzy sets by union.

Overall, the product inference engine includes the following three parts:

1. algebraic product for all the t-norm operators and max for all the s-norm operators;

2. Mamdani's product implication;

3. individual-rule based inference with union combination.

Based on the above structure of the product inference engine, equation (3.13) becomes

$$\begin{aligned} \mu_{B'}(y) &= \max_{l=1}^{M} \mu_{B'_l}(y) \\ &= \max_{l=1}^{M} \big[\sup_{\mathbf{x}\in X}(\mu_{A'}(\mathbf{x}) \prod_{j=1}^{n} \mu_{A^l_j}(x_j)\mu_{B^l}(y))\big] \end{aligned} \tag{3.14}$$

We take an example here. Assume that we have two fuzzy IF-THEN rules with two antecedents for each rule such that:

$$\begin{array}{ll} \text{premise 1 (rule 1):} & \text{if } x_1 \text{ is } A^1_1 \text{ and } x_2 \text{ is } A^1_2 \text{ then } y \text{ is } B^1 \\ \text{premise 2 (rule 2):} & \text{if } x_1 \text{ is } A^2_1 \text{ and } x_2 \text{ is } A^2_2 \text{ then } y \text{ is } B^2 \\ \text{premise 3 (fact):} & x_1 \text{ is } A'_1 \text{ and } x_2 \text{ is } A'_2 \\ \text{conclusion:} & y \text{ is } B' \end{array} \tag{3.15}$$

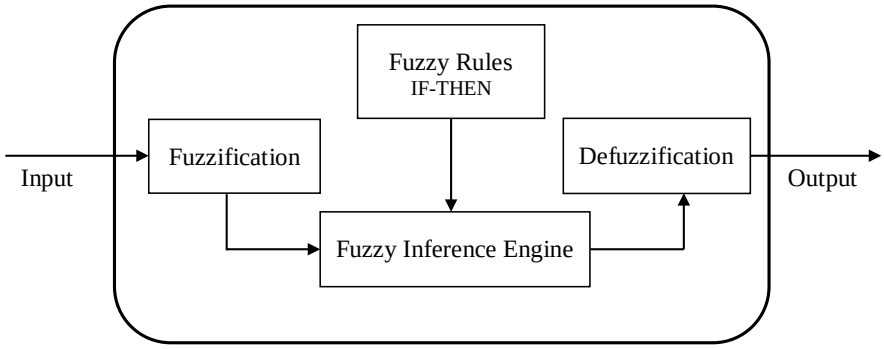


Figure 3.2: Fuzzy System components

Then the output of the product inference engine for (3.15) becomes

$$\mu_{B'}(y) = \max_{l=1}^{2} \max_{x_1,x_2} [\mu_{A'_1}(x_1)\mu_{A'_2}(x_2) \prod_{j=1}^{2} \mu_{A_i^l}(x_j)\mu_{B^l}(y)] \ . \tag{3.16}$$

3.3 Fuzzifier and Defuzzifier

3.3.1 Fuzzifier

Figure 3.2 shows the fuzzy system structure. The first block in the fuzzy system is the fuzzifier. The fuzzifier converts each input which is a precise quantity to degrees of membership in a membership function [15]. The fuzzification block matches the input values with the conditions of the rules. The fuzzification determines how well the condition of each rule matches that particular input. There is a degree of membership for each linguistic term that applies to that input variable.

3.3.2 Defuzzifier

Defuzzification is the conversion of a fuzzy quantity into a precise quantity. Many defuzzification methods have been proposed in the literature in recent years. The weighted average defuzzification method is the most frequently used in fuzzy applications since it is one of the most computationally efficient methods [22]. The weighted average defuzzification method is expressed as

$$f = \frac{\sum_{l=1}^{M} \left(\prod_{j=1}^{J} \mu^{A_j^l}(x_j) \right) f_l}{\sum_{l=1}^{M} \left(\prod_{j=1}^{J} \mu^{A_j^l}(x_j) \right)} \tag{3.17}$$

where J is number of inputs and M is number of rules.

3.4 Fuzzy Systems and Examples

Fuzzy systems are also known as fuzzy inference systems (FIS), or fuzzy controllers when used as controllers. Takagi–Sugeno (TS) fuzzy systems and Mamdani fuzzy systems are commonly used in fuzzy applications. We want to investigate how well fuzzy systems can approximate a given system. The following theorem from [25] is known as the "Universal Approximation Theorem." We will then give an example to show the capability of fuzzy systems from a function approximation point of view.

Theorem 3.4.1 *For any given real continuous function $g(\boldsymbol{x})$ on a compact set $U \subset R^n$ and arbitrary $\epsilon > 0$ with Gaussian MF's, there exists a fuzzy logic system $f(\boldsymbol{x})$ in*

the form of 3.17 such that

$$\sup_{\boldsymbol{x}\in U} |f(\boldsymbol{x}) - g(\boldsymbol{x})| < \epsilon \ . \tag{3.18}$$

The proof is shown on [25, p. 124-126].

The next example introduced in [18] shows the capability of fuzzy systems from a function approximation point of view. It shows that fuzzy systems are good approximators on a given nonlinear system.

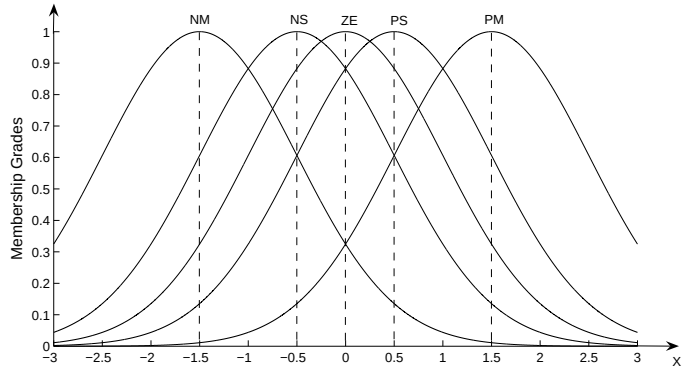
Example 2.1 Consider a first order nonlinear system. The dynamic equation of the system is given by [25]

$$\dot{x}(t) = \frac{1 - e^{-x(t)}}{1 + e^{-x(t)}} + u(t) = f(x) + u(t) \ . \tag{3.19}$$

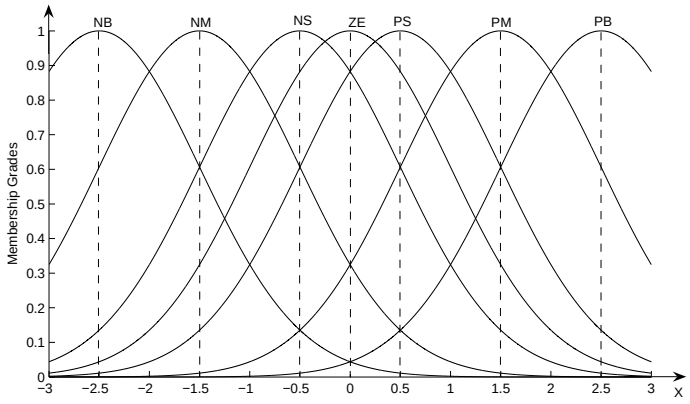
We define five fuzzy sets over the interval $[-3, 3]$: negative medium(NM), negative small(NS), zero(ZE), positive small(PS) and positive medium(PM). The membership functions are given by

$$\begin{aligned} \mu_{NM}(x) &= \exp(-(x+1.5)^2) \\ \mu_{NS}(x) &= \exp(-(x+0.5)^2) \\ \mu_{ZE}(x) &= \exp(-x^2) \\ \mu_{PS}(x) &= \exp(-(x-0.5)^2) \\ \mu_{PM}(x) &= \exp(-(x-1.5)^2) \ . \end{aligned} \tag{3.20}$$

Figure 3.3(a) shows the five membership functions.



(a) The membership functions of 5 fuzzy sets.



(b) The membership functions of 7 fuzzy sets.

Figure 3.3: The membership functions.

To estimate the dynamics of the system, the following linguistic descriptions (fuzzy if-then rules) are given to the designer:

$$\begin{aligned}
R^1: &\quad \text{if } x \text{ is near } -1.5 \text{ then } f(x) \text{ is near } -0.6 \\
R^2: &\quad \text{if } x \text{ is near } -0.5 \text{ then } f(x) \text{ is near } -0.2 \\
R^3: &\quad \text{if } x \text{ is near } 0 \text{ then } f(x) \text{ is near } 0 \\
R^4: &\quad \text{if } x \text{ is near } 0.5 \text{ then } f(x) \text{ is near } 0.2 \\
R^5: &\quad \text{if } x \text{ is near } 1.5 \text{ then } f(x) \text{ is near } 0.6
\end{aligned} \tag{3.21}$$

Set the conclusions $y^1 = -0.6$, $y^2 = -0.2$, $y^3 = 0$, $y^4 = 0.2$ and $y^5 = 0.6$ as in 3.4. Since there is only one antecedent for every fuzzy if-then rule, rewrite 3.17 as

$$\begin{aligned}
\hat{f}(x) &= \frac{\sum_{l=1}^{5} y^l[\mu_{A^l}(x)]}{\sum_{l=1}^{5}[\mu_{A^l}(x)]} \\
&= \frac{-0.6\mu_{NM}(x) - 0.2\mu_{NS}(x) + 0.2\mu_{PS}(x) + 0.6\mu_{PM}(x)}{\mu_{NM}(x) + \mu_{NS}(x) + \mu_{ZE}(x) + \mu_{PS}(x) + \mu_{PM}(x)} \\
&= \frac{-0.6e^{-(x+1.5)^2} - 0.2e^{-(x+0.5)^2} + 0.2e^{-(x-0.5)^2} + 0.6e^{-(x-1.5)^2}}{e^{-(x+1.5)^2} + e^{-(x+0.5)^2} + e^{-x^2} + e^{-(x-0.5)^2} + e^{-(x-1.5)^2}}
\end{aligned} \tag{3.22}$$

Figure 3.4(a) shows the estimation $\hat{f}(x)$ (dot line). Fig. 3.4(b) shows the estimation error (dash line)$|f(x) - \hat{f}(x)| < \epsilon = 0.35$ over the interval $[-3, 3]$.

To make ϵ smaller, we need more specific membership functions and linguistic descriptions in the fuzzy system. We define 7 fuzzy sets over the interval $[-3, 3]$: negative big(NB), negative medium(NM), negative small(NS), zero(ZE), positive small(PS), positive medium(PM) and positive big(PB). The membership functions of the fuzzy

sets NB and PB are defined as $\mu_{NB}(x) = \exp(-(x+2.5)^2)$, $\mu_{PB}(x) = \exp(-(x-2.5)^2)$. The membership functions of fuzzy sets NM, NS, ZE, PS and PM are the same as in 3.20. Figure 3.3(b) illustrates the membership functions of the 7 fuzzy sets. The fuzzy if-then rules are given as

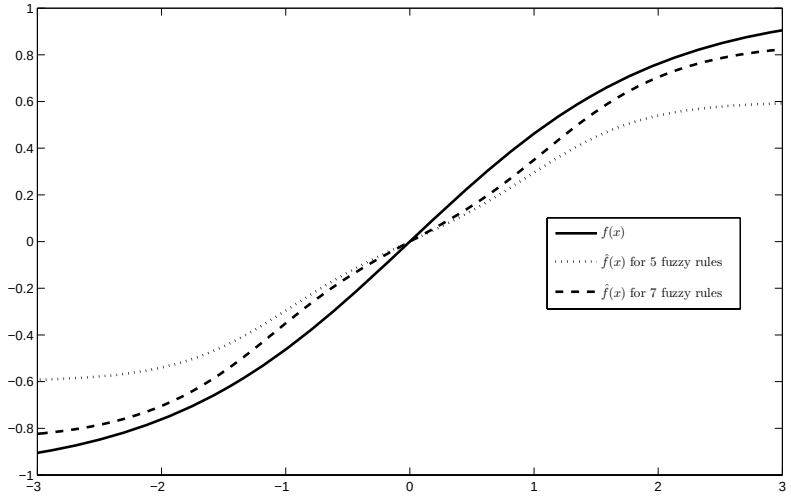
$$\begin{aligned}
R^1: &\quad \text{if } x \text{ is near } -2.5 \text{ then } f(x) \text{ is near } -0.85 \\
R^2: &\quad \text{if } x \text{ is near } -1.5 \text{ then } f(x) \text{ is near } -0.64 \\
R^3: &\quad \text{if } x \text{ is near } -0.5 \text{ then } f(x) \text{ is near } -0.24 \\
R^4: &\quad \text{if } x \text{ is near } 0 \text{ then } f(x) \text{ is near } 0 \\
R^5: &\quad \text{if } x \text{ is near } 0.5 \text{ then } f(x) \text{ is near } 0.24 \\
R^6: &\quad \text{if } x \text{ is near } 1.5 \text{ then } f(x) \text{ is near } 0.64 \\
R^7: &\quad \text{if } x \text{ is near } 2.5 \text{ then } f(x) \text{ is near } 0.85
\end{aligned} \tag{3.23}$$

where $y^1 = -0.85$, $y^2 = -0.64$, $y^3 = -0.24$, $y^4 = 0$, $y^5 = 0.24$, $y^5 = 0.64$ and $y^5 = 0.85$ the conclusions in 3.4.

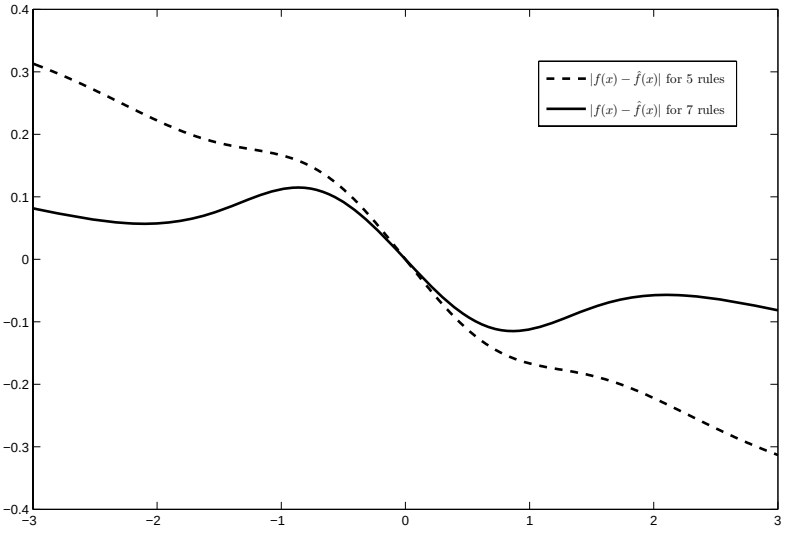
Then the fuzzy system becomes

$$\begin{aligned}
\hat{f}(x) &= \frac{\sum_{l=1}^{7} y^l[\mu_{A^l}(x)]}{\sum_{l=1}^{7}[\mu_{A^l}(x)]} \\
&= \frac{-0.85e^{-(x+2.5)^2} - 0.64e^{-(x+1.5)^2} - 0.24e^{-(x+0.5)^2} + }{e^{-(x+2.5)^2} + e^{-(x+1.5)^2} + e^{-(x+0.5)^2} + e^{-x^2} + } \\
&\quad \frac{0.24e^{-(x-0.5)^2} + 0.64e^{-(x-1.5)^2} + 0.85e^{-(x-2.5)^2}}{e^{-(x-0.5)^2} + e^{-(x-1.5)^2} + e^{-(x-2.5)^2}}
\end{aligned} \tag{3.24}$$

Figure 3.4(a) shows the estimation $\hat{f}(x)$ (dash line). Fig. 3.4(b) shows the estimation error (solid line) $|f(x) - \hat{f}(x)| < 0.15$ over the interval $[-3, 3]$.



(a) The nonlinear function $f(x)$ and the estimation $\hat{f}(x)$ with 5 rules and 7 rules.



(b) the estimation error $|f(x) - \hat{f}(x)|$ with 5 rules and 7 rules.

Figure 3.4: The nonlinear function $f(x)$ and the estimation $\hat{f}(x)$.

Chapter 4

Reinforcement Learning

4.1 Introduction

Reinforcement learning (RL) is a learning technique that maps situations to actions so as to maximize a numerical reward [23]. The learner in reinforcement learning does not know which actions can yield the most reward. The learner must use the reward and try possible actions and discover its optimal action to satisfy its goal. The learner's actions may affect not only the immediate reward but also the next situation and all subsequent rewards [23]. In reinforcement learning, there is no external supervisor that can guide the learner's learning process. The learner has to learn from its own experience by interacting with the environment. Therefore, reinforcement learning can be characterized as a problem of learning by interacting with the environment to achieve a goal.

In a reinforcement learning problem, the learner is called the agent, which interacts with the environment. The goal for an agent is to maximize the rewards received from

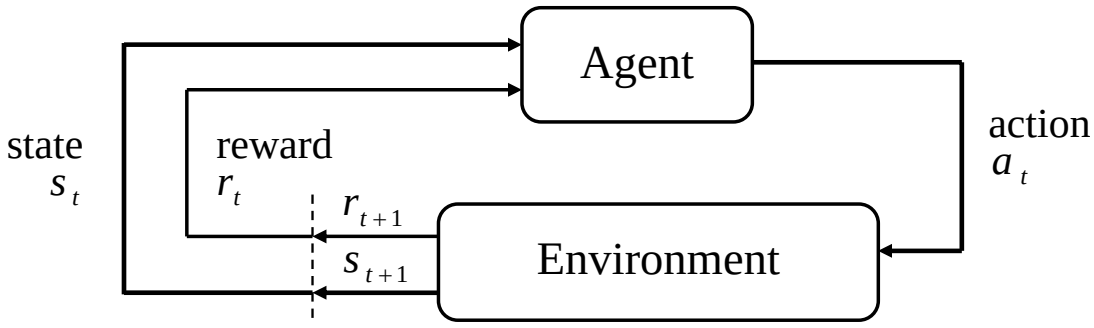


Figure 4.1: The reinforcement learning structure

the environment. Figure 4.1 shows the structure of a reinforcement learning problem. At time t, the agent observes the current state s_t and chooses an action a_t. Then the agent moves to the next state s_{t+1} with a received reward r_{t+1}. The goal for the agent is to maximize the accumulated rewards received from the next state and afterwards. A policy for an agent is defined as the mapping from states to probabilities of choosing each possible action [23]. We use $\pi_t(s, a)$ to denote the agent's policy, which is the probability of choosing action $a_t = a$ at the state $s_t = s$. A policy is called an optimal policy when the agent can maximize its accumulated received rewards if the agent follows this policy.

The above framework is flexible and can be applied in many different ways to many different problems. For example, the time steps can be fixed intervals of real time or successive stages. The actions can be voltages applied to the motors of a robot arm, or high-level decisions. Similarly, the states can be completely determined by direct sensor readings, or high-level and abstract, such as symbolic descriptions of objects in a room [23].

The reinforcement learning framework is a considerable abstraction of the problem of goal-directed learning from interaction [23]. It can be reduced to three signals passing back and forth between an agent and its environment: one signal to represent the choices made by the agent (the actions), one signal to represent the basis on which the choices are made (the states), and one signal to define the agent's goal (the rewards) [23].

In reinforcement learning, the goal of the agent is formalized in terms of the rewards received from the environment. The reward in a reinforcement learning problem is a simple number $r_t \in \Re$. The agent's goal is to maximize the total amount of reward it receives which is the cumulative reward in the long run. For example, suppose that we want a robot learn to escape from a maze. We can set up the reward as zero all the time except that we give a reward of 1 when the robot escapes. Or we can give a reward of -1 for every time step that the robot passes before the escape. In this way, we encourage the agent to escape as quickly as possible.

The agent receives a sequence of immediate rewards $r_{t+1}, r_{t+2}, r_{t+3}, ...$, after each time step t. The sum of the rewards at time t are given as

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots r_T \tag{4.1}$$

where T is a final time step in an episode. Each episode starts from an initial state and ends at a terminal state. We call the task which can be finished in a finite time as an episodic task. Similarly, we call the task as a continuing task if the task goes on continually without limit. Then we have $T = \infty$ in (4.1) for a continuing task.

If we add discounting into the rewards, the goal of the agent becomes maximizing

the sum of the discounted reward received over the future which is

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{4.2}$$

where $\gamma \in [0, 1]$ is the discount factor. The discount factor determines the present value of future rewards such that a reward received k time steps in the future is only worth γ^{k-1} times of the value if it were received immediately. If $\gamma < 1$, the infinite sum in (4.2) has a finite value as long as the reward sequence r_k is bounded. If $\gamma = 0$, the agent always considers maximizing its immediate rewards. If the discount factor γ approaches 1, the agent takes future rewards into account more strongly.

4.2 Markov Decision Processes

Single-agent reinforcement learning can be described as a Markov decision process (MDP) [2].

Definition 1 *A Markov decision process (MDP) is a tuple* (S, A, T, γ, R) *where* S *is the state space,* A *is the action space,* $T : S \times A \times S \to [0, 1]$ *is the transition function,* $\gamma \in [0, 1]$ *is the discount factor and* $R : S \times A \times S \to \mathbb{R}$ *is the reward function.*

In Definition 1, the transition function $T^a_{ss'}$ represents the probability of the next state being s' given the current state s and the selected action a. The transition function satisfies

$$\sum_{s' \in S} T^a_{ss'} = 1 \quad \forall s \in S, \ \forall a \in A \tag{4.3}$$

where s' represents a possible state at the next time step.

The reward function $R^a_{ss'}$ represents the received expected reward at the next state s' given the current action a and the current state s. In a Markov decision process, the states have the Markov property such that the player's next state and reward only depends on the player's current state and action. In other words, the conditional probability distribution of the player's next state and reward only depends on the player's current state and action such that

$$Pr\left\{ s_{t+1} = s', r_{t+1} = r' \,\middle|\, s_t, a_t, \ldots, s_0, a_0 \right\} = Pr\left\{ s_{t+1} = s', r_{t+1} = r' \,\middle|\, s_t, a_t \right\}. \quad (4.4)$$

A player's policy $\pi : S \to A$ in an MDP is defined as a probability distribution over the player's action set A from a given state $s \in S$. Given the state s, a player's policy $\pi(s, a)$ must satisfy

$$\sum_{a \in A} \pi(s, a) = 1 \quad \forall s \in S. \quad (4.5)$$

The goal of a player in an MDP is to maximize the expected long-term reward received from the environment. We denote $\pi^*(s, a)$ as the optimal policy for the player in an MDP as the policy that can maximize the long term rewards. In an MDP, the state-value function $V^\pi(s)$ (or the value of a state) is defined as the expected received reward when the player follows a policy π at state s and thereafter. The state-value function $V^\pi(s)$ is given as

$$V^\pi(s) = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+t+1} \,\middle|\, s_t = s \right\} \quad (4.6)$$

where t is the current time step, $\gamma \in [0,1]$ is a discount factor, and r_{k+t+1} is the received immediate reward at time step $k+t+1$. Under an optimal policy π^*, the state-value function satisfies

$$V^*(s) \geq V^\pi(s) \quad \forall \pi, \forall s \in S \tag{4.7}$$

According to [23], the state-value function in (4.6) can be rewritten as follows

$$\begin{aligned} V^\pi(s) &= E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+t+1} \,\middle|\, s_t = s \right\} \\ &= \sum_{a \in A} \pi(s,a) \sum_{s' \in S} T^a_{ss'} E_\pi\left\{ r_{t+1} + \gamma \sum_{k=0}^{T} \gamma^k r_{k+t+2} \,\middle|\, s_t = s, a_t = a, s_{t+1} = s' \right\} \\ &= \sum_{a \in A} \pi(s,a) \sum_{s' \in S} T^a_{ss'} \big(R^a_{ss'} + \gamma V^\pi(s') \big) \end{aligned} \tag{4.8}$$

where $T^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$ is the probability of the next state being $s_{t+1} = s'$ given the current state $s_t = s$ and action $a_t = a$ at time step t, and $R^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$ is the expected immediate reward received at state s' given the current state s and action a. Equation (4.8) is the Bellman equation for $V^\pi(s)$. We can also have the optimal state-value function $V^*(s)$ if the player follows the optimal policy π^* at state s and thereafter. Then the state-value function $V^*(s)$ is the Bellman optimality equation where

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} T^a_{ss'} \big(R^a_{ss'} + \gamma V^*(s') \big). \tag{4.9}$$

In an MDP, the action-value function $Q^\pi(s,a)$ is defined as the expected received reward when the player chooses an action a at state s and follows a policy π thereafter.

The action-value function $Q^\pi(s,a)$ is given as

$$Q^\pi(s,a) = \sum_{s' \in S} T^a_{ss'} \big(R^a_{ss'} + \gamma V^\pi(s') \big) \tag{4.10}$$

Similar to the state-value function, we have the optimal action-value function $Q^*(s,a)$ when the player chooses an action a at state s and follows the optimal policy π^* thereafter. The optimal action-value function is given as

$$Q^*(s,a) = \sum_{s' \in S} T^a_{ss'} \big(R^a_{ss'} + \gamma V^*(s') \big) \tag{4.11}$$

Given the transition function and the reward function, we can find the optimal state-value function and the optimal policy using a *value iteration* method [23]. The method is listed in Algorithm 1.

Algorithm 1 Value iteration [23]

Initialize $V(s) = 0$ for all $s \in S$
repeat
    $\Delta \leftarrow 0$
    For each $s \in S$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T^a_{ss'} \big(R^a_{ss'} + \gamma V(s') \big)$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ for all $s \in S$ (θ is a small positive number)
Output a deterministic policy π such that
$\pi(s) = \arg\max_a \sum_{s' \in S} T^a_{ss'} \big(R^a_{ss'} + \gamma V(s') \big)$

Example 4.1 *We present an MDP example. A player plays on a* 9×9 *playing field. Starting from the initial position denoted as "P" in Figure 4.2, the player tries to reach a goal denoted as "*". The player can move up, down, left and right. If the*

player's chosen action is taking the player off the grid, the player stays still. At each time step, the player takes an action a and moves one cell. For simplicity, the transition function is set to 1 for each movement. The reward function is set to -1 *on all transitions such that*

$$R^a_{ss'} = -1, \quad \forall s \in S \tag{4.12}$$

The above equation denotes that the player receives -1 *for each movement until the player reaches the goal.*

The player's objective is to reach the goal with minimum steps from the fixed initial position. Based on the player's reward function and transition function, we can find the player's optimal policy using the above value iteration algorithm. We set the discount factor $\gamma = 1$ *and apply Algorithm 1 to the game. Figure 4.3 shows the iterations for updating the state-value function. After 11 iterations, the state-value function converges to the optimal state-value function. Starting from the initial state, the player's optimal policy is shown in Figure 4.4.*

4.3 Temporal-Difference Learning

Temporal-difference(TD) learning is a prediction technique that can be used for solving the reinforcement learning problem. TD learning methods can learn directly from new experience without knowing the environment's dynamics [23]. TD methods can be used to estimate the state-value function V^π for a given policy π. The simplest

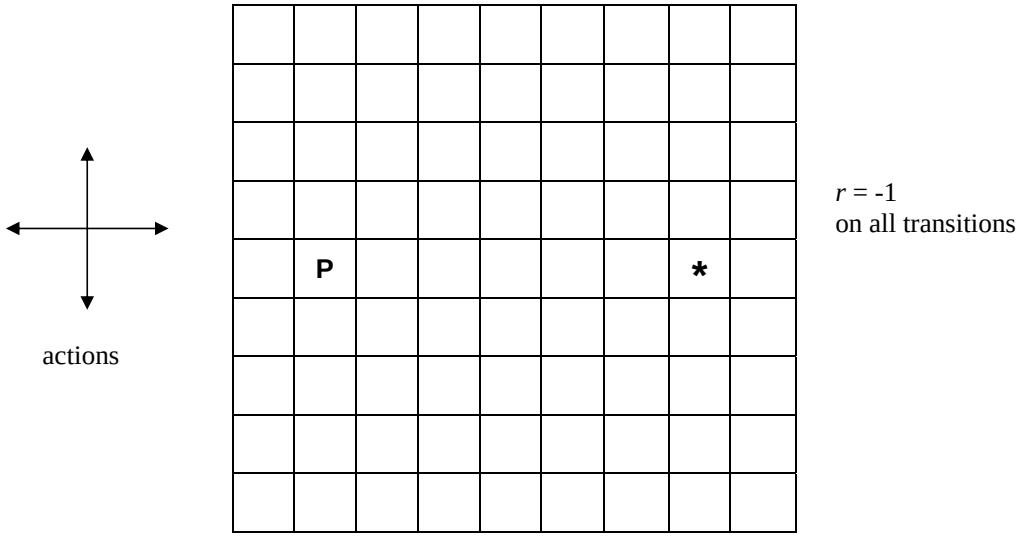


Figure 4.2: An example of Markov decision processes

TD method is given as

$$V(s_t) = V(s_t) + \alpha\big[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\big] \tag{4.13}$$

where α is the learning rate, γ is the discount factor and r_{t+1} is the received immediate reward at time $t+1$. The method in (4.13) is also called TD(0) method. The complete TD(0) method is shown in Algorithm 2.

4.3.1 Q-learning

Q-learning is also described as a model-free reinforcement learning method which is learning the controller without the need of learning the model. Q-learning was introduced by Watkins [26]. The agent in Q-learning learns to choose the correct action without knowing its reward function and transition function.

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	0	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

(a) The state-value function at iteration 1

-2	-2	-2	-2	-2	-2	-2	-2	-2
-2	-2	-2	-2	-2	-2	-2	-2	-2
-2	-2	-2	-2	-2	-2	-2	-2	-2
-2	-2	-2	-2	-2	-2	-2	-1	-2
-2	-2	-2	-2	-2	-2	-1	0	-1
-2	-2	-2	-2	-2	-2	-2	-1	-2
-2	-2	-2	-2	-2	-2	-2	-2	-2
-2	-2	-2	-2	-2	-2	-2	-2	-2
-2	-2	-2	-2	-2	-2	-2	-2	-2

(b) The state-value function at iteration 2

-5	-5	-5	-5	-5	-5	-5	-4	-5
-5	-5	-5	-5	-5	-5	-4	-3	-4
-5	-5	-5	-5	-5	-4	-3	-2	-3
-5	-5	-5	-5	-4	-3	-2	-1	-2
-5	-5	-5	-4	-3	-2	-1	0	-1
-5	-5	-5	-5	-4	-3	-2	-1	-2
-5	-5	-5	-5	-5	-4	-3	-2	-3
-5	-5	-5	-5	-5	-5	-4	-3	-4
-5	-5	-5	-5	-5	-5	-5	-4	-5

(c) The state-value function at iteration 5

-11	-10	-9	-8	-7	-6	-5	-4	-5
-10	-9	-8	-7	-6	-5	-4	-3	-4
-9	-8	-7	-6	-5	-4	-3	-2	-3
-8	-7	-6	-5	-4	-3	-2	-1	-2
-7	-6	-5	-4	-3	-2	-1	0	-1
-8	-7	-6	-5	-4	-3	-2	-1	-2
-9	-8	-7	-6	-5	-4	-3	-2	-3
-10	-9	-8	-7	-5	-5	-4	-3	-4
-11	-10	-9	-8	-7	-6	-5	-4	-5

(d) The state-value function at iteration 11

Figure 4.3: state-value function iteration algorithm in Example 4.1

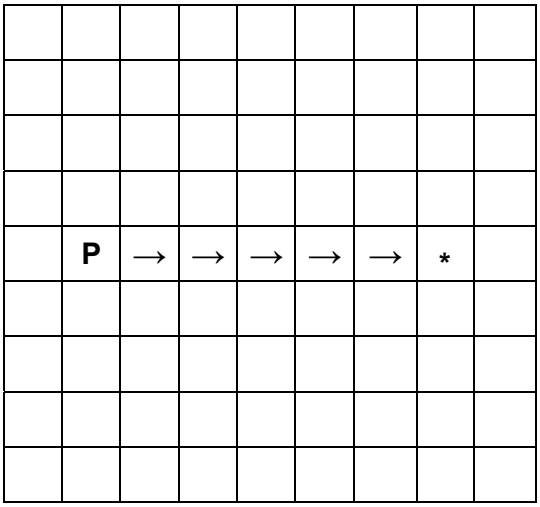


Figure 4.4: The player's optimal policy in Example 4.1

Algorithm 2 TD(0) method (evaluate the given poilcy)

1: Initialize the state-value function $V(s)$
2: **for** Each iteration **do**
3: Initialize s
4: Select an action a at the current state s based on the given policy π.
5: Observe the received immediate reward r at the subsequent state s'.
6: Update $V(s)$ by

$$V(s) \leftarrow V(s) + \alpha\big[r + \gamma V(s') - V(s)\big]$$

7: $s \leftarrow s'$
8: **end for**
9:

The simplest form of Q-learning called one-step Q-learning is given by

$$Q(s,a) = Q(s,a) + \alpha\big(r + \gamma \max_a Q(s',a') - Q(s,a)\big) \tag{4.14}$$

where r is the received immediate reward at time $t+1$, α is the learning rate and γ is the discount factor. The part $r + \gamma \max_a Q(s',a') - Q(s,a)$ in (4.14) is the temporal-difference error calculated at time $t+1$. Using the Q-learning method, the optimal action-value function $Q^*(x,a)$ can be estimated based on the TD error received from the environment.

Q-learning can be directly applied to a learning problem in a discrete domain. For a learning problem in a continuous domain, the action space and the state space need to be discretized in order to setup the Q function. However, the number of state-action pairs in the Q function becomes large when we perform a fine discretization. This requires a large memory storage capacity and results in a slow learning process. The Q-learning algorithm is given in Algorithm 3.

Example 4.2 *We use the same example illustrated in Example 4.1 to test the performance of Q-learning. Assume the player has no prior knowledge of its transition function or reward function. Q-learning algorithm presented in Algorithm 3 is applied to Example 4.1. The discount factor is set to 1 and the learning rate is set to 0.9. For each movement, the player received an immediate reward of -1. The action selection is based on an ϵ-greedy strategy such that the player selects an action randomly with probability 0.2 or a greedy action with probability 0.8. We run one simulation which includes 200 episodes of the game. Each episode starts at the player's initial position and ends when the player reaches the goal. In each episode, we record how many*

Algorithm 3 Q-learning

Initialize $Q(s,a) = 0\ \forall s \in S, a \in A$
for each episode **do**
    initialize s
    for each step **do**
        select an action a at the current state s using ϵ-greedy strategy.
        observe the received immediate reward r
        observe s'
        update $Q(s,a)$ by
        $Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big)$
        $s \leftarrow s'$; $a \leftarrow a'$
    end for
end for
(ϵ-greedy strategy is an action-selection strategy such that the player selects a greedy action with a probability 1-ϵ and a random action with a probability ϵ)

steps the player needed to reach the goal. We plot the steps to reach the goal versus episodes. Then we run 40 simulations and average the result. Figure 4.5 shows that Q-learning helped the player minimize the steps to reach the goal.

4.4 Eligibility Traces

Eligibility traces are one of the basic mechanisms of reinforcement learning that can be used for temporal-difference methods, such as Q-learning. One way to view eligibility traces is based on the mechanistic perspective [23]. From this perspective, an eligibility trace records the occurrence of an event, such as the number of times that the same state is visited or the number of times the same action is taken. When a TD error is received from the environment, the trace checks all possible states and actions that are related. Credit or blame is assigned to only the eligible states or actions. The TD error is then updated according to the eligible states or actions.

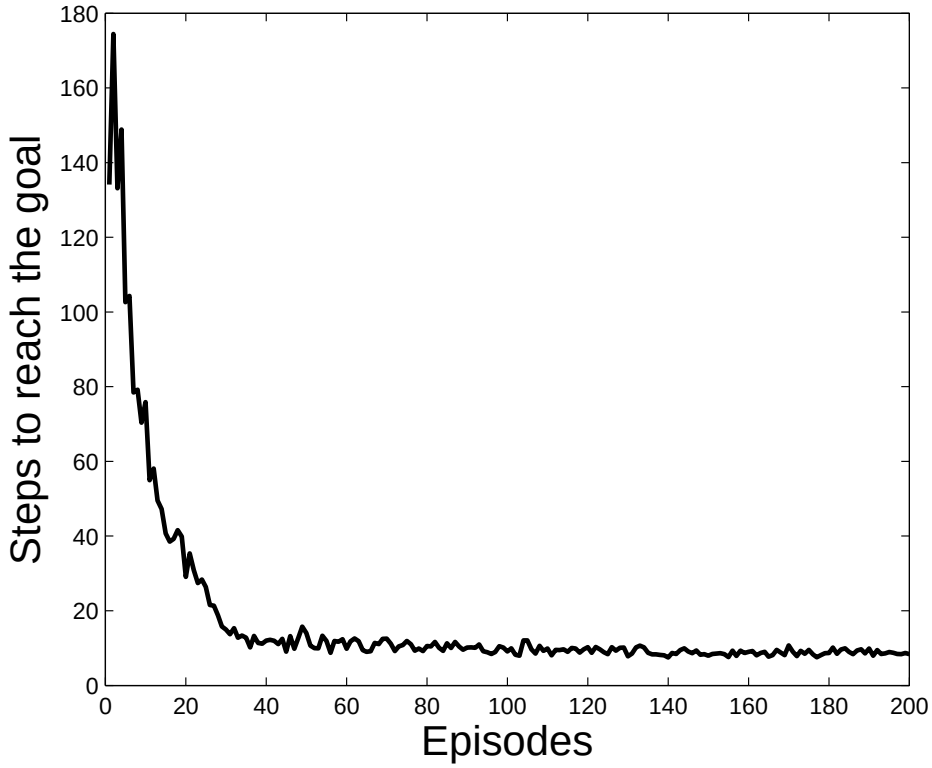


Figure 4.5: Q-learning applied to Example 4.1

We present the backward view of the TD(λ) algorithm. The eligibility trace is represented as an additional memory variable associated with each state. The eligibility trace is denoted by $e_t(s) \in \Re^+$ for state s at time t. One type of eligibility trace is the replacing trace. After each time step t, the eligibility traces for all states decay by $\gamma\lambda$, and the eligibility trace for the specific state revisited at time t is set to 1 such that

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ 1 & \text{if } s = s_t \end{cases} \tag{4.15}$$

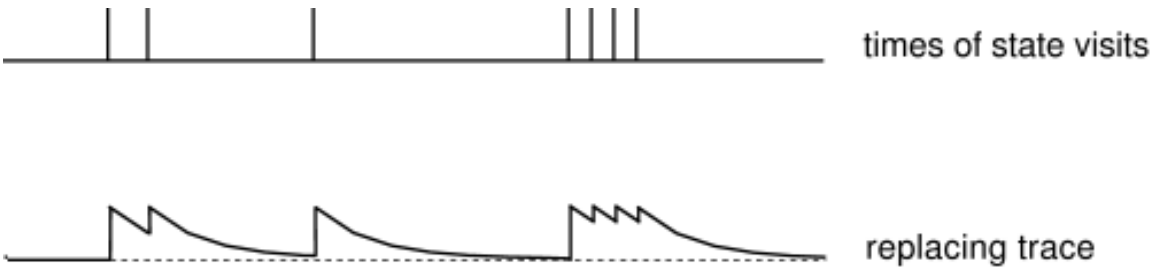


Figure 4.6: Illustration of eligibility traces (origin from [23])

where γ is the discount factor and λ is called the decay-rate parameter for eligibility traces. The replacing trace in (4.15) resets to 1 the state revisited, and fades away gradually when the state is not revisited. The replacing trace is illustrated in Figure 4.6.

The traces in Figure 4.6 record which states have recently been visited in terms of $\gamma\lambda$ [23]. The TD error for state-value prediction is given as

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t) \tag{4.16}$$

Based on the above equation and the eligibility traces in (4.15), the global TD error is given by

$$\Delta V_t(s) = \alpha \delta_t e_t(s), \forall s \in S \tag{4.17}$$

A complete TD(λ) algorithm is given in Algorithm 4, where each update that depends on the current TD error is combined with traces of past events.

In TD(λ) algorithm, if $\lambda = 0$, we call it TD(0) where all traces are zero at t except for the trace corresponding to s_t. Then the TD(λ) reduces to the simple TD rule in

Algorithm 4 TD(λ)

Initialize s
for Each step **do**
    Select an action a at the current state s given by π.
    Observe the received immediate reward r at the subsequent state s'.
    Update
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For all s:
    $V(s) \leftarrow V(s) + \alpha\delta e(s)$
    $e(s) \leftarrow \gamma\lambda e(s)$
    $s \leftarrow s'$
end for

(4.13).

4.4.1 Q(λ)-learning

Watkin's $Q(\lambda)$ learning [27] is the combination of eligibility traces and Q-learning. At each time step, the traces for all state-action pairs are decayed by $\gamma\lambda$ if a greedy action was taken or set to 0 if an exploratory action was taken, and the trace corresponding to the current state and action is reset to 1. The replacing trace is given as

$$e_t(s,a) = \begin{cases} 1 & \text{if } s = s_t \text{ and } a = a_t \\ 0 & \text{if } s = s_t \text{ and } a \neq a_t \\ \lambda\gamma e_{t-1}(s,a) & \text{if } s \neq s_t \end{cases} \quad \text{for all s and a} \tag{4.18}$$

The eligibility trace for the continuous state and action spaces is defined as [3, 6]

$$e_t = \gamma\lambda\, e_{t-1} + \frac{\partial Q_t(s_t, a_t)}{\partial \xi} \tag{4.19}$$

where ξ is the parameter to be tuned.

The learning part for the $Q(\lambda)$ learning algorithm is given as

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha\delta_t e_t(s, a), \tag{4.20}$$

where

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t). \tag{4.21}$$

The complete $Q(\lambda)$ learning algorithm is shown in Algorithm 5.

Algorithm 5 Q(λ)-learning

initialize Q(s,a) arbitrarily
$e \leftarrow 0$ ▷ initialize the eligibility traces
for each episode **do**
    Initialize s, a
    for each step **do**
        Take action a, observe r, s'.
        Choose action a' from state s' using ϵ-greedy strategy
          $a^* \leftarrow \arg\max_a Q(s', a)$
          $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s, a)$
        For all s, a:
          $Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$
        Update eligibility traces $e(s, a)$ for all s, a based on (4.18)
        $s \leftarrow s'; a \leftarrow a'$
    end for
end for

We now apply the above $Q(\lambda)$-learning to Example 4.1. We use the same parameter settings as in Example 4.2. The decay-rate parameter is set to 0.9. We run 40 simulations which includes 200 runs of the game in each simulation and average the result. The dash line in Figure 4.7 is the result from Figure 4.2 for Q-learning. We add it here for comparison. The solid line is the result from $Q(\lambda)$-learning. Compared with Q-learning, $Q(\lambda)$-learning slightly speeded up the convergence of the player's learning process. The learning speed did not change significantly. Note that when using traces, one should consider that traces require more computations per episode and more memory capacity.

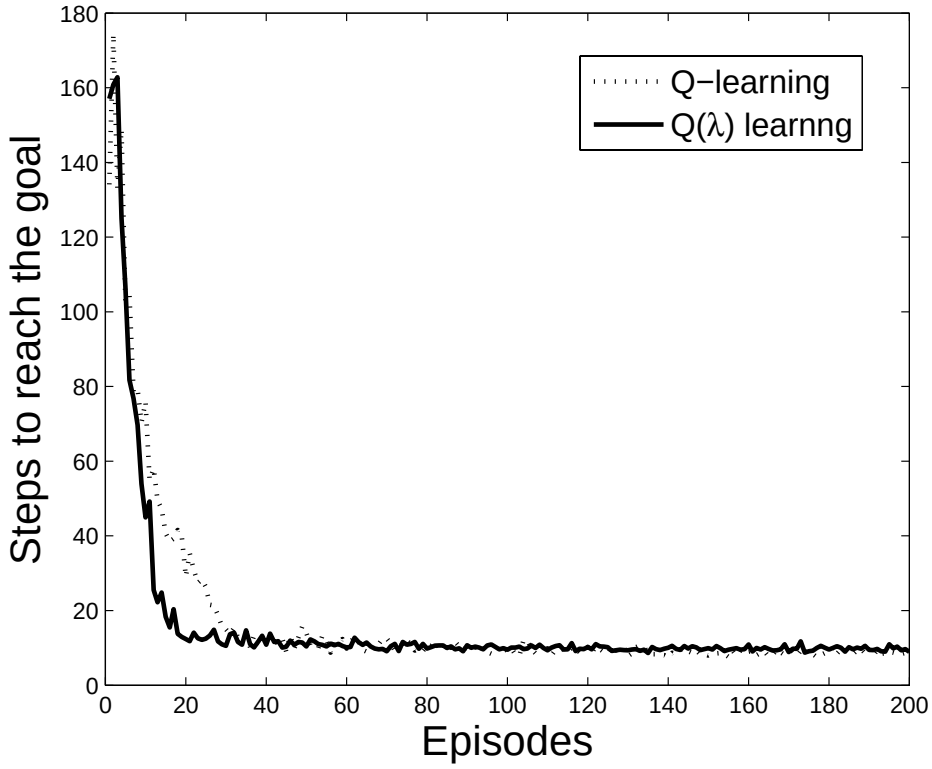


Figure 4.7: $Q(\lambda)$-learning and Q-learning applied to Example 4.1

4.5 Actor-Critic Methods

The actor-critic structure is shown in Fig. 4.8, where the policy structure is called the actor and the estimated value function is called the critic [23]. The actor is used to select the action based on the current policy, and the critic is used to estimate the state-value function $V(s)$. After the actor generates an action, the critic will evaluate the generated action in the form of TD errors defined as

$$\delta_t = r_{t+1} + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t) \tag{4.22}$$

where $\hat{V}$ is the estimated value function.

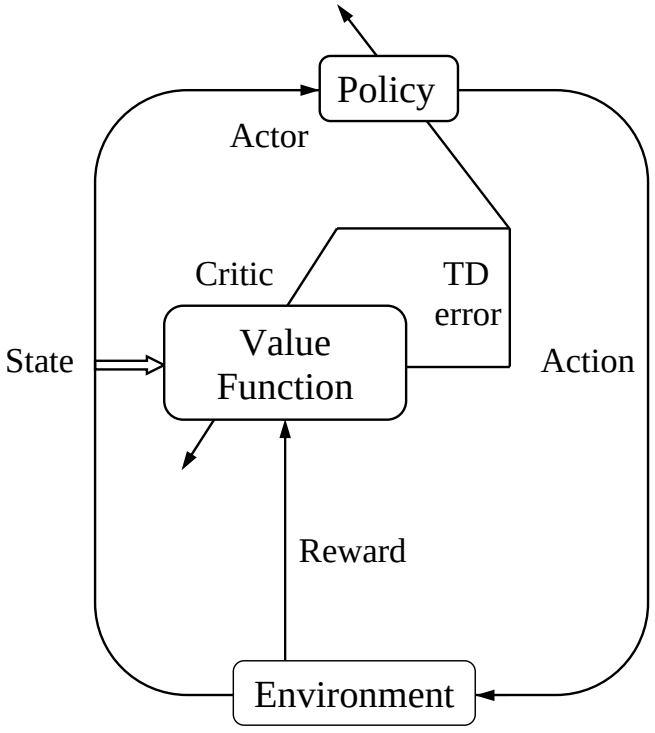


Figure 4.8: The actor-critic architecture (origin from [23])

Actor-critic methods can be used for reinforcement learning problems in a continuous domain such as the pursuit-evasion differential game. In order to represent the continuous state space and generate output signals, the critic and the actor can be represented by fuzzy systems [9]. Based on Figure 4.8, the parameters of the fuzzy system can be updated by TD errors. One of the earliest methods approximates $Q^*(s, a)$ rather than $V(s)$, where the critic and the actor can be updated with Q-learning algorithm [6]. In the next chapter, we will present the structure in [6] and apply the algorithm for differential games.

4.6 Summary

In this chapter, we reviewed the framework of reinforcement learning. We presented the elements of a reinforcement learning problem. These elements include the reinforcement learning agent and its environment, the reward and the agent's policy. We reviewed Markov decision processes and the Markov property. We presented temporal-difference learning methods and Q-learning as one of the TD methods. After that, we discussed eligibility traces which can be combined with TD methods. We illustrated $Q(\lambda)$-learning which is a combination of the eligibility traces and Q-learning. We investigated the learning speed when eligibility traces are combined with Q-learning. Actor-critic methods were also introduced in this chapter.

Chapter 5

Reinforcement Learning applied to Pursuit-Evasion Games

Desouky and Schwartz [6] proposed a learning technique called the "Q(λ)-learning fuzzy inference system" based on reinforcement learning. Reinforcement learning is used to tune the parameters of the fuzzy systems without the need of an expert or prior knowledge of the system's behaviours. The QLFIS technique has been successfully applied to the pursuit-evasion differential game to train the pursuer to capture the evader in minimum time.

There is however a lack of studies on how the evader can learn its optimal strategy by playing the game. In this work, we focus more on learning the behaviours and the strategies of the evader. The QLFIS algorithm will be modified to train both the evader and the pursuer simultaneously. We extend the game by adding the distance as an input for the evader to take the appropriate actions whenever the pursuer reaches some threshold distance. Unlike the previous work, the trained evader learns to find

this distance and to make sharp turns (extreme strategy) to maximize the time of capture and, if possible, avoid being captured. At the same time, the pursuer learns to capture the evader, if possible, and minimize the time of capture.

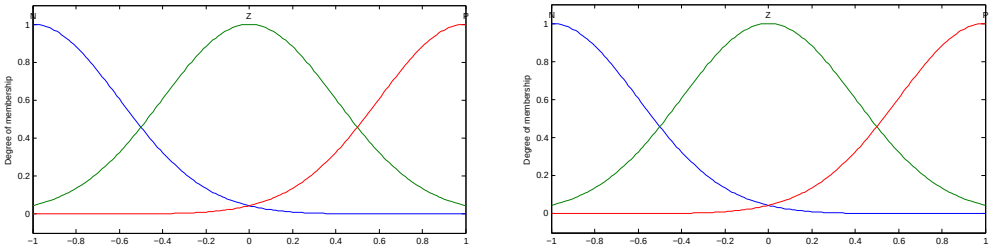
The complexity of the game and the learning process increases when using function approximation and when designing an evader that learns to turn and escape from the pursuer. Therefore we need to consider the learning speed. This chapter also investigates the learning speed when using eligibility traces with Q-learning.

Two versions of pursuit-evasion differential games are used: the homicidal chauffeur game and the game of two cars. The learning technique will be applied to the games and simulation results will be evaluated and compared to the optimal solution results. The capture condition will be considered when evaluating the trained evader.

This chapter is organized as follows. The fuzzy system structure used with the learning system is developed in Sect. 5.1. Sect. 5.2 describes the QLFIS technique and the updating rules. The learning algorithm and the simulation results are presented in Sect. 5.3.

5.1 Fuzzy Controller Structure

The pursuit-evasion games is described in Sect. 2.1. The inputs for the pursuer are the angle difference between the pursuer and the evader ϕ and its rate of change $\dot{\phi}$. The control strategy for the pursuer and the evader is to drive the angle difference to zero. Moreover, because the evader is higher in maneuverability, the distance between the evader and the pursuer d is critical for the evader to decide when to make a sharp turn. The inputs for the evader are the angle difference ϕ and the distance d.



(a) The angle difference ϕ

(b) The rate of change of the angle difference $\dot{\phi}$

Figure 5.1: The pursuer's membership functions before training

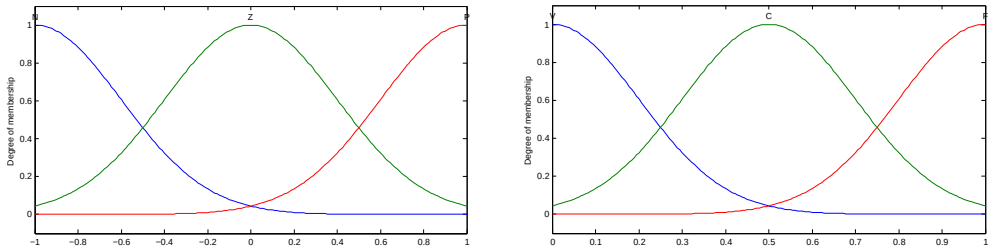
For simplicity and to avoid the "curse of dimensionality" problem, three fuzzy sets are formed for each input to construct the controller. The fuzzy sets of the pursuer are positive (P), zero (Z), and negative (N) for the angle difference ϕ and its derivative $\dot{\phi}$. The fuzzy sets for the evader are positive (P), zero (Z) and negative (N) for the angle difference, and far (F), close (C) and very close (V) for the distance.

Three Gaussian membership functions (MFs) are used for each input, which results in constructing $3^2 = 9$ rules. The Gaussian MFs are given as

$$\mu_{A_l}(x_i) = \exp\left(-(\frac{x_i - c_i^l}{\sigma_i^l})^2\right) \tag{5.1}$$

where the Gaussian MF parameters, the mean c and the standard deviation σ, are the input parameters to be tuned by RL signals. Figures 5.1 and 5.2 show the initial MFs before tuning for the pursuer and the evader respectively.

We assume that the controller is a fuzzy logic controller (FLC). We use a zero-order Takagi-Sugeno (TS) fuzzy inference system (FIS) described in Sect. 3.1. The TS fuzzy model consists of fuzzy IF-THEN rules and a fuzzy inference engine. The



(a) The angle difference ϕ

(b) The distance between the pursuer and the evader d

Figure 5.2: The evader's membership functions before training.

consequence of a zero-order TS fuzzy system is constant. Given the fuzzy variables x_i and the corresponding fuzzy sets A_j and B_j, the fuzzy IF-THEN rules are

$$\Re_l : if \;\; x_1 \;\; is \;\; A_j \;\; and \;\; x_2 \;\; is \;\; B_j \;\; THEN \;\; f_l = K^l \tag{5.2}$$

where x_i are the inputs ϕ and $\dot{\phi}$ for the pursuer, and ϕ and d for the evader. The term f_l is the output of rule l, and K^l is the consequence part of the fuzzy rules.

The controller is formed using a zero-order TS FIS. The output of the FLC is the steering angle u formed by the weighted average defuzzification expressed as

$$u = \frac{\sum_{l=1}^{9}((\prod_{i=1}^{2}\mu_{A_l}(x_i))K^l)}{\sum_{l=1}^{9}(\prod_{i=1}^{2}\mu_{A_l}(x_i))} \tag{5.3}$$

Similarly, the function approximation is formed using a zero-order TS FIS. The

output of the FIS is called a Q-value calculated as

$$Q(s,a) = \frac{\sum_{l=1}^{9}((\prod_{i=1}^{2}\mu_{A_l}(x_i))\beta^l)}{\sum_{l=1}^{9}(\prod_{i=1}^{2}\mu_{A_l}(x_i))} \tag{5.4}$$

where β is the consequence constant K^l in Eq. (5.2)

The fuzzy rules are illustrated using the tabular format. The initial output constant K^l of the fuzzy rules before learning for the pursuer and the evader are shown in tables 5.1 and 5.2 respectively .

Table 5.1: The output constant K^l of the pursuer's fuzzy decision table before learning

ϕ \ $\dot{\phi}$	N	Z	P
N	-0.5	-0.25	0.0
Z	-0.25	0.0	0.25
P	0.0	0.25	0.5

Table 5.2: The output constant K^l of the evader's fuzzy decision table before learning

ϕ \ d	V	C	F
N	$-\pi/2$	$-\pi/2$	$-\pi/4$
Z	$-\pi/2$	$\pi/2$	0.0
P	$\pi/2$	$\pi/2$	$\pi/4$

5.2 Q(λ)-learning Fuzzy Inference System

The construction of the learning system is given in Figure 5.3. As we discussed in Section 4.5, we present in this section a fuzzy actor-critic method algorithm called Q-learning fuzzy inference system QLFIS proposed in [6]. In the QLFIS technique, the controller and the function approximator are represented by fuzzy systems. The function approximator "FIS" is used to generalize or predict the optimal-value function $Q^*(s, a)$, while Q(λ)-learning is used to tune the input and the output of the fuzzy controller "FLC" and the function approximator. The advantage of QLFIS technique is that one can use Q-learning in a continuous domain by using a fuzzy inference system to represent the continuous state space and action space. Desouky and Schwartz [6] derived the update rules of the learning process. We will briefly describe the derivation of the learning process to understand the update rules.

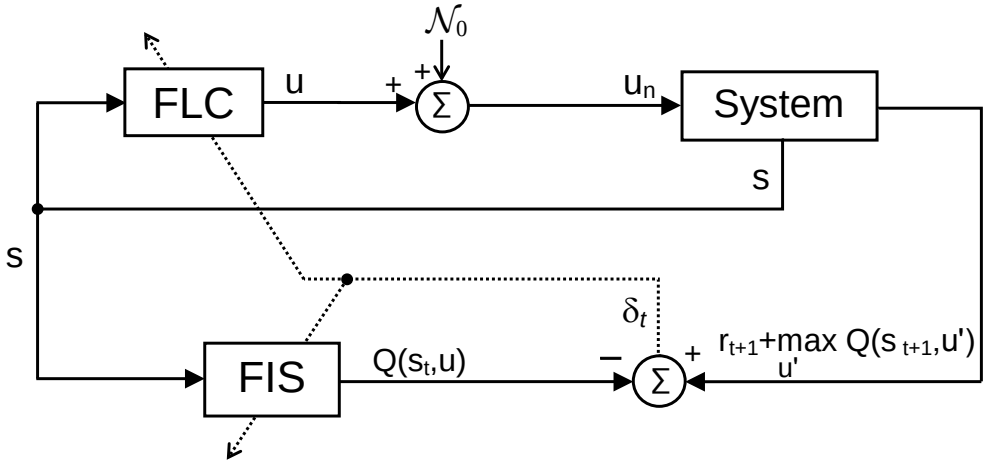


Figure 5.3: Construction of the learning system where the white Gaussian noise $\mathcal{N}(0, \sigma_n^2)$ is added as an exploration mechanism.

The Q-learning estimates the $Q^*(s,a)$ based on the TD error. The TD error δ_t shown in Figure 5.3 is calculated as

$$\delta_t = r_{t+1} + \gamma \max_{\acute{a}} Q_t(s_{t+1}, \acute{a}) - Q_t(s_t, a_t) \tag{5.5}$$

Apply the eligibility traces e_t as defined in Sect. 4.4.1. Then the action-value function $Q(s,a)$ is updated by

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \delta_t \, e_t \tag{5.6}$$

where $(0 < \alpha \leq 1)$ is the learning rate. The eligibility traces with the trace decay-rate parameter $(0 \leq \lambda \leq 1)$ are given in Eq. (4.19).

$$e_t = \gamma \lambda \, e_{t-1} + \frac{\partial Q_t(s_t, a_t)}{\partial \xi} \tag{5.7}$$

Define the vector of the input and output parameters by

$$\xi = \begin{pmatrix} K \\ c \\ \sigma \end{pmatrix} \tag{5.8}$$

Then the update rules for the FIS are defined by [6]

$$\xi_{FIS}(t+1) = \xi_{FIS}(t) + \eta \delta_t \left[\gamma \lambda e_{t-1} + \frac{\partial Q_t(s_t, u_t)}{\partial \xi_{FIS}} \right] \tag{5.9}$$

and the update rules for the FLC are defined by

$$\xi_{FLC}(t+1) = \xi_{FLC}(t) + \zeta\delta_t \left[\frac{\partial u}{\partial \xi_{FLC}} (\frac{u_n - u}{\sigma_n}) \right] \tag{5.10}$$

where

$$\frac{\partial Q_t(s_t,u_t)}{\partial \xi_{FIS}} = \begin{pmatrix} \frac{\partial Q_t(s_t,u_t)}{\partial K^l} \\ \frac{\partial Q_t(s_t,u_t)}{\partial c_i^l} \\ \frac{\partial Q_t(s_t,u_t)}{\partial \sigma_i^l} \end{pmatrix} = \begin{pmatrix} \sum_l \bar{\omega}_l \\ \frac{(K^l - Q_t(s_t,u_t))}{\sum_l \omega_l} \omega_l \frac{2(x_i - c_i^l)}{(\sigma_i^l)^2} \\ \frac{(K^l - Q_t(s_t,u_t))}{\sum_l \omega_l} \omega_l \frac{2(x_i - c_i^l)^2}{(\sigma_i^l)^3} \end{pmatrix} \tag{5.11}$$

$$\frac{\partial u}{\partial \xi_{FLC}} = \begin{pmatrix} \frac{\partial u}{\partial K^l} \\ \frac{\partial u}{\partial c_i^l} \\ \frac{\partial u}{\partial \sigma_i^l} \end{pmatrix} = \begin{pmatrix} \sum_l \bar{\omega}_l \\ \frac{(K^l - u)}{\sum_l \omega_l} \omega_l \frac{2(x_i - c_i^l)}{(\sigma_i^l)^2} \\ \frac{(K^l - u)}{\sum_l \omega_l} \omega_l \frac{2(x_i - c_i^l)^2}{(\sigma_i^l)^3} \end{pmatrix} \tag{5.12}$$

with the learning rate η for the FIS and ζ for the FLC. The terms ω_l and $\bar{\omega}_l$ are called the firing strength and the normalized firing strength of rule l [6], defined as follows

$$\omega_l = \prod_{i=1}^{2} \exp\left(-(\frac{x_i - c_i^l}{\sigma_i^l})^2 \right) \tag{5.13}$$

$$\bar{\omega}_l = \frac{\omega_l}{\sum_{l=1}^{9} \omega_l} \tag{5.14}$$

We now construct the immediate reward r used in RL update rules. The goal of the players is to use the received rewards to learn appropriate actions. Because the pursuer aims to capture the evader and minimize the time of capture, the distance becomes the optimization criteria, such that the pursuer wants to minimize the distance while the evader wants to maximize the distance. The distance $D(t)$ at each time step t is calculated by Eq. (2.8). The variation of the distance, $\Delta D(t)$, is calculated

as

$$\Delta D(t) = D(t) - D(t+1) \tag{5.15}$$

The maximum value of variation of the distance is defined as $\Delta D_{max} = V_{rmax}T$ where V_{rmax} is the maximum relative velocity of the pursuer and the evader ($V_{rmax} = V_p + V_e$) and T is the sampling time.

The pursuer approaches the evader with a positive value of $\Delta D(t)$, while the evader moves away from the pursuer with a negative value. The immediate reward r for the players is given by

$$r_{t+1} = \begin{cases} \dfrac{\Delta D(t)}{\Delta D_{max}} & \text{for pursuer} \\ -\dfrac{\Delta D(t)}{\Delta D_{max}} & \text{for the evader} \end{cases} \tag{5.16}$$

5.2.1 Q-learning Fuzzy Inference System Without Eligibility Traces

Eligibility traces are used to modify a one-step update rule to a multi-step update. This means that the agent will take into account not only the immediate reward but also the future rewards. If $\lambda = 0$, traces are set to zero at t except for the trace corresponding to s_t. Then the Q(λ) is reduced to one-step update, or simply Q-learning. We will simulate the system with and without the use of eligibility traces to test the effect of the eligibility traces on the learning process.

5.3 Simulation Results

The learning process is modified for our game to train both the evader and the pursuer as shown in Algorithm 7, where M is the number of episodes (games) and N is the number of steps (plays) in each episode.

The game is simulated for different numbers of episodes. The number of steps is 600, and the sampling time is $0.1sec$. The game terminates when the pursuer captures the evader, or when the time exceeds 60 sec (escape).

The pursuer is twice as fast as the evader such that $w_p = 2m/s$ and $w_e = 1m/s$. The wheelbase of the pursuer is $L = 0.3m$. The capture radius is $\ell = 0.15m$ which is half the wheelbase of the pursuer $\ell = \frac{L_p}{2}$. The maximum steering angle of the pursuer is $-0.5rad \leq u_{p_{max}} \leq 0.5rad$ with $R_p = 0.5491m$. Given the stated parameters of the system and using Isaacs' capture condition in Eq. (2.9), there exists a strategy for the evader to avoid capture. The reason why we choose these parameters is that, based on the chosen parameters, the evader with its optimal strategy can always escape. Therefore, our goal is to test whether the evader will learn this optimal strategy. We assume that the evader has no knowledge of the strategy of the pursuer.

The parameters of the game are also initialized such that the pursuer can capture the evader with the current strategies before learning. However, the capture conditions in Eq. (2.9) are set such that the evader can theoretically escape. The goal of initializing the parameters such that the pursuer captures the evader, is to test whether the evader will eventually learn to escape. We then run the simulation allowing both the evader and the pursuer to learn simultaneously. Both players use the same learning algorithm.

Algorithm 6 QLFIS Algorithm for the pursuer and the evader.

1: Initialize the membership function values for the antecedents of the FLC and the FIS as shown in Figures 5.1 and 5.2.
2: Initialize the values for the consequent K^l of the controller with the values shown in tables 5.1 and 5.2. The values of the FIS are the Q-values initialized to zeroes.
3: $e \leftarrow 0$ ▷ eligibility traces of the FIS
4: $\gamma \leftarrow 0.95$; $\lambda \leftarrow 0.9$
5: $\sigma_n \leftarrow 0.08$
6: **for** $i \leftarrow 1$ **to** M **do**
7: $\eta \leftarrow (0.1 - 0.09\left(\frac{i}{\mathrm{M}}\right))$
8: $\zeta \leftarrow (0.01 - 0.009\left(\frac{i}{\mathrm{M}}\right))$
9: Initialize the position of the pursuer at the origin $(x_p, y_p) = (0, 0)$
10: Initialize the position of the evader (x_e, y_e) randomly
11: Update the state of the pursuer $s_p = (\phi, \dot{\phi})$ and the evader $s_e = (\phi, d)$
12: Calculate the output of the FLC

$$u \leftarrow \frac{\sum_{l=1}^{9}((\prod_{i=1}^{2}\mu_{A_l}(x_i))K^l)}{\sum_{l=1}^{9}(\prod_{i=1}^{2}\mu_{A_l}(x_i))}$$

▷ for the pursuer and the evader

13: **for** $j \leftarrow 1$ **to** N **do**
14: $u_n \leftarrow u + \mathcal{N}_0$ ▷ for the pursuer and the evader
15: Calculate the output of the FIS for the current states

$$Q(s_t, u) \leftarrow \frac{\sum_{l=1}^{9}((\prod_{i=1}^{2}\mu_{A_l}(x_i))\beta^l)}{\sum_{l=1}^{9}(\prod_{i=1}^{2}\mu_{A_l}(x_i))}$$

16: Play the game, observe the next states s_{t+1} for P and E
17: Calculate the reward r from Eq. (5.16)
18: Calculate the output of the FIS for the new states

$$Q_(s_{t+1}, u') \leftarrow \frac{\sum_{l=1}^{9}((\prod_{i=1}^{2}\mu_{A_l}(x_i))\beta^l)}{\sum_{l=1}^{9}(\prod_{i=1}^{2}\mu_{A_l}(x_i))}$$

Algorithm 7 QLFIS Algorithm (Continued.)

Calculate the TD error

$$\delta_t \leftarrow r_{t+1} + \gamma \max_{\acute{a}} Q_t(s_{t+1}, \acute{a}) - Q_t(s_t, a_t)$$

Calculate the gradient for the input and the output parameters of the FLC and the FIS from Eq. (5.11) and Eq. (5.12).

Update the eligibility traces e_t for the FIS using Eq. (4.18)

Update the input and output parameters of the FIS

$$\xi(t+1)_{FIS} \leftarrow \xi_{FIS}(t) + \eta\delta_t \left[\gamma\lambda e_{t-1} + \frac{\partial Q_t(s_t,u_t)}{\partial \xi_{FIS}}\right]$$

Update the input and output parameters of the FLC

$$\xi(t+1)_{FLC} \leftarrow \xi_{FLC}(t) + \zeta\delta_t \left[\frac{\partial u}{\partial \xi_{FLC}}\left(\frac{u_n - u}{\sigma_n}\right)\right]$$ ▷

$s_t \leftarrow s_{t+1}$; $u \leftarrow u'$

end for

end for

We start by simulating the pursuer only assuming that the evader is a fixed point on the plane to evaluate the fuzzy controller and the learning process. We further will investigate how much the learning process can be improved with the use of eligibility traces. Then we apply the learning algorithm to the homicidal chauffeur game and to the game of two cars to make both the pursuer and the evader learn simultaneously. In the game of two cars, we will investigate the use of eligibility traces in $Q(\lambda)$-learning and test the convergence speed of the trained evader which learns to escape.

5.3.1 Pursuer Moving to a Fixed Point

We start by simulating a simple game such that the pursuer is moving to a fixed point on the plane. The initial position of the pursuer is at the origin $(x_p, y_p) = (0, 0)$ and the initial orientation is $\theta = 0rad$. We assume that the fixed point is at $(x, y) = (30, 17)$.

When the pursuer plays its optimal strategy given in Eq. (2.12), the pursuer

reached the point after 17.1 sec, which is called the time of capture. Figure 5.4 shows the path of the pursuer.

Now we use a fuzzy controller as the main controller. The output of the fuzzy controller is calculated by Eq. (5.3). The MFs of the pursuer are shown in Figure 5.1 and the fuzzy consequence parameters K^l are shown in table 5.1. The parameters of the pursuer are initialized with a priori knowledge of its optimal play. When using the fuzzy control before learning, the pursuer reached the point in 17.6 sec. The path of the pursuer with the stated initial parameters is shown in Figure 5.4.

To evaluate the learning technique, the learning algorithm is applied to the fuzzy controller for the pursuer to learn its optimal strategy. The pursuer learns from the game by tuning the input and the output parameters of the fuzzy controller. After approximately 90 learning episodes, the pursuer reduced the time to reach the point to 17.2 sec. The path of the pursuer after learning in Figure 5.4 shows that the evader can learn to correct its path to reach the point in minimum time. Note that we used three membership function for simplicity. The path can be improved by adding more membership functions as we have shown in Example 3.1. However, the number of fuzzy rules will increase exponentially.

We now apply Q-learning without the use of eligibility traces to test the performance of $Q(\lambda)$-learning. In each episode, we record how much time the player needs to reach the point. We plot the time to reach the goal versus 100 episodes. Then we run 20 simulations and average the result. The solid line in Figure 5.5 is the result from using $Q(\lambda)$-learning. The dash line is the result from using Q-learning. Compared with Q-learning, $Q(\lambda)$-learning did not significantly speed up the convergence of the player's learning process. Moreover, when a function approximation is

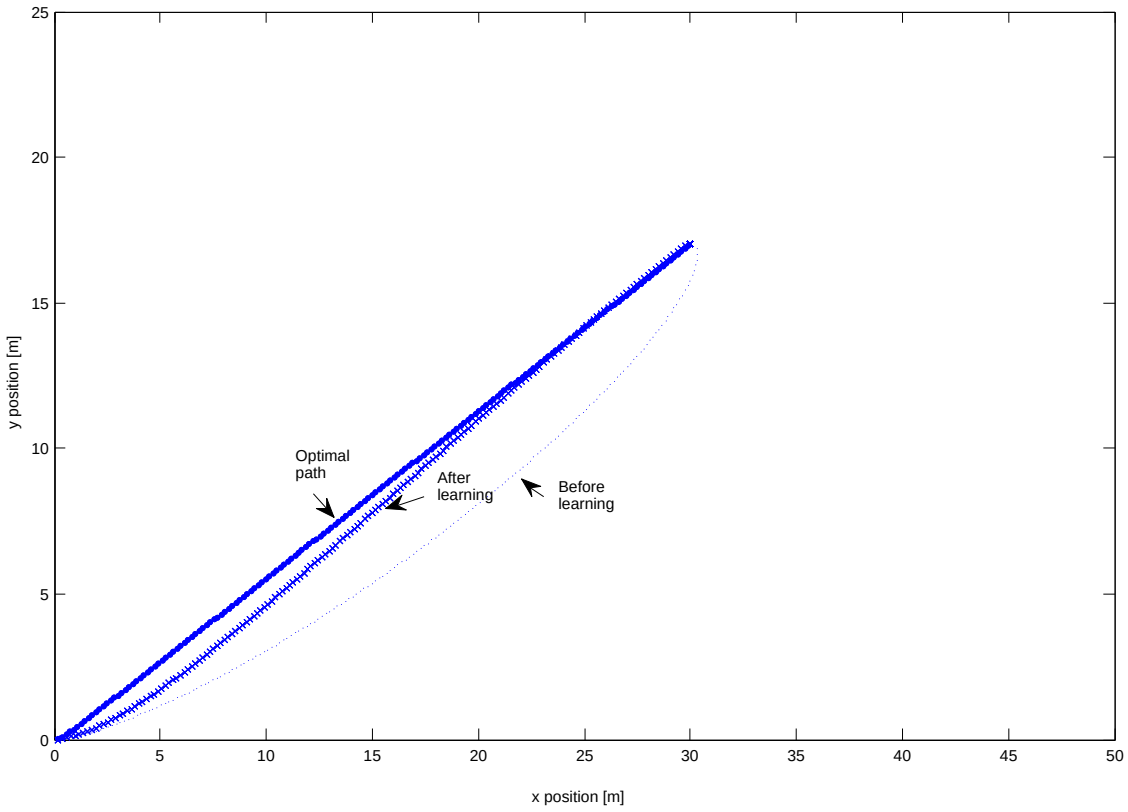


Figure 5.4: The path of the pursuer after learning

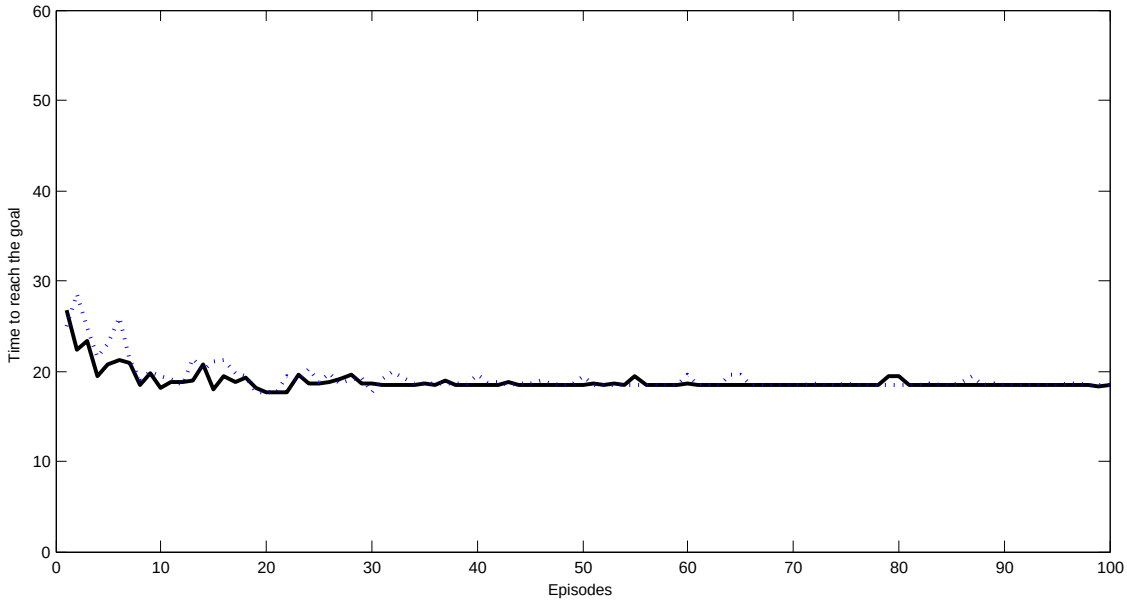


Figure 5.5: $Q(\lambda)$-learning and Q-learning applied to the pursuer. The solid line is the result from using $Q(\lambda)$-learning. The dash line is the result from using Q-learning.

used with traces, the traces required more computations per episode and more memory capacity. The average time of computation to complete 100 learning episodes is 28.8 sec when using $Q(\lambda)$-learning, while the average time of computation when using Q-learning is 21.4 sec.

5.3.2 Homicidal Chauffeur Game

We now simulate the homicidal chauffeur game when the evader learns to turn. The learning algorithm is given in Algorithm 7. Each episode starts by initializing the position of the evader randomly. The initial position of the pursuer is at the origin $(x_p, y_p) = (0, 0)$ and the initial orientation is $\theta = 0rad$. The kinematic equations of the pursuer and the evader are given in Eq. (2.6). The parameters of the fuzzy

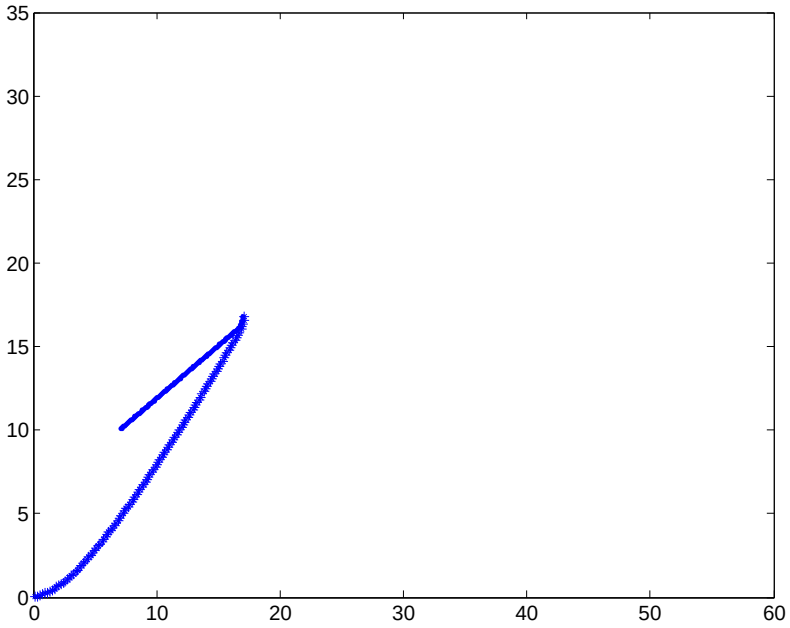


Figure 5.6: The pursuer captures the evader with 100 learning episodes

controller are initialized as shown in Figures 5.1 and 5.2, and tables 5.1 and 5.2.

At the beginning of learning, the pursuer always captures the evader as shown in Figure 5.6. After 500 episodes in Figure 5.7 the evader increased the capture time and made successful maneuvers. Figure 5.8 and table 5.5 show that the evader learned to escape from the pursuer after approximately 900 episodes. The evader makes sharp turns when the distance $d \leq R_p$. The evader avoids capture by changing its direction and entering into the pursuer's turning radius constraint.

The MFs of the evader and the pursuer after tuning the input parameters c and σ are shown in Figure 5.9 and Figure 5.10, respectively. The learners' fuzzy consequence parameters K^l after 900 episodes are shown in tables 5.3 and 5.4.

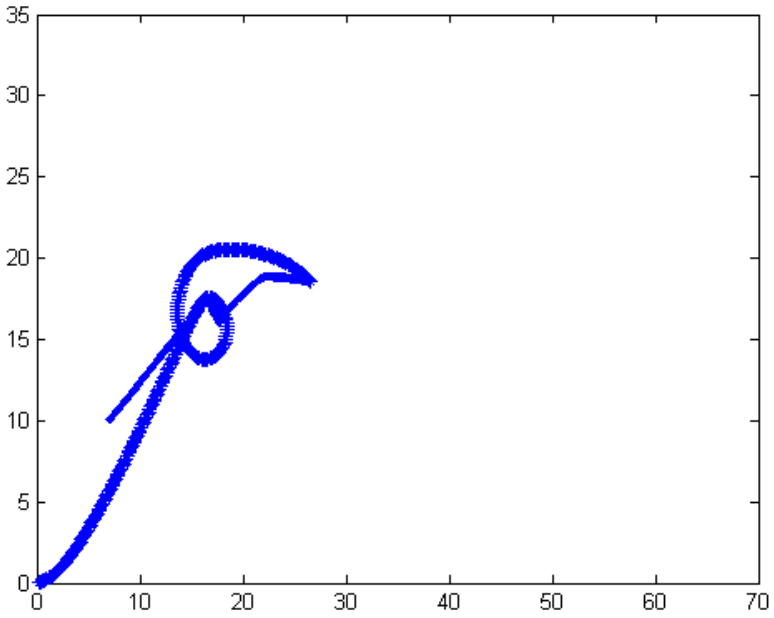


Figure 5.7: The evader increases the capture time after 500 learning episodes

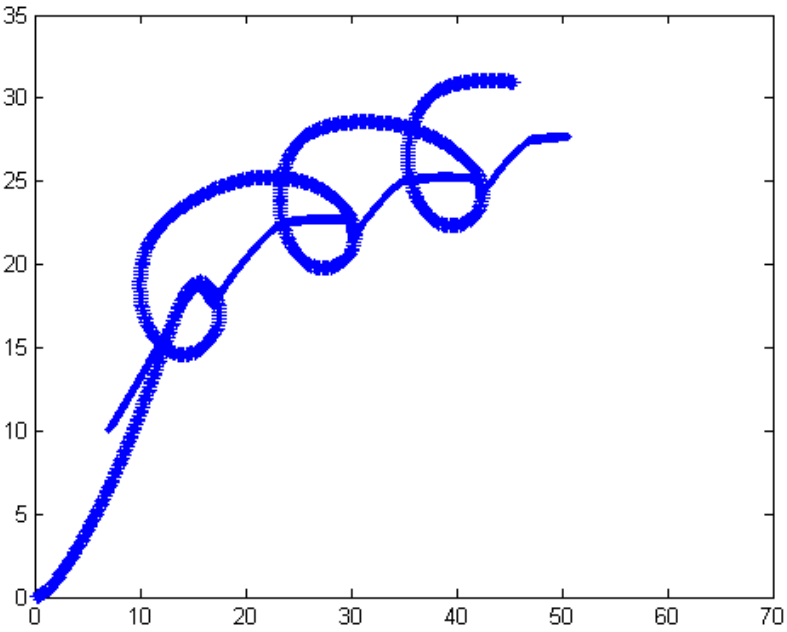
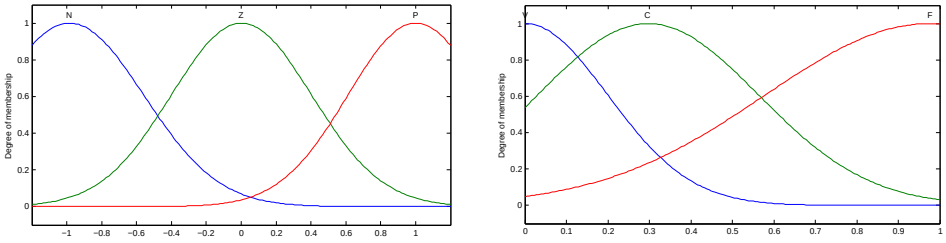


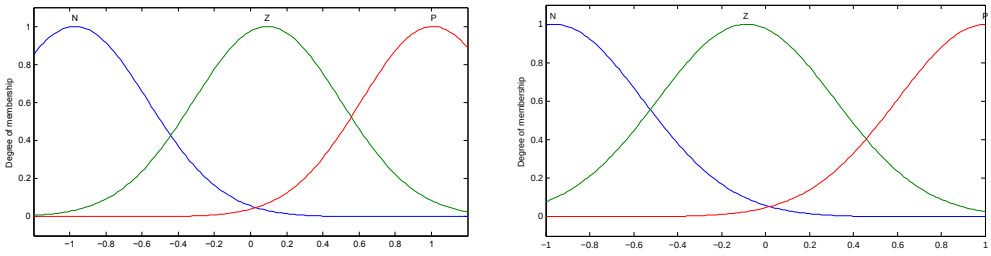
Figure 5.8: The evader learns to escape after 900 learning episodes



(a) The angle difference ϕ

(b) The distance between the pursuer and the evader d

Figure 5.9: The membership functions of the evader after training.



(a) The angle difference ϕ

(b) The rate of change of the angle difference $\dot{\phi}$

Figure 5.10: The membership functions of the pursuer after training

Table 5.3: The evader's fuzzy decision table and the output constant K^l after learning

ϕ \ d	V	C	F
N	-1.585	-1.578	-0.407
Z	-1.576	1.553	0.033
P	1.593	1.580	0.265

Table 5.4: The pursuer's fuzzy decision table and the output constant K^l after learning

ϕ \ $\dot{\phi}$	N	Z	P
N	-0.476	-0.250	-0.008
Z	-0.242	0.002,	0.152
P	-0.005,	0.265	0.478

Table 5.5: This table summarizes the capture time for different number of learning episodes compared to the optimal solution for the homicidal chauffeur game

Game	no. of episodes	Capture time T_c (sec)
Theoretical	–	> 60 (escape)
After learning using QLFIS	100	12.90
	500	25.10
	900	> 60 (escape)

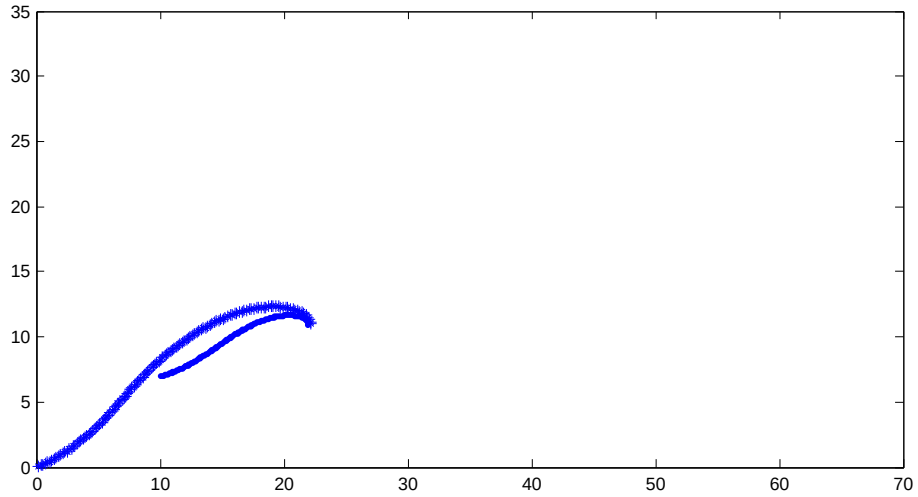


Figure 5.11: The pursuer captures the evader with 100 learning episodes

5.3.3 The Game of Two Cars

The initial position of the pursuer is at the origin $(x_p, y_p) = (0, 0)$, and the position of the evader is initialized randomly at each episode. The initial orientation of the pursuer, and the initial positions are the same as the homicidal chauffeur game. The initial orientation of the evader is $\theta_e = 0rad$. We use the kinematic equations of the pursuer and the evader given in Eq. (2.14).

Similarly, the game is initialized such that the pursuer can capture the evader as shown in Fig.5.11. After 500 episodes in Fig.5.12 the evader increased the capture time and made a successful maneuver. Figure 5.8 and table 5.6 show that the evader learned to escape from the pursuer after approximately 1300 episodes. The evader makes sharp turns to enter into the pursuer's turning radius constraint when the distance $d \leq R_p$.

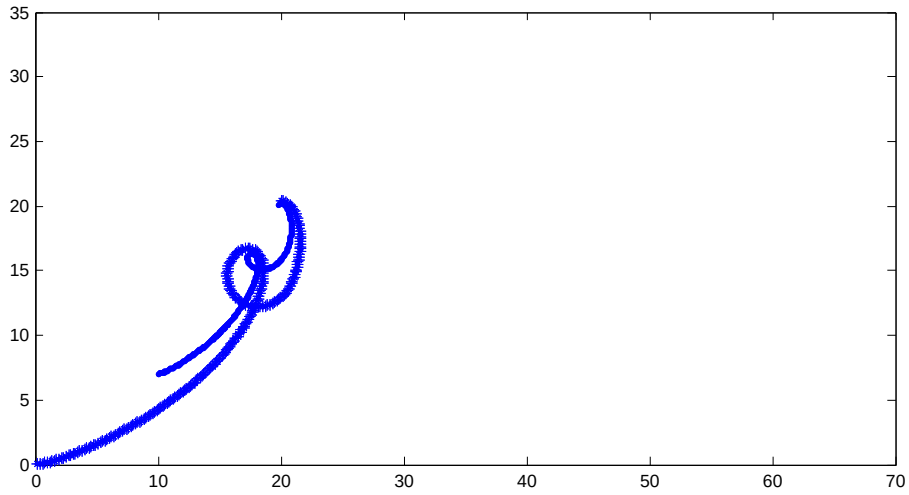
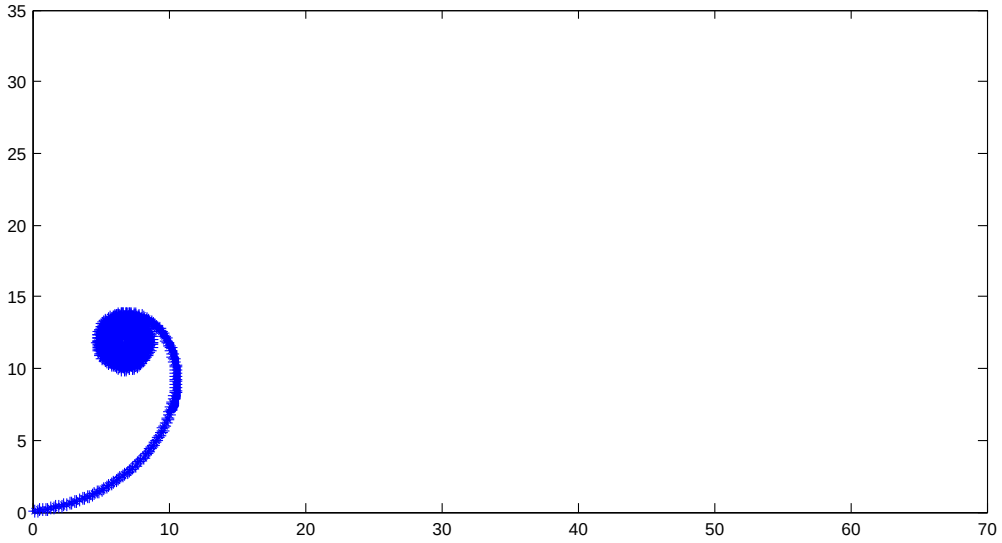


Figure 5.12: The evader increases the capture time after 500 learning episodes

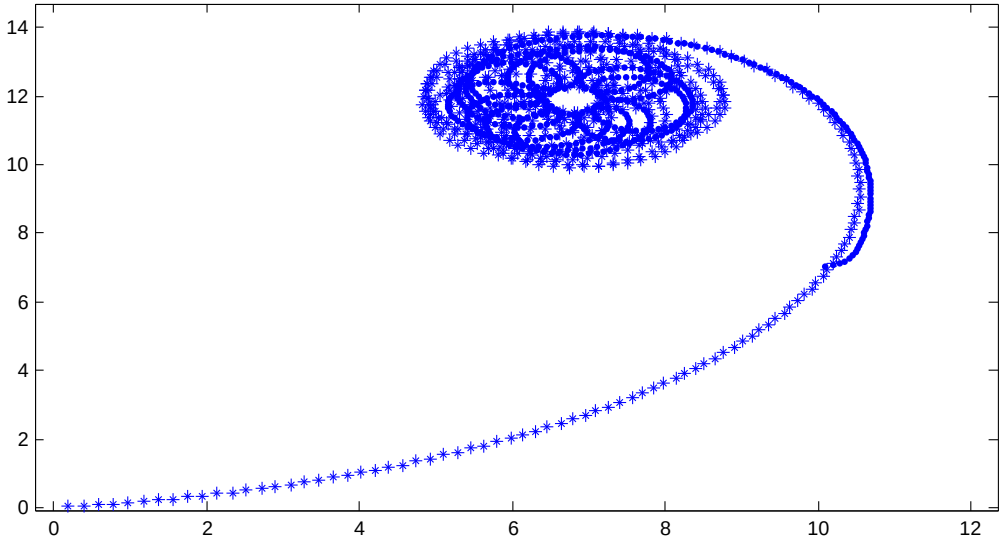
Figures 5.15 and 5.14 show the MFs of the evader and the pursuer after learning. The consequence parameters K^l after training are shown in tables 5.7 and 5.8 .

Table 5.6: Summary of the time of capture for different number of learning episodes in the game of two cars

Game	no. of episodes	Capture time T_c (sec)
Theoretical	–	> 60 (escape)
After learning using QLFIS	100	13.70
	500	27.50
	1300	> 60 (escape)

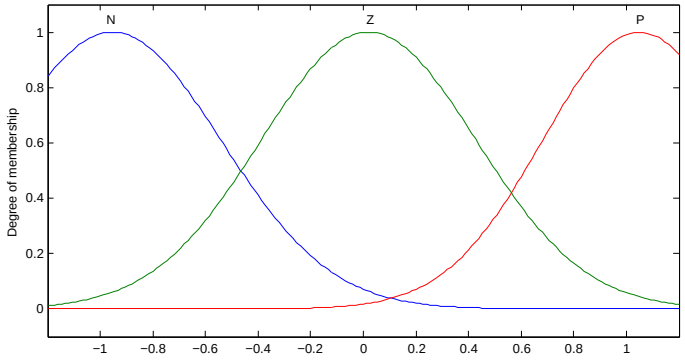


(a) The evader learns to escape after 1300 learning episodes

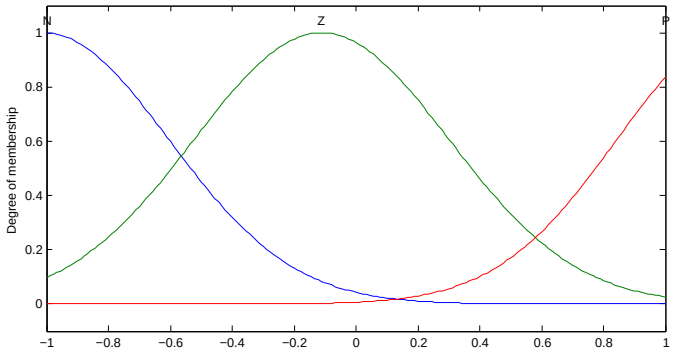


(b) Zoomed version of Figure 5.13(a)

Figure 5.13: The evader learns to escape after 1300 learning episodes.

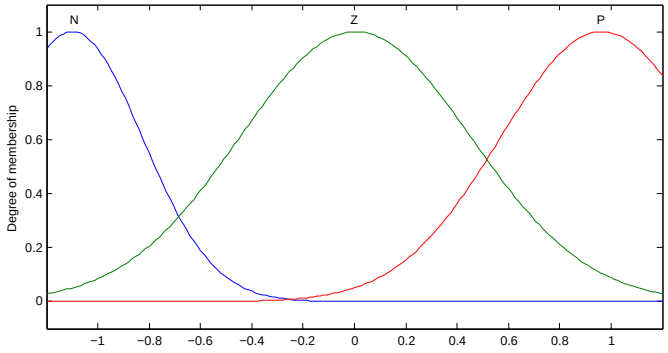


(a) The angle difference ϕ

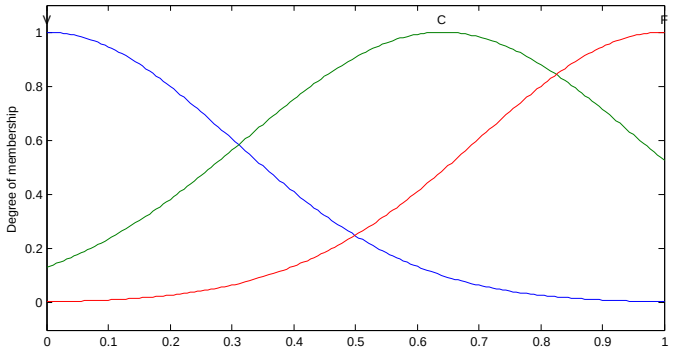


(b) The rate of change of the angle difference $\dot{\phi}$

Figure 5.14: The pursuer's membership functions after training



(a) The angle difference ϕ



(b) The distance between the pursuer and the evader d

Figure 5.15: The evader's membership functions after training.

Table 5.7: The evader's fuzzy decision table and the output constant K^l after learning

ϕ \ d	V	C	F
N	-1.591	-1.572	-0.337
Z	-1.613	1.571	0.146,
P	1.537	1.573	0.429

Table 5.8: The pursuer's fuzzy decision table and the output constant K^l after learning

ϕ \ $\dot{\phi}$	N	Z	P
N	-0.4660,	-0.2512	-0.0005
Z	-0.3507	0.0274,	0.1765
P	-0.0124	0.2615,	0.4830

Without Eligibility Traces. We now apply Q-learning without the use of eligibility traces. In each episode, we record the time of capture and plot the time versus 500 episodes. Then we run 10 simulations and average the result. The solid line in Figure 5.16 is the result from using $Q(\lambda)$-learning. The dash line is the result from using Q-learning. Compared with Q-learning, the learning speed is similar to that with the eligibility traces. The convergence speed of the player's learning process was not improved significantly when using Q(λ)-learning.

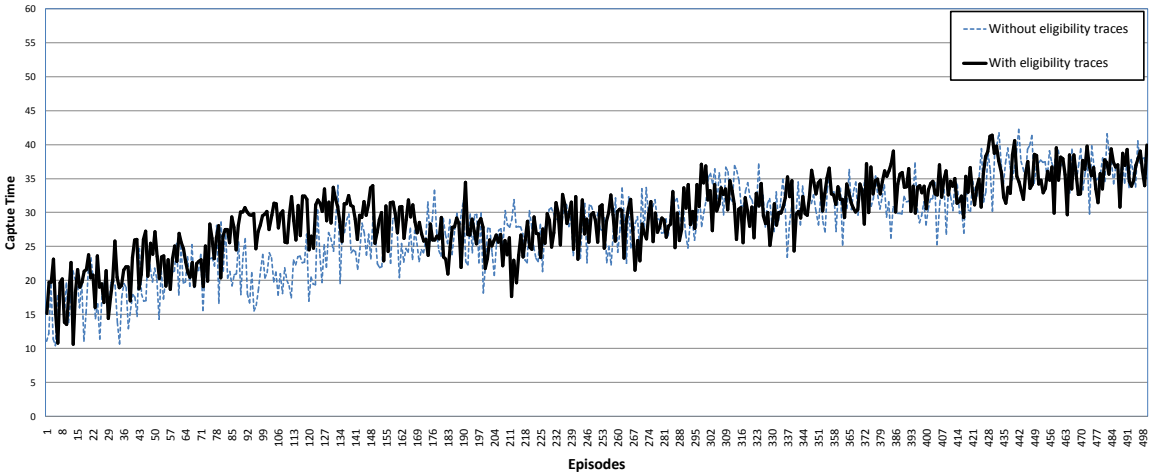


Figure 5.16: The time of capture with the use of eligibility traces in the game of two cars

5.4 Summary

This chapter presented the applications of fuzzy Q(λ)-learning and fuzzy Q-learning to pursuit-evasion differential games. The fuzzy controller, the convergence of the learning process and the learning speed were investigated. The QLFIS technique is then used to train both the evader and the pursuer simultaneously. The trained evader learned to make sharp turns (extreme strategy) to maximize the time of capture and, if possible, avoid being captured. Simulation results of the homicidal chauffeur game and the game of two cars showed that the evader successfully learned to escape from the pursuer. The use of eligibility traces did not significantly improve the learning speed when used in Q(λ)-learning. Moreover, eligibility traces required more computations per episode.

Chapter 6

Conclusions And Future Work

Pursuit-evasion differential games have been studied and solved using various optimization techniques such as optimal control and reinforcement learning. In this thesis, pursuit-evasion differential games and the results of the formal theoretical solutions of optimality are shown in Chap. 2. Chapter 3 illustrated the constructions of fuzzy systems. We also examined the capability of fuzzy systems from a function approximation point of view. Chapter 4 presented the concept of reinforcement learning. Under the framework of reinforcement learning, we presented Markov decision processes, temporal-difference learning and eligibility traces. Q(λ)-learning combines Q-learning with the eligibility traces. The learning speed was tested when the eligibility traces is combined with Q-learning. The use of eligibility traces did not significantly improve the learning speed when used in Q(λ)-learning. Moreover, eligibility traces required more computations per episode and more memory capacity.

In Chap. 5, pursuit-evasion differential games are investigated on how the evader

and the pursuer can learn their optimal strategies simultaneously. The capture condition parameters of the homicidal chauffeur game are set such that there exist a strategy for the evader to avoid capture. The simulation results in Chap. 5 showed that the evader successfully learned its optimal strategy. The trained evader learns to turn and escape from the pursuer after using the QLFIS algorithm to tune the input and the output parameters of the fuzzy controller. The QLFIS technique showed that the players' strategies converges to an equilibrium. When eligibility traces is used in Q(λ)-learning, eligibility traces did not significantly improve the learning speed. Furthermore, eligibility traces can increase the complicity of the learning process and required more computations per episode and more memory capacity. In future research, the learning performance of the evader can be improved by adding more membership functions to the QLFIS algorithm. The evaluation of the QLFIS technique was conducted via computer simulation. Empirical results are needed in the future to evaluate the proposed method. Furthermore, the QLFIS algorithm can be applied to other multi-agent reinforcement learning problems such as the soccer game.

References

[1] Tamer Basar and Geert Jan Olsder. *Dynamic Noncooperative Game Theory.* SIAM, Academic Press, New York, 1999.

[2] R. Bellman. *Dynamic programming.* Princeton University Press, Princeton, NJ, 1957.

[3] Xiaohui Dai, C. Li, and A.B. Rad. An approach to tune fuzzy controllers based on reinforcement learning for autonomous vehicle control. *Intelligent Transportation Systems, IEEE Transactions on*, 6(3):285 – 293, Sept. 2005.

[4] Sameh F. Desouky and Howard M. Schwartz. Q(λ)-learning adaptive fuzzy logic controllers for pursuit–evasion differential games. *Adaptive Control and Signal Processing, International Journal of*, 25(10):910–927, 2011.

[5] S.F. Desouky and H.M. Schwartz. A novel technique to design a fuzzy logic controller using Q(λ)-learning and genetic algorithms in the pursuit-evasion game. In *IEEE International Conference on Systems, Man and Cybernetics, 2009. SMC 2009.*, pages 2683–2689, San Antonio, TX, Oct. 2009.

[6] S.F. Desouky and H.M. Schwartz. Q(λ)-learning fuzzy logic controller for a

multi-robot system. In *IEEE International Conference on Systems, Man and Cybernetics, 2010. SMC 2010.*, Istanbul, Turkey, Oct. 2010.

[7] S.F. Desouky and H.M. Schwartz. Learning in n-pursuer n-evader differential games. In *IEEE International Conference on Systems, Man and Cybernetics, 2010. SMC 2010.*, Istanbul, Turkey, Oct. 2010.

[8] Meng Joo Er and Chang Deng. Online tuning of fuzzy inference systems using dynamic fuzzy q-learning. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(3):1478 –1489, June 2004.

[9] Sidney Givigi, Howard Schwartz, and Xiaosong Lu. A reinforcement learning adaptive fuzzy controller for differential games. *J. Intelligent and Robotic Systems*, 59(1):3–30, July 2010.

[10] S.N. Givigi, H.M. Schwartz, and Xiaosong Lu. An experimental adaptive fuzzy controller for differential games. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 3017 –3023, oct. 2009.

[11] P.Y. Glorennec and L. Jouffe. Fuzzy q-learning. In *Fuzzy Systems, 1997., Proceedings of the Sixth IEEE International Conference on*, volume 2, Jul 1997.

[12] R. Isaacs. *Rand Reports RM-1391 (30 november 1954), RM-1399 (30 november 1954), RM-1411 (21 december 1954), and RM-1486 (25 march 1955), all entitled in part* Differential Games. Rand Reports, 1954,1955.

[13] R. Isaacs. *Differential Games*. John Wiley and Sons, 1965.

[14] Jyh-Shing Roger Jang and Chuen-Tsai Sun. *Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[15] J. Jantzen. *Design of fuzzy controllers*. Technical Report (No:98-E864) Department of Automation, Technical Univ. of Denmark, 1999.

[16] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[17] S. H. Lim, T. Furukawa, G. Dissanayake, and H. F. D. Whyte. A time-optimal control strategy for pursuit-evasion games problems. In *Proceeding. of the 2004 IEEE International Conference on Robotics and Automation*, New Orleans, LA, Apr. 2004.

[18] Xiaosong Lu. An investigation of adaptive fuzzy sliding mode control for robotic manipulators. Master's thesis, Carleton University, Ottawa, ON, Canada, 2007.

[19] A.W. MERZ. The homicidal chauffeur. *AIAA Journal*, 12(3):259–260, March 1974.

[20] M. Pachter. Simple-motion pursuit-evasion differential games. In *Proceedings of the 10th Mediterranean Conference on Control and Automation*, Lisbon, Portugal, July 2002.

[21] Kevin M. Passino and Stephen Yurkovich. *Fuzzy Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1998. ISBN 020118074X.

[22] Timothy J. Ross. *Fuzzy Logic with Engineering Applications*. John Wiley and Sons, Ltd, 2010.

[23] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[24] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modelling and control. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15(1):116–132, Jan. 1985.

[25] Li-Xin Wang. *A course in fuzzy systems and control*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[26] C. Watkins. *Learning From Delayed Rewards*. PhD thesis, Cambridge, 1989.

[27] C. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3): 279–292, May 1992.

[28] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.