Optimal Control for a Two Player Dynamic Pursuit Evasion Game; The Herding Problem

Samy A Shedied

Dissertation submitted to the faculty of Virginia Polytechnique Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
In
Electrical Engineering

Jan 21st, 2002
Blacksburg, Virginia

Keywords; Pursuit Evasion, Shortest Path, Dynamic Programming, Non-holonomic Systems

Optimal Control for a Two Player Dynamic Pursuit Evasion Game; The Herding Problem

Samy A Shedied

(ABSTRACT)

In this dissertation we introduce a new class of pursuit-evasion games; the herding problem. Unlike regular pursuit evasion games where the pursuer aims to hunt the evader the objective of the pursuer in this game is to drive the evader to a certain location on the x-y grid. The dissertation deals with this problem using two different methodologies. In the first, the problem is introduced in the continuous-time, continuous-space domain. The continuous time model of the problem is proposed, analyzed and we came up with an optimal control law for the pursuer is obtained so that the evader is driven to the desired destination position in the x-y grid following the local shortest path in the Euler Lagrange sense. Then, a non-holonomic realization of the two agents is proposed. In this and we show that the optimal control policy is in the form of a feedback control law that enables the pursuer to achieve the same objective using the shortest path.

The second methodology deals with the discrete model representation of the problem. In this formulation, the system is represented by a finite di-graph. In this di-graph, each state of the system is represented by a node in the graph. Applying dynamic programming technique and shortest path algorithms over the finite graph representing the system, we come up with the optimal control policy that the pursuer should follow to achieve the desired goal. To study the robustness, we formulate the problem in a stochastic setting also. We analyze the stochastic model and derive an optimal control law in this setting.

Finally, the case with active evader is considered, the optimal control law for this case is obtained through the application of dynamic programming technique.

Table of Contents

Chapter 1

Pursuit Evasion: Objectives and Technique

1.1. Introduction

Contests of pursuit and evasion are among the most widespread, challenging, and important optimization problems that confront mobile agents, and represent some of the most important potential applications for robots and other artificial autonomous agents. In a typical contest of this sort, a predator chases a prey animal around until the prey is captured. Although pursuit-evasion games have been relatively neglected in research on the simulation of adaptive behavior, they have some major features that render them interesting and relevant [1].

Pursuit and evasion contests are difficult, because dynamic, stochastic, continuous-space, continuous-time or discrete-time discrete-space games are usually difficult to handle. Agents that pursue or evade must maintain complex sensory-motor coordination with respect to both a physical environment and a hostile opponent. Pursuit-evasion contests also require continuous, real-time, dynamical control, in the face of an opponent that will ruthlessly exploit any delay, uncertainty, or error. Natural or artificial behavior-control systems that are slow, brittle, or easily confused do not survive long in pursuit-evasion scenarios. For these reasons, traditional artificial intelligence methods may prove particularly poor as techniques for dealing with pursuit-evasion games.

Pursuit and evasion are scientifically interesting, because the agents evolve against one another in a continuing, open-ended, frequency-dependent way. In addition, since some of the pursuit-evasion scenarios may be so simple, their investigation may illuminate behavioral arms races in more general cases [2]. Further, because effective pursuit may often require prediction and “mind-reading”, while effective evasion may require the use of unpredictable or deceptive tactics [3], such contests raise issues of signaling, communication, and tactical deception [4], [5].

Pursuit-evasion contests have recently received serious attention from at least three scientific disciplines: behavioral biology, neuroethology, and game theory. Animal behavior studies have revealed the ubiquity and importance of pursuit-evasion tactics, anti-predator behaviors, and fighting skills [6]. The central part of such behaviors is revealed by the fact that pursuit-evasion games are the most common form of animal play behavior [7]; such play facilitates learning sensory-motor coordination through “developmental arms races" between play-mates. Neuroethology [8] has spent much effort understanding neural systems for pursuit (approach) and evasion (avoidance), including: explorations of specific circuits for rapid startle and escape behaviors [9], [10]. Game theorists have also studied some classes of pursuit-evasion contests intensely for several decades, because of their importance in tactical air combat (e.g. telling pilots how to evade guided missiles) and other military applications [11]. “Differential game theory" [12] has developed a vocabulary for analyzing the structure and complexity of pursuit-evasion games, and a number of formal results concerning optimal strategies for particular pursuit-evasion games.

The study of pursuit-evasion behaviors has many scientific implications and practical applications. A better understanding of the cognitive dynamics of pursuit-evasion contests would have many applications in robotics, video games, virtual environments, and any other technology where real or simulated mobile agents come into behavioral conflict with other agents.

Finally, understanding pursuit evasion games may open a new area of investigation for simulation of adaptive behavior, and will explore the evolution of pursuit and evasion in a variety of games under various conditions. Also, such work can be extended to investigate:

1. Whether co-evolution between simulated robots engaged in pursuit-evasion contests can lead to more complex pursuit and evasion tactics over generations.
2. Whether the use of continuous recurrent neural networks as control systems allows the emergence of more interesting and dynamic perceptual, predictive, pursuit and evasion abilities.

3. Whether the incorporation of random-activation units in the control system allows the evolution of adaptively unpredictable tactics.
4. Whether changes in the relative physical speed and neural processing speed of pursuers and evaders influences the pursuit and evasion tactics that evolve.

According to Isaacs [12] (Isaacs, 1975), *control theory can be viewed largely as the solution of one-player differential games*; differential game theory addresses the more complex multi-player cases. Considering the above mentioned factors, the most commonly used techniques adapted to deal with pursuit evasion games are:

1. *Classical calculus of Variations and Optimal Control Technique*: This technique provides a very strong tool of analysis and design to the researcher especially when the practical issues of robot dynamics are to be considered in addition to obtaining optimal control policy. The main advantage of this technique is its ability to give a real time solution, if it exists, since the system and the constraints are represented by a set of differential equations. Despite the previously mentioned advantages, this technique has not been widely used in pursuit evasion games due to the complications that arise as the number of players increase. The basic foundations and principal tools using in this techniques will be covered at the beginning of chapter 2.
2. *Dynamic Programming*: A very efficient technique that mainly deals with discrete systems with a value function that needs to be optimized. The basic types and principles of dynamic programming will be covered at the beginning of chapter 4.
3. *Reinforced Machine Learning*: Reinforcement learning [13] is a technique of learning how to map situations to actions---so as to maximize a numerical reward value function. The learner is not told which actions to take, but instead he must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward, but also the next state and, through that, all subsequent rewards. These two characteristics, trial-and-error search and delayed reward, are the two most important distinguishing features of reinforcement learning. Reinforcement learning is defined not by characterizing learning algorithms, but by characterizing a learning *problem*. The basic idea is simply to *capture the most important aspects of the real problem facing a learning*

agent interacting with its environment to achieve a goal [13]. This goal is related to the state of the environment. The formulation is intended to include just these three aspects---sensation, action, and goal---in the simplest possible form without trivializing any of them.

In addition to the agent and the environment, one can identify four main elements to a reinforcement learning system: a *policy*, a *reward function*, a *value function*, and, optionally, a *model* of the environment.

A *policy* defines the learning agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior.

A *reward function* defines the goal in a reinforcement learning problem. It maps the states (or state-action pairs) of the environment to a single number, a *reward value*, indicating the essential desirability of the state. A reinforcement-learning agent's objective is to maximize the total reward it receives in the long run.

A *value function* specifies what is good in the long run. Roughly speaking, the *value* of a state is the total amount of reward an agent can expect to accumulate over the future starting from that state.

4. *Games Theory*: This field took prominence in late 40's. It provides the researchers with tools that enable them to analyze and design optimal control policies for more complicated games. Again, the basic foundation of game theory will be covered later in this chapter.

1.2. Typical pursuit and evasion behaviors

Observing and understanding the animals' behavior in pursuit evasion represents the first step in designing an optimal control policy for any pursuit evasion games. Pursuit is fairly simple: animals are usually observed to move towards the remembered, observed, or predicted location of the target. Evasion is more complex. For example, animal escape behavior in asymmetric pursuit-evasion contests generally breaks down into three phases:

- Directional fleeing if a pursuer is threatening but still distant.
- Erratic zig-zagging if the predator begins catching up.
- Convulsive “death-throes” if caught.

Along with directional fleeing, adjustable escape behaviors are probably the most widespread and successful of all behavioral anti-predator tactics, being used by virtually all mobile animals on land, under water, and in the air [3]. (e.g. because predators' use of search images penalizes common appearances). Indeed, apostatic selection may be a general feature of pursuit-evasion arms races: novel and unexpected tactics may be favored at a variety of levels.

In our study, the first phase of the evasion motion is the key characteristic in our class of pursuit evasion games; The Herding problem.

1.3. Foundations of the Game Theory

Game theory [14] is concerned with the formal analysis of situations called “games" where players can choose different strategies that determine their actions under particular conditions. Conditions and outcomes unfold through the interactions of the players' strategies. Players have preferences among outcomes. In other words a “value function” be present [15].

Briefly, players can be represented by agents that can make choices, employ strategies, and receive payoffs. Traditional game theory has focused on games with discrete moves (e.g. chess), but in the 1950s, Isaacs (1951, 1965) tried to utilize game theory in modeling and analyzing pursuit-evasion situations such as aerial combat, where moves unfold continuously over time, and control systems can vary continuously in the strategies they implement. Isaacs had two basic insights. First, pursuit-evasion contests do require game theory rather than simple optimality theory. However, we show in chapter 2 how to utilize optimality theory in solving for optimal control policy for the herding pursuit evasion game. Second, the continuous nature of pursuit-evasion contests can be modeled using differential equations that specify how state conditions (such as player positions and velocities) change incrementally as a function of players' strategies and previous state conditions: pursuit and evasion moves become continuous trajectories through a state-space. Isaacs (1951, 1965) developed the "*Tenet of Transition*" which specifies that players must optimize (find the minimax solution for) the transitions between states leading towards a goal-state, which can be represented as optimizing the temporal derivatives of the relevant state variables. Applying the tenet of transition, pursuers at each moment in time should try to maximize the rate of their instantaneous approach towards the capture-state, and evaders should try to minimize it. If a solvable set of differential equations can be written that identify the continuous effect of strategies on state-conditions, then the optimal pursuit and evasion strategies can be found by applying the tenet of transition.

Aside from differential game theory, there are also large and relevant literatures on using control theory [16], to deal with some classes of pursuit evasion games such as; missile guidance, aircraft control, aerial tactics, and sports tactics.

Differential pursuit-evasion games are defined by a set of controls (what each player can do), a set of dynamics (that can be considered as a mapping from the control variables onto the state variables of the game, and from state variables at one moment in time to the next moment), and a set of termination conditions (state conditions that determine when successful capture or evasion happens). For example, in a classic case analyzed by Berkovitz [17], a pursuer and an evader move with equal and constant speed in a plane, and control the direction of their velocity vector (which thus becomes their control variable). These two velocity vectors give rise to a system of first-order differential state equations that determine how the players move over time. The pursuer wants to minimize time to capture the evader and the evader wants to maximize time until capture, with capture defined as proximity within some small distance. Both players know the present state of the game (e.g. both of their positions and velocity vectors) but at each time-point they make separate and simultaneous decisions about what to do next. The available strategies are therefore functions that map from current states of the game (i.e. the positions and velocity vectors of both players) onto velocity-vector decisions about what direction to move next. In all differential games, strategies determine trajectories through the relevant state-space; in pursuit-evasion games, strategies determine trajectories through physical space.

1.4. The optimality of mixed strategies

The key to formal analysis in game theory is for games to be reduced from descriptive form (e.g. rules and heuristics) or "extensive form" (i.e. decision-tree form) to "normal form" (i.e. a joint payoff matrix that lists game outcomes given all possible strategies for all players). Some games in normal form have "minimax solutions" (a.k.a. "saddle points") that minimize each player's expected payoff "value function" apart from what the opponent does to maximize their expected gain.Minimax solutions, if they exist, are jointly optimal for both players. In games of perfect information, players are accurately and continuously aware of all moves made by other players, so that deception, confusion, and uncertainty are impossible. *All games of perfect information have one or more saddle points corresponding to "pure" deterministic optimal strategies*.

However, games of imperfect information (e.g. games where deception is possible) may have multiple saddle points or no saddle points. In such cases, "*mixed strategies*" (probability distributions across pure strategies) may be optimal. Perhaps the most important and interesting result from von Neumann and Morgenstern (1944) was that *every two-player, zero-sum game of incomplete information with multiple saddle points has an optimal strategy that is mixed rather than pure*:

"*One important consideration for a player in such a game is to protect himself against having his intentions found out by his opponent. Playing several different strategies at random, so that only their probabilities are determined, is an effective way to achieve a* *degree of such protection. By this device the opponent cannot possibly find out what the player's strategy is going to be, since the player does not know it himself. Ignorance is obviously a good safeguard against disclosing information directly or indirectly.*" [14].

The logic of mixed strategies is simple [18]. If a player's choice sometimes remains unknown to others after the move is made, the game is one of imperfect information. This can result from the move being unknown, or the other players' sensors being insufficient to detect all moves with complete accuracy. Typically, games lose their saddle points when they are no longer games of perfect information, such that the first player's minimax solution does not correspond to the second player's minimax solution. In general, mixed strategies randomize moves to confuse opponents and keep them guessing. (But the task of determining the optimal mixed strategy is usually very difficult for games with many pure strategies and complex interactions.)

Because many pursuit-evasion games are ones of incomplete information with multiple saddle points, mixed strategies have often proven useful in such games. Mixed strategies are optimal for a pursuit-evasion game with rectilinear movement on a planar grid [19]. Such game-theoretic results support the assumption of Driver and Humphries (1988) that erratic zig-zagging by animals is truly stochastic behavior that derives its utility from its unpredictability. We might expect then that in any pursuit-evasion game with incomplete information and complex dynamics, unpredictable pursuit and evasion strategies will evolve.

Dynamic programming methods [20] may prove more useful in analyzing pursuit-evasion contests, since they can optimize stochastic dynamic strategies, even in two-player games [21].

1.5. Reasons to simulate pursuit-evasion games

Generally speaking, games are characterized by various dimensions of complexity:

1. The number of players, ranging from one-player cases to classic two-player cases to more difficult multi-player cases.
2. The number of moves, ranging from “static’ games of one discrete move per player (e.g. Rock, Paper, Scissors) to games with multiple discrete moves per player (e.g. chess), to differential games with continuous moves (e.g. air combat).
3. The continuity of the strategy space, with discrete spaces being simpler than continuous spaces.
4. The payoff “value function” structure, with zero-sum games usually being simpler than non-zero-sum games.
5. The information structure, with games of complete information being much simpler than games of incomplete information. Generally, anything that complicates the differential state equations complicates the game analysis.
6. Finally, Solving for optimal strategy “policy” requires the complete specification of a strategy space. Such a complete specification may not be possible if the strategies are evolving properties of human heuristics, animal brains, or advanced robot control systems, and if the emergent strategies can vary continuously along a number of dimensions.

These problems propose that differential pursuit-evasion games are complicated to analyze even under the best circumstances, and that the introduction of realistic complexity makes most of them formally inflexible.

To avoid these complexities, differential game theory usually assumes that the pursuit-evasion game is one of perfect information between two players with fixed and pre-determined roles (one “pursuer” and one “evader”), deterministic dynamics and constant speeds, and a zero-sum payoff structure. Mathematically proficient researchers can relax one or two of these assumptions at a time to obtain results for special and simplified cases, but relaxing all the assumptions at once makes the game hopelessly complex. Yet even with bounded uncertainties in dynamics, the classical game-theoretic concepts of optimality, value, and saddle point may be inappropriate (Galperin & Skowronski, 1987).

Pursuit-evasion games that cannot be reduced to differential state-space equations can not be analyzed using the traditional methods of differential game theory. A recent complexity-theoretic analysis of differential pursuit-evasion games by (Reif&Tate,1993) illustrates the difficulties of designing a control systems for robots and autonomous vehicles playing such games.

Another important assumption, rarely mentioned in game theory “meanwhile it can be considered in control theory”, is that strategies can be implemented instantaneously, without time-lags. That is, decision dynamics are assumed to be much faster than system dynamics. For robots, this assumption is far from being realistic.

In recognition of these problems, some game theorists have recently shifted to numerical and simulation methods to derive near-optimal strategies for more complex pursuit-evasion games. For example, Rodin et al. (1987) [22] used artificial intelligence (AI) methods to simulate players in an air combat maneuvering scenario. Each player derives tactical maneuvers using a world-model based on sensor inputs, an inference engine linked to a database (containing player parameters and capabilities and an environment model), and a knowledge base (containing a basic set of pursuit-evasion algorithms). The inference engine updates tactical plans every time an opponent's actual trajectory deviates from its expected trajectory. Clearly, unexpected behavior increases problem complexity and processing time. But such AI methods for controlling autonomous agents tend to become desperately slow as the dynamics of agents and environments become more complex and noisy. We need simulation methods that yield reactive, robust, dynamic pursuit-evasion strategies, rather than slow, brittle, hand-designed AI systems.

Although differential game theory provides a framework for describing the important features of pursuit-evasion contests, and a set of normative results concerning optimal strategies in simple cases, it cannot generally provide optimal strategies for practically pursuit-evasion problems, nor can it show how strategies can be implemented in a real control system subject to limited sensory capacities, sensory and motor noise, component failure, and constraints on processing speed and accuracy.

1.6. Review of Some Previous Simulation and Robotics Work

Pursuit-evasion is embedded in much of the recent work in artificial life and simulation of adaptive behavior. Classic problems of obstacle avoidance and navigation can be viewed as relaxed special cases of evasion and pursuit, respectively, with the "opponents" represented by, non-moving obstacles, or a trajectory "path" to follow. In addition to the work introduced above on pursuit evasion, the following paragraph we'll give a brief description of some recent works made on pursuit evasion game in the recent years.

1. In [23] Joao et al, introduced a probabilistic framework of pursuit evasion game simulation where a swarm of autonomous pursuers are chasing an evader, and the objective is to come up with a policy that will maximize the probability of finding the evader in finite time.
2. In [24] a model for an active evader chased by several pursuers in a non-precisely mapped region is presented. Instead of solving the problem in two phases, region-mapping estimation and then solving a deterministic pursuit-evasion game over the estimated region map assuming that the map is determined accurately, the inaccuracy of the map information is merged with the players movements in one stochastic partial information Markov game.
3. The work presented in [25] discusses the optimal escape policy of an aircraft from an optimally guided missile. The solution is obtained by decomposing the minimax value function into two separate parts. One part for the maxmizer and the second for the minimizer. The two parts are iteratively solved alternatively. Solution for both sub-problems is obtained by appropriate application of discretization and non-linear programming techniques.
4. Boris Stilman [26] introduced a heuristic search algorithm based on discretizing time appropriately to produce a finite game tree of finite number of states. This tree of all possible states of the game is searched for the optimal trajectory.

1.7. Summary and Dissertation Overview

The main features of the pursuit evasion games obtained form the literature search presented above can be summarized in the following points:

1. Pursuit evasion games vary in complexity ranging from avoiding (evading) stationary targets (obstacle avoidance) and navigating, or following moving target, to the cooperative effort of a group of pursuers to catch a single evader.
2. Pursuit evasion games studies so far concentrated on hunting the evader rather than directing it to a certain predetermined position in the coordinate grid.
3. The games always begin with one initial state and ends with one final state (capturing).
4. Despite the fact that differential game theory provide the framework for describing the important features of pursuit evasion contests, it can not generally provide realizable optimal strategies nor can it illustrate how to implement these strategies in real control system subjected to constrained dynamics.
5. Implementing and applying optimal strategies on real robots taking into consideration their constrained dynamics was rarely considered in pursuit-evasion games.

In this dissertation, we try to cover some of the points that have not been represented in research so far as mentioned above. A moderately complex model consisting of two players in a dynamic game is considered. The objective of the pursuer is to drive the evader to certain predetermined location rather than intercepting or hunting it. This modified objective results in an additional difficulty of having multiple final states instead of having one terminal state. Moreover, practical considerations of robotic realization of the pursuer's optimal policy are taken into consideration while designing the optimal control law .

The dissertation can be divided into 2 main parts. The first part includes chapters 2 and 3 where a continuous time, continuous space model of the problem is introduced. In chapter 2, the continuous time continuous space model is introduced as an optimization problem where the pursuer objective is to drive the evader to the pen, without loss of generality in the neighborhood of the origin, through shortest path possible. The system equations with the detailed derivation of the optimal trajectory are given the some supporting sample results of the model simulation

In chapter 3, a rarely considered realization of the pursuit evasion game is introduced. In this realization both the evader and the pursuer are represented by wheeled mobile robot. Although this adds some more constraints on the system dynamics, it enables the researcher to convert the optimal control policy obtained analytically to be practically implementable. The chapter begins by introducing the basic definitions and concepts of the non-holonomic constrained systems with the basic theories used to study and analyze their properties. After that, the non-holonomic constrains are added to the system model introduced in chapter 2. Taking into consideration the non-holonomic constraints, the optimal control law is derived and some supporting simulation results are given at the end of the chapter.

Part two of the dissertation deals with different frameworks of the discrete-time discrete space model. Due to this discrete nature of the model, dynamic programming is used as the optimization technique in this case due to its simplicity and applicability to problems of this kind. Part two begins with introducing the basic foundation and principles of dynamic programming. Then, a basic deterministic model with passive evader is presented. Due to the finite number of states of the system of the discrete model, our system can always be represented by a finite graph. The objective of the pursuer is to search this finite graph for the shortest path beginning from any initial state to the final destination state. As a result of the new objective of the pursuer, our model has multiple final destination states which motivated us to modify some of the most commonly used algorithms used to solve for shortest path problem in finite graphs. We introduced three ways to solve for the shortest path in the finite graph representation model. The first solution approach uses dynamic programming technique directly while the other two techniques depend on modifying the famous Dijkstra's algorithm for shortest path over finite graphs. The simulation results of all the three techniques used for solving for the shortest path are given at the end of the chapter.

In chapter 5, a degree of uncertainty is added to the system model. The uncertainty introduced is in the state transition, rather than in the link costs as usually considered. This introduced uncertainty adds more difficulty to the calculations of the state cost value and hence to the evaluation of the optimal control policy. The state cost values are calculated by three techniques; which are admissible policy search technique, the value iteration technique, and the policy iteration technique. At the end of the chapter, the supporting simulation results are presented.

In chapter 6 we consider the deterministic case of an active evader, where the evader in this model is not only trying to avoid the pursuer, but also is to maximizing the value of its state cost.. Similar to the deterministic passive evader model, solution for the shortest path over the finite graph representing the problem is provided using the same techniques given in chapter 4.

Finally, a conclusion of the work covered, contributions made in the dissertation, future directions for work and some possible applications are given.

Chapter 2

Optimal Trajectory for a Class of Pursuit Evasion Games: The Continuous Time Herding Problem

2.1. Introduction

Pursuit evasion problems have been studied and solved using various optimization techniques such as dynamic programming [27-29], calculus of variations, optimal control [30-31], and reinforced machine learning [32]. In most of the pursuit evasion models considered so far, if not all of them as shown previously in chapter 1, the pursuer's aim is to hunt, or intercept the evader. Unlike these previously introduced models, the work presented in this chapter uses a different view. The aim of the pursuer in our case is to drive the evader to a certain location in the x-y grid.

This chapter begins with introducing the basic concepts and theories of calculus of variations that we are going to utilize in developing our optimal trajectory. Then, the model of the problem is introduced in section 2.2. The solution for the optimal trajectory for the pursuer is derived and it is illustrated in section 2.3 how it satisfies the necessary and sufficient conditions for a minimizing curve. Finally, simulation results for the system are given for different initial condition.

2.2. Basic Principles and Theories of Calculus of Variations

Calculus of variations is the science used to study optimization of an objective *functional* subjected to a set of constraints. The basic definitions, concepts, and theories that we are going to use in solving for our optimal trajectory, are briefly covered briefly in this section. The definitions are adapted from U. Berchtken-Manderscheid "*Introduction to the Calculus of variations*" [33].

2.2.1. *Definitions and Concepts*

Functional: is defined as a function of functions.

Thus, given a functional of an integral form;

$$J = \int_{x(t_0)}^{x(t_f)} L(t, x(t), \dot{x}(t))\, dt \tag{2.1}$$

we would like to find the function, $x_*(t)$, from the set of all *admissible functions*, $x(t)$, that will optimize (maximize or minimize) the value of *J* subjected to certain end point constraints.

By the set of *admissible functions* here we mean the set of all smooth functions that satisfy the end points constraints.

The integrand, $L(t, x(t), \dot{x}(t))$, of equation 2.1 is assumed to satisfy the following conditions;

-*L* is defined for all points $(t, x(t), \dot{x}(t)) \in [0, \infty) \times R^2$.

-*L* is at least twice continuously differentiable, i.e. *L* is continuos and has continuous partial derivatives with respect to its variables.

2.2.2. Existence of Solutions

Prior to going into the details of the necessary and sufficient conditions that a function, $x_*(t)$, should satisfy to minimize (maximize) the value of the objective functional ***J***, it is important to first check for the existence of the solution itself. According to the Weierstras theorem; "*Every continuos function f that is defined on a closed and bounded domain* $\boldsymbol{M} \subset \boldsymbol{R}^n$ *has a minimum and a maximum*". This means that the existence of a solution for any optimization problem is related to the domain of operation. Therefore, the domain of operation has to be appropriately selected or modified such that the objective functional ***J*** is guaranteed to have an optimizer.

2.2.3. Necessary Conditions

Before introducing the first necessary condition that an optimizing function $x_*(t)$ has to satisfy, we introduce first the definition of a "*weak*" and "*strong*" solution.

***Definition* 2.1**: the function $x_*(t)$ is said to provide a *strong solution* of the variational problem given in 2.1 if there is an ε>0 such that $\boldsymbol{J}(x_*(t)) \leq \boldsymbol{J}(x(t))$ for all admissible functions $x(t)$ with:

$$d_0(x(t), x_*(t)) = \max_{x \in [x(t_0), x(t_f)]} \left| x(t) - x_*(t) \right| < \varepsilon$$

***Definition 2.2*:** the function $x_*(t)$ is said to provide a *week solution* of the variational problem given by 2.1 if there is an ε>0 such that $\boldsymbol{J}(x_*(t)) \leq \boldsymbol{J}(x(t))$ for all admissible functions $x(t)$ with:

$$d_1(x(t), x_*(t)) = \sup_{x \in [x(t_0), x(t_f)]} \left(\left| x(t) - x_*(t) \right| + \left| x'(t) - x_*'(t) \right| \right) < \varepsilon$$

Now, we are ready to state the first necessary condition that any admissible optimizing function $x_*(t)$ has to satisfy;

Euler's Necessary Condition: *If the function* $x_*(t)$*, which has continuous first, second and third order derivative, is a candidate minimizer (maximizer) "strong or weak" for the variational problem given in* (1)*, then it has to satisfy Euler's Differential Equation given by:*

$$L_x(t, x_*(t), x'_*(t)) - \frac{d}{dt} L_{x'}(t, x_*(t), x'_*(t)) = 0 \qquad (2.2)$$

Therefore, Euler's necessary condition provides a primary means that can be used to solve for the set of extremals of the variational problem given in (2.1) regardless of this extremal being weak or strong one.

In addition to Euler's necessary condition, Legendre condition gives a simpler and yet easier necessary condition that $x_*(t)$ has to satisfy.

Legendre Necessary Condition: If $x_*(t)$ *is a smooth function that provides a weak solution of the variational problem given by equation* (2.1) *then,* $\forall\ x(t) \in [x(t_0), x(t_f)]$;

$$L_{x'x'}(t, x_*(t), x'_*(t)) \geq 0 \qquad (2.3)$$

In addition to the above-mentioned necessary conditions, a more powerful necessary condition is given by Weirstrass. The difference between Euler's, Legendre and Weirstrass's necessary conditions is that Euler's can't differentiate between week or strong minmizers (maximizers), Legendre's is used for weak extremals, while Weirstrass's necessary condition provides a tool to check if the minimizer (maximizer) is a strong one.

Weierstrass's necessary conditions: *The function* $x_*(t)$ *provides a strong solution to the variational problem given in* (2.1) *if at any point* $x(t) \in [x(t_0), x(t_f)]$ and $\forall q \in$ R, the Weierstrass Excess function $\mathcal{E}$,

$\mathcal{E}(\mathrm{t}, x_*(t), x'_*(t), q) \geq 0$

Where;

$$\mathcal{E}(\mathrm{t}, x_*(t), x'_*(t), q) = L(t, x_*(t), q) - L(t, x_*(t), x'_*(t)) - (q - x'_*(t))L_{x'_*}(t, x_*(t), x'_*(t)) \qquad (2.4)$$

2.2.4. Sufficient Condition

So far, the previously mentioned necessary conditions give no way to check if any member of the set of extremals resulting form solving Euler's differential equation is a solution of the variational problem given by (2.1) or not. In this section we bring in a theorem that can deal with a special class of variational problems of the form given by equation (2.1), but the integrand L is *convex*.

First, we have to define what is meant by convex function then we'll present the theorem afterwards.

Definition* 2.3:** A function *f* defined on a convex set is said to be convex ***M if $\forall\ u_1, u_2 \in \boldsymbol{M}$ and all $q \in (0,1)$, it holds that;

$$f(qu_2 + (1-q)u_1) \leq qf(u_2) + (1-q)f(u_1)$$

Another easy way to check for the convexity of a function *f* is given by the following theorem.

***Theorem* 2.1:** *Let function f*: $\boldsymbol{M} \subset \mathbf{R}^2 \rightarrow \mathbf{R}$ *that have continuos first and second partial derivatives, then f is convex if the entire Hessian matrix is positive semi-definite.*

Finally, we can state the theorem about the sufficient condition for the solution of problem 1 to exist.

***Theorem* 2.2:** *If the integrand* **L** *of the variational problem* (2.1) *is convex* $\forall t$ *and with respect to the variables* ($x(t)$, $x'(t)$), *then a smooth function* $x_*(t)$ *that satisfies Euler's differential equation is a solution of this problem.*

Using the previous theorems and following the same order, we introduce our system model and make use of them to come up with the optimal trajectory of our game. Then, we will show that the obtained trajectory satisfies the necessary conditions required for a minimizing curve.

2.3. System Model

Figure 2.1 gives a quick summary representation of our model. As shown in the figure, the intial position of the pursuer is (x_{p0}, y_{p0}) and that of the evader is (x_{e0}, y_{e0}). Beginning at these initial positions, the pursuer is supposed to drive the evader to the (0,0) position in the x-y grid through the shortest path. The associated dynamics of the

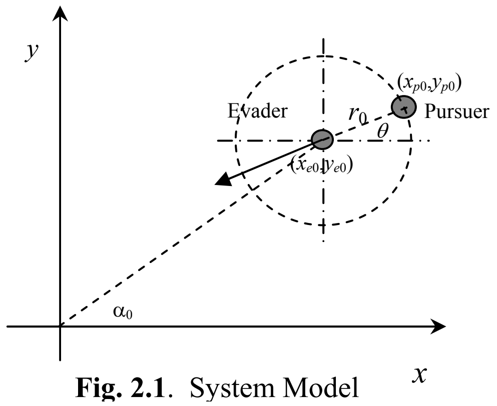


Fig. 2.1. System Model

problem are given below:

$$\dot{y}_e = \dot{x}_e \tan\theta \tag{2.5}$$

$$\dot{x}_e^2 + \dot{y}_e^2 = 1 \tag{2.6}$$

$$\begin{aligned} x_p &= x_e + r_0 \cos\theta \\ y_p &= y_e + r_0 \sin\theta \end{aligned} \tag{2.7}$$

$$\begin{aligned} \int_0^{t_f} \dot{x}_e \, dt &= -x_{eo} \\ \int_0^{t_f} \dot{y}_e \, dt &= -y_{eo} \end{aligned} \tag{2.8}$$

Based on the above dynamic equations of the system, the evader moves away from the pursuer according to equations (2.8) and (2.6), that show that the direction of the motion of the evader is in the straight line joining the pursuer and the evader. We have chosen the normalized velocity of the evader to be 1 unit. Notice that the distance between the two agents is always the same and equals r_0. The goal of the pursuer is to drive the evader form the given initial position to the final one, following the above constraint, such that a norm characterizing distance traveled by both the pursuer and the evader is minimized. The previous statement can be represented by the following objective function to be minimized:

$$J = \min \int_0^{t_f} \dot{x}_e^2 + \dot{y}_e^2 + \dot{x}_p^2 + \dot{y}_p^2 \, dt \tag{2.9}$$

Thus, equations (2.5), (2.6), (2.7), (2.8), and (2.9) give a complete representation of the problem at hand with the constraints that should be satisfied $\forall t \in [t_0, t_f]$.

2.4. Solving for Optimal Trajectory

Examining figure 1, with the given system equations, shows that the main control variable that the pursuer can use to achieve the objective is its relative position angle θ. This, in turn can be directly controlled via its rate of change. Therefore, the first step in obtaining the optimal trajectory is to express the objective function as well as the system constraints as functions of θ. By substituting (2.8) in (2.6), we get

$$\dot{x}_e^2 + \dot{x}_e^2 \tan^2\theta = 1 \tag{2.10}$$

Manipulating and simplifying equation (2.10) results in:

$$\begin{aligned} \dot{x}_e^2 &= \cos^2\theta \\ \dot{y}_e^2 &= \sin^2\theta \end{aligned} \tag{2.11}$$

Differentiating equation (2.7) and using (2.11) gives:

$$\begin{aligned} \dot{x}_p &= \dot{x}_e - r_0\,\dot{\theta}\sin\theta \\ \dot{y}_p &= \dot{y}_e + r_0\,\dot{\theta}\cos\theta \end{aligned} \tag{2.12}$$

By using equations (2.11) and (2.12), and substituting in equation (2.9) of the objective function, the integrand, *L*, becomes

$$L = \dot{x}_e^2 + \dot{y}_e^2 + \dot{x}_p^2 + \dot{y}_p^2$$

$$\begin{aligned} L &= 1 + (\dot{x}_e - r_0\,\dot{\theta}\sin\theta)^2 + (\dot{y}_e + r_0\,\dot{\theta}\cos\theta)^2 \\ &= 1 + 1 + (r_0\,\dot{\theta})^2 + 2r_0\,\dot{\theta}[\pm\sin\theta\cos\theta - (\pm\cos\theta\sin\theta)] \end{aligned}$$

$$L = 2 + r_0^2\,\dot{\theta}^2 \tag{2.13}$$

Therefore, the original model of the problem can be transformed into the following equivalent one.

$$J = \min\int_0^{t_f} 2 + r_0^2\,\dot{\theta}^2\,dt \tag{2.14}$$

Subjected to the following constraints

$\theta(0) = \theta_0$ $\qquad\qquad$ $\theta(t_f) = \theta_f$

where $\theta_0 = \tan^{-1}\left(\frac{(y_{p0} - y_{e0})}{(x_{p0} - x_{e0})}\right)$

and $\theta(t_f)= \theta_f$ is free. The vertical and horizontal components of the evader's velocity have to satisfy;

$$\int_0^{t_f} \pm \cos\theta \; dt = x_{e0}$$
$$\int_0^{t_f} \pm \sin\theta \, dt = y_{e0} \qquad (2.15)$$

Clearly, the Lagrangian of equation (2.14) satisfies all the requirements of theorem 2.2 since we have;

$$L_{\dot{\theta}\dot{\theta}} = 2r_0^2 \geq 0$$

Therefore, the solution of the Euler's differential equation provides a solution of (2.14). Combining the constraints given by (2.15), the problem model becomes an isoperimetric model whose Lagrangian is given by;

$$L = 2 + (r_0 \dot{\theta})^2 + \lambda_1 \sin\theta + \lambda_2 \cos\theta \qquad (2.16)$$

Assuming that θ_* is the optimizer of the equation (2.16), it has to satisfy the following Euler-Lagrange differential equation [33].

$$L_\theta - \frac{\partial}{\partial t} L_{\dot{\theta}} = 0 \qquad (2.17)$$

Therefore,

$$2{r_0}^2 \ddot{\theta}_* = -\lambda_2 \sin\theta_* + \lambda_1 \cos\theta_* \qquad (2.18)$$

Where, θ_* is the optimal angle of the pursuer with respect to the evader position at any time *t*. Since the final angle of arrival, θ_f, is free, the transversality condition has to be satisfied at the final time [34]; i.e.

$$L_{\dot{\theta}} = 0 \;\; at\, t_f$$

Since, the control variable $\dot{\theta}$ is completely state dependent and has no explicit dependence on time, we can use

$$v = \dot{\theta} \;\; \Rightarrow \ddot{\theta} = v\frac{dv}{d\theta} \qquad (2.19)$$

and therefore,

$$\dot{\theta}_*(t_f) = 0 \qquad \Rightarrow \ddot{\theta}_*(t_f) = 0 \tag{2.20}$$

From equation (2.18) and (2.19), we get;

$$\lambda_1 \sin\theta_f = \lambda_2 \cos\theta_f$$

$$\frac{\lambda_2}{\lambda_1} = \frac{\sin\theta_f}{\cos\theta_f} \tag{2.21}$$

Substituting equation (2.21) into equation (2.18) and after some trigonometric manipulations we get;

$$\ddot{\theta}_* = \frac{1}{2r_0^{\,2}}\sin(\theta_* - \theta_f) \tag{2.22}$$

Using (2.19) in (2.22) gives

$$\dot{\theta}_*^{\,2} = -\frac{1}{r_0^{\,2}}\cos(\theta_* - \theta_f) + C \tag{2.23}$$

By equation (2.19), we can easily see that;

$$C = \frac{1}{r_0^{\,2}}$$

which gives us

$$\dot{\theta}_*^{\,2} = \frac{2}{r_0^{\,2}}\sin^2(\frac{\theta_* - \theta_f}{2}) \tag{2.24}$$

Solution of the differential equation (2.24) gives the family of all extremal curves. Hence, checking for Legendre necessary condition, we find that;

$$L_{\dot{\theta}\dot{\theta}} = 2r_0^2 > 0$$

This means that the solution obtained by solving Euler's differential equation provides a weak minimum of the objective function ***J***.

According to the Weiestrass condition [35], in order for that extremal θ_* to give a strong minimum of the objective function J, it is sufficient that

θ_* is a member of a field of extremals "which is satisfied from equation (2.24)".

$E(t,\ \theta, \dot{\theta}, p) \geq 0$

where,

$E(t,\ \theta, \dot{\theta}, p) = L(t,\ \theta, \dot{\theta}) - L(t,\ \theta, p) -\ (\theta - \dot{\theta})\partial\ L(t,\ \theta, p)/\partial p.$

To satisfy the Weiestrass condition we notice that

$$\begin{aligned}
E(t,\theta,\dot{\theta},p) &= 2+(r_0\,\dot{\theta})^2-2-(r_0 p)^2-(\dot{\theta}-p)(2r_0^{\,2}p)\\
&= 2+(r_0\,\dot{\theta})^2-2-(r_0 p)^2-2r_0^{\,2}p\,\dot{\theta}+2r_0^{\,2}p^2\\
&= (r_0\,\dot{\theta})^2-2r_0^{\,2}p\,\dot{\theta}+(r_0 p)^2\\
&= (r_0\,\dot{\theta}-r_0 p)^2\\
&\geq 0
\end{aligned}$$

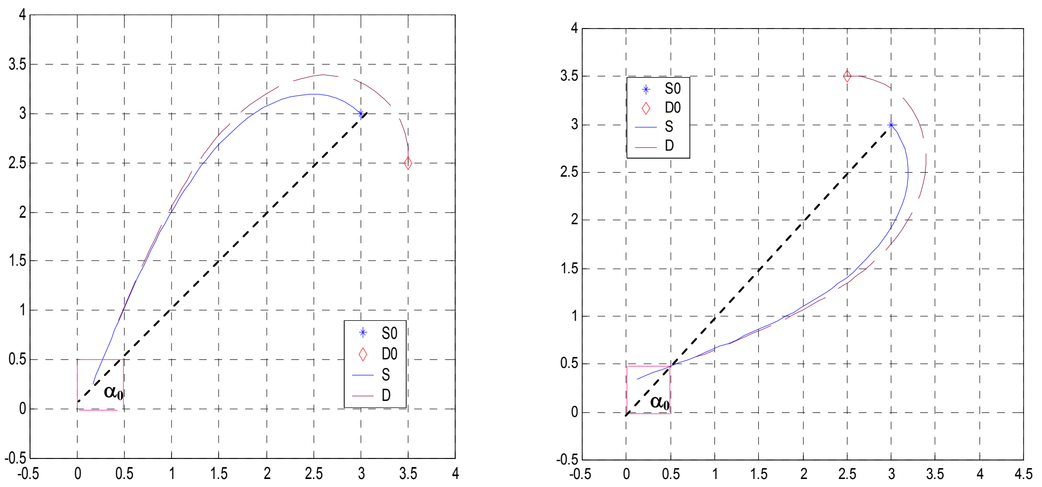
Therefore, θ_* provides a strong minimum for our objective function *J*. Based on this analysis we obtain the nonlinear feedback control law for the pursuit-evasion problem as:

$$\dot{\theta}=-\frac{\sqrt{2}}{r_0}\sin\left(\frac{\theta-\theta_f}{2}\right) \qquad (2.25)$$

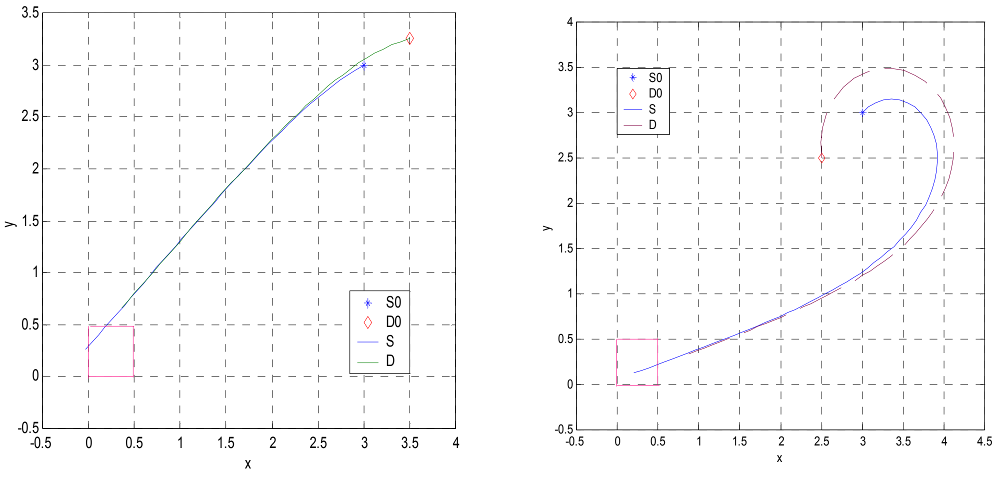
2.5. Simulation Results

Solving for θ_* involves several difficulties such as the non-linear nature of the differential equation given in (2.25), the unknown final angles, θ_f, and the final time t_f. To overcome such problems, the optimal value for the minimizing θ_* is obtained by numerically solving equation (2.25) with iterative guessed values of θ_f, and t_f. Due the discretization process of both time and θ_f, we put a threshold around the origin , such that, once the evader is within this threshold, the simulation ends.

Figures 2.2.a, 2.2.b, 2.2.c, and 2.2.d illustrate the optimal trajectory for the pursuer and the corresponding trajectory of the evader based on different initial conditions. In the figures, S refers to the evader and D refers to the pursuer.



(a)



(b)

Fig.2.2 Optimal trajectory for the pursuer (dashed line) and the evader (solid line) from different initial positions

Inspection of the simulation results shows not only, a clear symmetry of θ_f about the line connecting the initial position of the evader to the origin, but also that there is a relationship between the θ_f , θ_0 and the slope of the symmetry line (angle α_0). This is illustrated in figures 2.2.a and 2.2.b.

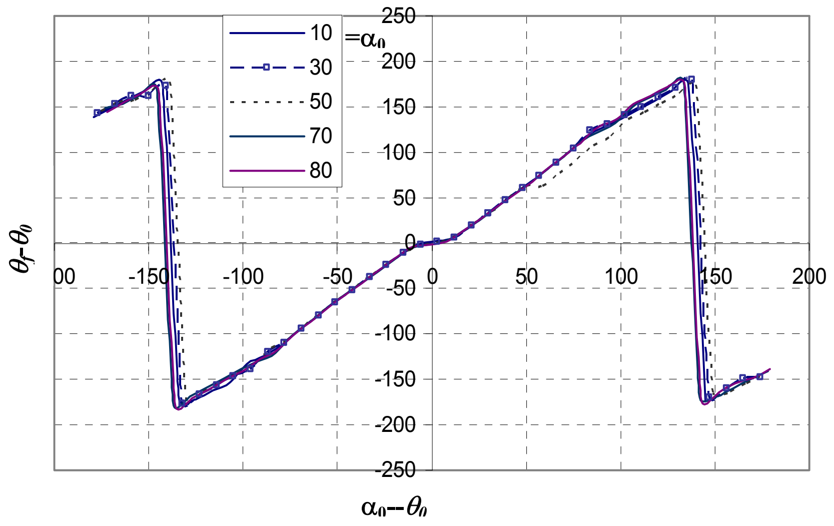


Fig.2.3. Dependence of (θ_f - θ_0) on (α_0-θ_0)

Trying to estimate this dependence of θ_f, on θ_0 and α_0, we plot (θ_f - θ_0) versus (α_0-θ_0) and come up with the results shown in figure 2.3 where the plots are drawn for different values of α_0. Based on the plots, we consider a linear approximation of the dependence of (θ_f - θ_0) on (α_0-θ_0) given as:

$$\theta_f - \theta_0 = k(\alpha_0 - \theta_0) \tag{2.26}$$

where, $\alpha_0 = \arctan \dfrac{y_{e0}}{x_{e0}}$ and a value of approximately 1.35 was obtained for K using the plots. Since the value of θ_f is same for all the intermediate values of θ_0 and α_0 on the system integral curves, we replace (2.26) by

$$\theta_f = k(\alpha - \theta) + \theta \tag{2.27}$$

Therefore, the feedback control law is given in (2.25) with θ_f given by (2.27). Notice from (2.23) that when $\alpha = \theta$ then $\theta_f = \theta$, and from (2.25) we can see that $\dot{\theta} = 0$. This means that when the pursuer and the evader are on the straight line joining the evader with the origin, then the final angle is reached, and after that instant, the pursuer and the evader travel on the same straight line till the evader reaches the origin.

We can show that the optimal feedback control law (2.25) is in fact also a stabilizing control law. In order to show that, let us define a candidate Lyapunov function as

$$v = e^2 \tag{2.28}$$

where $e = \theta - \theta_f$. Differentiating (2.28) with respect to time along the integral curves of (2.25) we get

$$\begin{aligned} \dot{v} &= 2e\dot{e} \\ &= -\frac{4e}{r_0^2}\sin(\frac{e}{2}) \qquad (2.29) \\ &\leq 0 \end{aligned}$$

For $0 \leq e < \pi$ the largest invariant subset of the set $V = \{e : \dot{v} = 0\}$ is given by e = 0. Therefore, from the application of LaSalle's theorem [10] [36], $\underset{t\to\infty}{Lt}\, e(t) = 0$. This implies that the pursuer drives the evader to the origin ($\theta \to \theta_f$).

2.5.1. Impulsive versus smooth solution

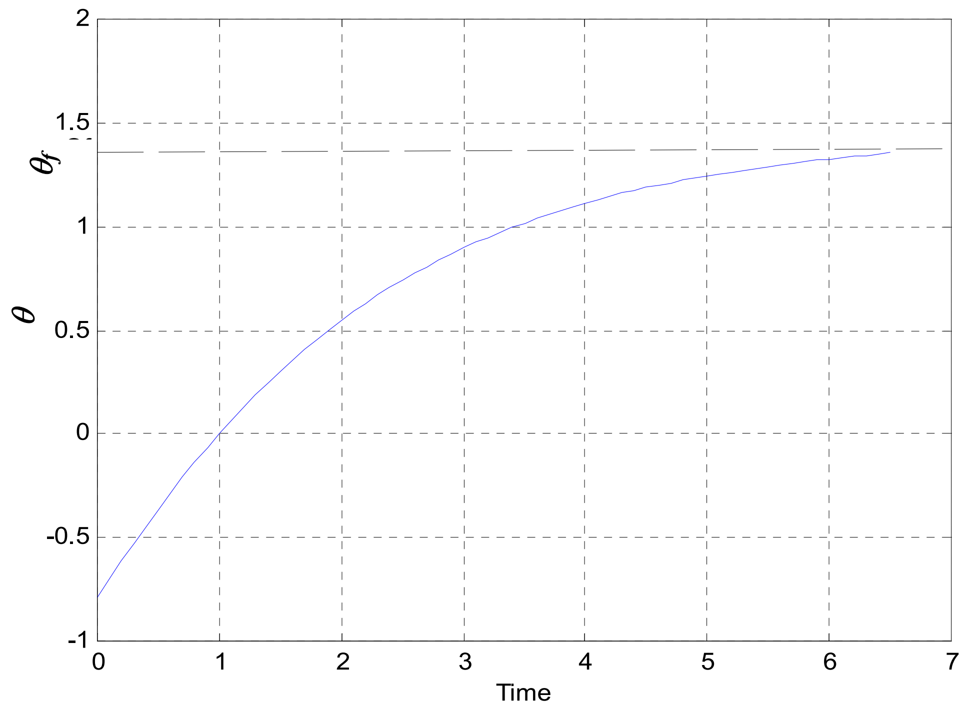


Fig. 2.4 Impulsive versus smooth solutions for θ

One important issue has to be addressed before we conclude this chapter. It can be observed from the optimal trajectory solution and the simulation results that the pursuer tries to align its position on the line connecting the origin to the location of the evader. As shown in figure 2.4, as the pursuer angle θ approaches θ_f, the evader would move in a linear motion till it hits the origin.

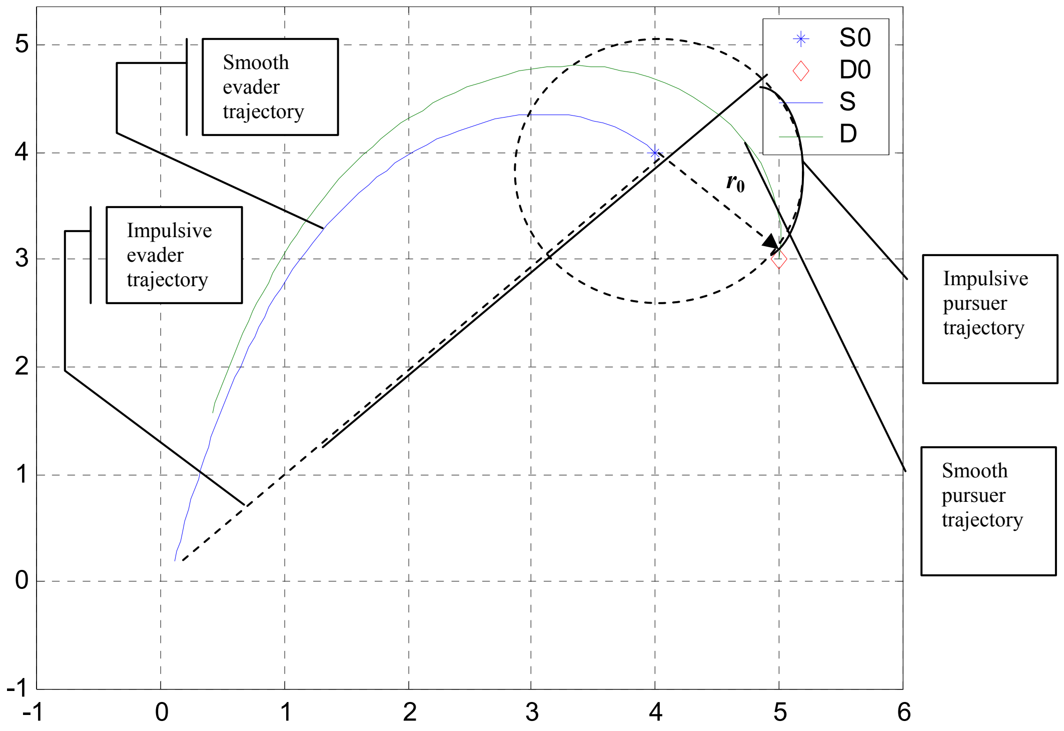


Fig. 2.5 Impulsive versus smooth trajectories

So, if the pursuer is able to get from the θ_0 to θ_f in an impulsive motion, the evader's tangential motion will be negligible due to the restriction of its velocity to be 1 and it will move linearly in the radial direction. This situation is illustrated in figure 2.5 where the smooth path represent the optimal trajectory from Euler Lagrange sense, while the impulsive one represents the trajectory resulting from the instantaneous motion of the pursuer from θ_0 to θ_f. The cost value of the impulsive path will be the sum of the two line segments representing the motion of the radial motion of the evader plus that of the pursuer once it gets behind the evader in addition to the arc length representing the motion of the pursuer from the initial location till it gets behind the evader. Unlike the smooth solution, the value of the cost function for the impulsive motion trajectory is dependent on the radius r_0. Most likely, the cost value for the impulsive motion will be less than that of the smooth motion for small values of r_0, as demonstrated in figure 2.5. However, for large values of r_0 the smooth trajectory is quite likely to have a lower cost value than that of the impulsive motion trajectory. More detailed study of the dependence of the cost function value on the radius r_0 is needed to find the domain of r_0 that can result in lower cost value.

However, the following issues have to be taken in consideration while designing the trajectory;

- The capability to achieve the required infinite rate of change in θ.
- For the final implementation by wheeled mobile robots, we are interested more in a smooth trajectory so the paths are follow-able as it will be discussed in the coming chapter.

2.6. Conclusion

In this chapter, we have provided a quick review of the basic theory of calculus of variations and optimal control that we used to give derivation of the optimal trajectory for one special case of the pursuit evasion game. The optimal control policy was proven to be dependent only on the space variables and therefore was a feedback control law. Finally, a sample result simulation that supports our conclusion was give at the end of the chapter.

Chapter 3

Feedback Control for the Optimal Pursuit-Evasion Trajectory

3.1. Introduction

In the previous chapter, a feedback control law was obtained so that the pursuer, when applying it, will drive the evader to the pen using the shortest path. In obtaining this trajectory, both pursuer and evader were assumed to be points with omni-directional motion capabilities. In real world, robots are usually wheeled mobile robots (WMR). Practical wheeled mobile robots have some additional constraints on their dynamics. The details of different types of WMRs considered in research so far can be found in [37]. Some of these constraints may be on position, while the other may be on velocity. Therefore, these additional constraints have to be taken car of when the agents are to be represented by WMR.

In this chapter, we will begin by introducing the basic concepts, definitions and principles of non-holonomically constrained systems. Then, we'll add these additional constrains to the dynamical constraints of the system introduced in chapter 1 such that the WMRs representing the pursuer and the evader move from any initial position to in the x-y grid to the pen located in the neighborhood of the origin. Finally, a sample of the simulation results will be given.

3.2. Basic Principles of Non-holonomic Motion Planning

In addition to the differential equations describing the dynamics of any system, the motion of this system can be subjected to additional set of kinematics constraints in the form;

$$a_i^T(q,\dot{q}) = 0 \qquad i = 1,2....k \qquad 3.1$$

where q is generalized n-dimensional coordinate vector $(q_1, q_1, \ldots.., q_n)^T$.

If the set of constraints given in 3.1 can be written in the linear form

$$a_i^T(q)\dot{q} = 0 \qquad i = 1,2....k \tag{3.2}$$

They are called ***Pfiffian*** constraints [38].

This set of Pfiffian constrains is said to be ***holonomic*** if it is integrable (in this case, the constraints represent a geometric limitation). Meanwhile, if these constraints are not integrable, they are called ***non-holonomic*** constrains (in this case, they represent a set of kinematic limitation).

Therefore, the addition of holonomic/non-holonomic constraints to any dynamical system gives rise to an important question; given any 2 points q_i, $q_j \in$ the configuration space $\mathcal{Q}$; when does a trajectory $q(t)$ connecting the 2 points exist such that is satisfies the kinematic constraints? This question represents another form of studying the controllability of dynamical systems with holonomic/non-holonomic constraints.

3.2.1 Controllability of Non-holonomic Systems

Consider a non-linear control system of the form

$$\overset{*}{q} = f(q) + \sum_{j=1}^{n} g_j(q)u_j \tag{3.3}$$

with the states $q \in \mathcal{Q} \approx \mathcal{R}^n$ and the control inputs $u \in \mathcal{U} \subseteq \mathcal{R}^m$

- The system given in 3.3 is said to be a *drift-less* system if $f(q)=0$, which means any configuration $q = (q_1, q_1, \ldots., q_n)^T$ is at equilibrium with zero input controls u_j.
- The system described by equation 3.3 is *controllable* if $\forall q_1$, $q_2 \in \mathcal{Q}$, $\exists T<\infty$, and $u:[0,T]$ $u \in \mathcal{U}$ such that $q(0,T,q_1,u)=q_2$.
- The control system given by equation 3.3 is said to be ***Locally Accessible*** (LA) form point q_0 if for any neighborhood ***V*** of q0 and T>0, there exist a non-empty set Ω such that;

  $\Omega \subset R_T^V(x0)$

Where $R_T^V(x0)$ is the set of all *reachable* states from q0 within time interval [0,T], given by;

$R_T^V(x0) = \{q \in Q | q(0,T,q_0,u) = q, \forall t \in [0,T], q(0,T,q_0,u) \in V$

- System 3.3 is ***Globally Accessible*** if it is locally accessible $\forall\ q \in Q$.

For drift-less control systems, controllability and local time accessibility are equivalent. A useful theorem that allows testing the controllability of drift-less nonlinear systems is *Chow Theorem*, which gives a similar tool to the controllability rank condition of linear time invariant systems. This test is based on *Lie Algebra rank condition* [39].

Chow's Theorem: *A non-linear, drift-less control system, is locally accessible "controllable" if and only if the rank of the accessibility matrix equals n i.e.*;

$Rank([g_1; g_2; [g_1, g_2]; [g_1, [g_1, g_2]] ....]) = n$

Where $[g_1, g_2](q)$ the Lie Bracket of the two vector fields g_1, and g_2 is given by;

$$[g_1, g_2](q) = g_1 \frac{\partial g_2}{\partial q} - g_2 \frac{\partial g_1}{\partial q} \qquad 3.4$$

3.3. Feedback Control Design for the Optimal Pursuit-Evasion Trajectory

With these illustrated basic tools and concepts in mind, our objective now is to design a feedback control law for our pursuer, as a function of the evaders states such that both the evader and the pursuer will follow the optimal trajectory obtained in chapter 2.

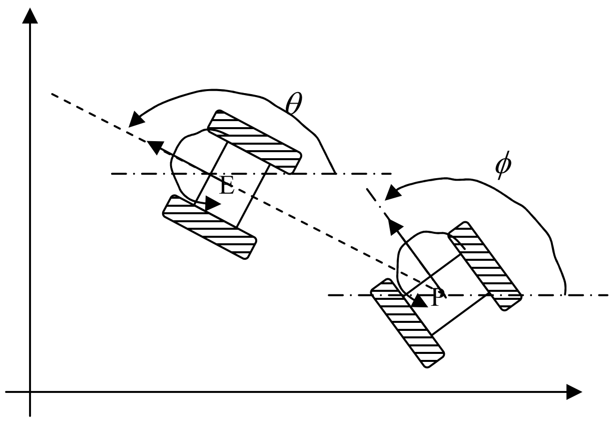


Fig. 3.1. WMR representation of pursuer and evader

Assuming that both the pursuer and the evader are represented by a two wheeled mobile robot as shown in figure 3.1. In addition to the system dynamics given in equations 1.5-1.8, both the evader and pursuer have to satisfy the non-holonomic constraints imposed on both robots.

Non-holonomic constraint on the evader is given by;

$$\dot{x}_e \sin \varphi_e - \dot{y}_e \cos \varphi_e = 0 \qquad 3.5$$

Similarly, the pursuer non-holonomic constraint is given by;

$$\dot{x}_p \sin \phi_p - \dot{y}_p \cos \phi_p = 0 \qquad 3.6$$

Alternatively, the above constraints given by 3.5 and 3.6 can be represented in terms of the two control variables, named the driving velocity *v* and the steering velocity ω as follows;

$$\begin{aligned} \dot{x}_e &= v_e \cos \varphi_e \\ \dot{y}_e &= v_e \sin \varphi_e \\ \dot{\varphi}_e &= \omega_e \end{aligned} \qquad 3.7$$

$$\begin{aligned} \dot{x}_p &= v_p \cos \varphi_p \\ \dot{y}_e &= v_e \sin \varphi_p \\ \dot{\varphi}_p &= \omega_p \end{aligned} \qquad 3.8$$

Equation 3.7 and 3.8 can be put in a form of a drift-less system as shown below;

$$\dot{q} = \sum_{i=1}^{m} u_i g_i(q) \qquad ,u \in U \quad and \quad q \in Q \qquad 3.9$$

With

$$g_1 = \begin{pmatrix} \cos\varphi \\ \sin\varphi \\ 0 \end{pmatrix} \quad ,g_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \qquad 3.8$$

and;

$$[g_1, g_2] = \begin{pmatrix} \sin\varphi \\ -\cos\varphi \\ 0 \end{pmatrix} \qquad 3.9$$

Therefore, the accessibility matrix is given by;

$$C = \begin{pmatrix} \cos\varphi & 0 & \sin\varphi \\ \sin\varphi & 0 & -\cos\varphi \\ 0 & 1 & 0 \end{pmatrix} \qquad 3.10$$

With *rank*(C) =3, this means each of the robots is completely controllable. Therefore any configuration in the working space is accessible.

Now, we need to come up with the driving velocity and the driving speed of each robot to satisfy both the non-holonomic constraints and the optimality constraints imposed on the angle θ_*

Using equations 2.11, and 2.25 with 3.5, we get

$$\begin{aligned} &\sin\varphi_e \cos\theta_{*e} - \cos\varphi_e \sin\theta_* = 0 \\ &\sin(\varphi_e - \theta_*) = 0 \Rightarrow \\ &\varphi_e = \theta_* \end{aligned} \qquad 3.11$$

Therefore, the optimal steering speed of the evader robot is;

$$\begin{aligned} \omega_e &= \dot{\theta}_* \\ &= \frac{\sqrt{2}}{r_0}\sin\left(\frac{\theta - \theta_f}{2}\right) \end{aligned} \qquad 3.12$$

The driving speed of the evader robot is given by;

$$\begin{aligned} v_e^2 &= \dot{x}_e^2 + \dot{y}_e^2 \\ &= 1 \end{aligned} \qquad \ldots.. \qquad 3.13$$

Similarly, using 2.12, and 3.12 with 3.6 we get;

$$\sin\varphi_p(\cos\theta_* - r_0\dot{\theta}_*\sin\theta_*) - \cos\varphi_p(\sin\theta_* - r_0\dot{\theta}_*\cos\theta_*) = 0$$
$$\sin\varphi_p\cos\theta_* - \cos\varphi_p\sin\theta_* - r_0\dot{\theta}_*(\sin\varphi_p\sin\theta_* - \cos\varphi_p\cos\theta_*) = 0 \qquad 3.14$$

With some mathematical manipulations, we get;

$$\sin(\varphi_p - \theta_*) + r_0\dot{\theta}_*\cos(\theta_* - \varphi_p) = 0$$
$$\tan(\varphi_p - \theta_*) = r_0\dot{\theta}_* \qquad 3.15$$
$$= r_0\omega_e$$

Differentiating equation 3.15, the pursuer optimal steering speed is given by;

$$\omega_p = \dot{\theta}_* + \frac{r_0^2\ddot{\theta}_*}{1 + r_0^2\dot{\theta}_*^2}$$
$$= \omega_e + \frac{r_0^2\dot{\omega}_e}{1 + r_0^2\omega_e^2} \qquad 3.16$$

Using form 2.22 in 3.16 we get;

$$\omega_p = \omega_e + \frac{\sin(\theta - \theta_f)}{2(1 + r_0^2\omega_e^2)} \qquad 3.19$$

Similarly, the driving speed of the pursuer robot can be obtained as shown below

$$v_p^2 = \dot{x}_p^2 + \dot{y}_p^2 \qquad 3.20$$

From 2.11, and 2.12 in 3.20 we get;

$$v_p^2 = (\dot{x}_e - r_0\dot{\theta}_*\sin\theta_*)^2 + (\dot{y}_e + r_0\dot{\theta}_*\cos\theta_*)^2$$
$$= \dot{x}_e^2 + \dot{y}_e^2 + (r_0\dot{\theta}_*)^2 \qquad 3.21$$
$$= 1 + r_0^2\dot{\theta}_*^2$$

Thus,

$$\begin{aligned} v_p &= \pm\sqrt{1 + r_0^2 \dot{\theta}^2} \\ &= \pm\sqrt{1 + r_0^2 \dot{\varphi}_e^2} \\ &= \pm\sqrt{1 + r_0^2 \omega_e^2} \end{aligned} \qquad 3.22$$

Therefore, equations 3.19 with equations 3.22 give the applicable feedback control law for the pursuer WMR.

One important point to notice is as that; for the point pursuer-evader representation given in chapter 2, the initial conditions of the pursuer could be any point on the circumference of a circle of radius r_0. From this initial position, the pursuer was able, optimally, to drive the evader to the pen. Unlike this, addition of the non-holonomic constraints on the pursuer and the evader restricts the initial position of the pursuer to a single location on the evader's circle of detection. This initial position of the pursuer is given by the initial orientation of the WMR given by

$$\varphi_p(0) = \tan^{-1}\left(\frac{\dot{y}_p(0)}{\dot{x}_p(0)}\right) \qquad 3.23$$

Using 2.11 and 2.12 here gives us

$$\varphi_p(0) = \tan^{-1}\left(\frac{\sin\theta_*(0) + r_0\dot{\theta}_*(0)\cos\theta_*(0)}{\cos\theta_*(0) - r_0\dot{\theta}_*(0)\sin\theta_*(0)}\right) \qquad 3.24$$

3.4. Simulation Results

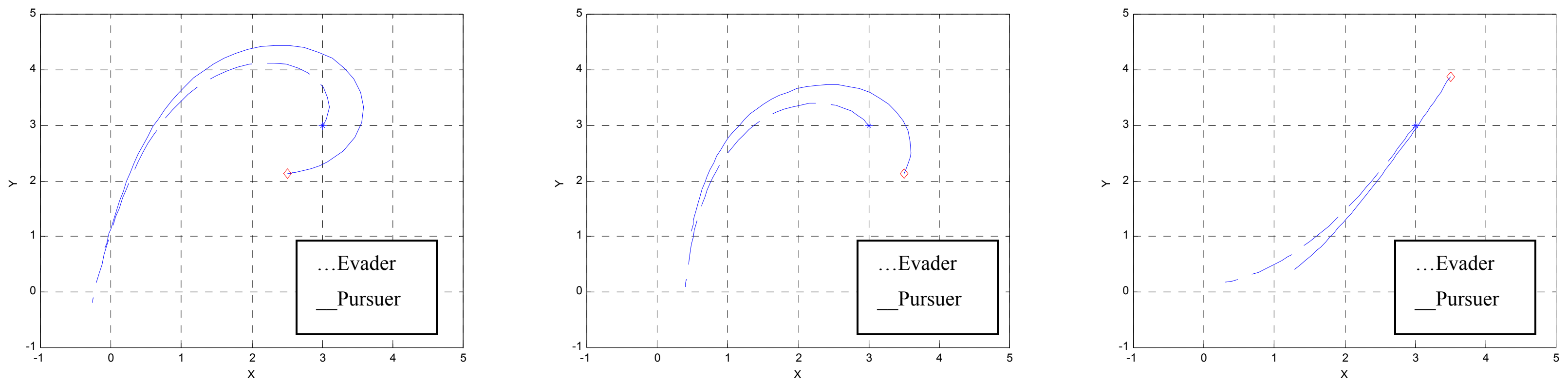
Figure 3.2 (a) gives a sample of the simulation results with different initial conditions for the pursuer. The simulation begins with the pursuer detecting the evader's initial position and orientation. Then, based on the initial orientation of the evader, the pursuer evaluates the optimal initial angle of approach. Once the pursuer is on the circumference of the circle of detection of the evader, the evader begins its avoidance motion following the dynamics of equation 3.7.

Similar to the point representation of the pursuer and evader, the optimal control law is dependent on both the final time and the final orientation angle which are unknowns. So, we incremented the time in small steps and searched the final orientation angle that satisfies the optimal control law. The search time can be highly reduced in a similar way to that given in chapter 2 by estimating a similar dependence of the final orientation angle on the initial conditions.

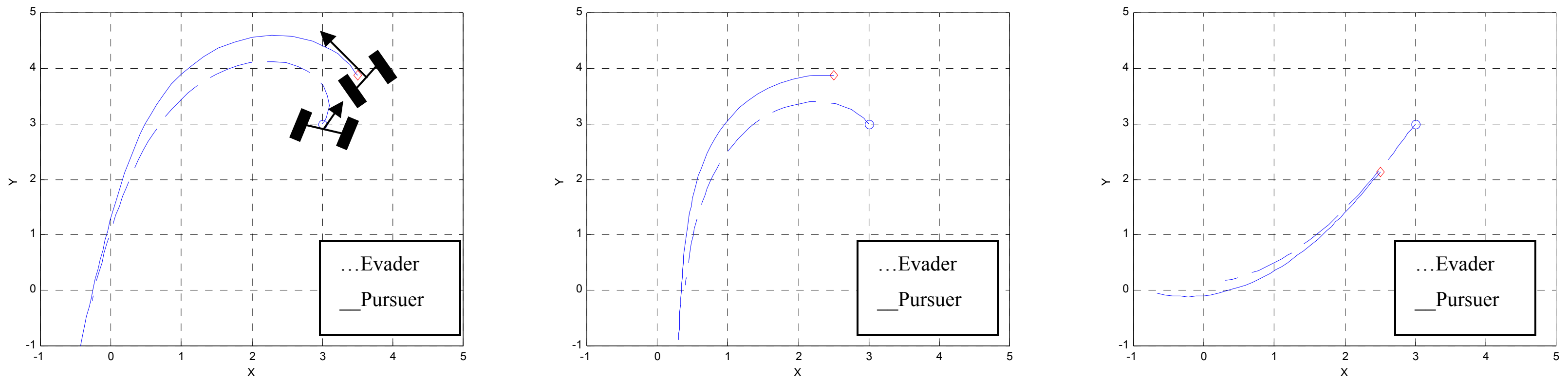
Another interesting issue came up if the velocity of both the pursuer and the evader robots was restricted to forward motion only rather than being restricted to forward or backward motion. Addition of such constraint doesn't affect the objective function and agents' trajectories still satisfy the non-holonomic constraints. Despite the failure of the controllability of the modified system from Lie bracket accessibility sense (because the motion is restricted to one direction only rather than two) the system is still controllable because any configuration is reachable. In addition, it follows the optimal constraints derived for θ_*. Figure 3.2 (b) illustrates a sample of the simulation results for this modified system.

3.5. Conclusion

In this chapter, we considered a realization of the pursuit evasion games that is different from the standard form covered by researchers in previous work. We have provided a detailed derivation of an optimal trajectory for the pursuit evasion game realization using WMRs. The optimal control law was proven to be dependent only on the space variables and therefore was a feedback control law. This design realization was supported by the simulation results illustrated in section 3.4.



(a) Non-restricted velocity direction



(b) Forward velocity direction only

Fig 3.2 Simulation results

Chapter 4

The Discrete Deterministic Model

4.1. Introduction

In this chapter we present the dynamics and control design of discrete-time, discrete-space representation of the herding problem. The motivation behind studying such model is based on the following;

- The design of cooperative multi-agent systems is dependent on understanding the dynamic behavior of smaller systems.
- The design of high-level systems beginning from low level ones allows the designer to devise a top down methods, by which low-level systems may be obtained from the high level ones through specifying some constraints on the high-level systems.
- Studying and developing this model, introduces an evaluation method of the effectiveness of any machine learning technique dealing with similar problems.

The work presented in this chapter represents a primary step in building reinforcement learning model to deal with such situations, as it will be explained. Based on this objective, and relaying on the definition of reinforcement learning as an approach to machine intelligence that combines dynamic programming and supervised learning disciplines to successfully solve problems that neither discipline can address individually [40-41], dynamic programming appears to be the candidate approach to come up with the most advantageous control technique. Moreover, dynamic programming has proven to be an efficient technique due to its simplicity and applicability to wide range of problems. For these reasons, we'll begin this chapter by giving a brief introduction on dynamic programming principles.

Then, the problem model is introduced for a simple, deterministic, and passive-evader. After that, the proposed solution algorithms are illustrated with complete analysis and proofs of the validation of each proposed algorithm. Finally, the simulation results of each of the proposed solution technique is given at the end of the chapter

4.2. Dynamic Programming Principles

Dynamic Programming is a particular approach to optimization. By optimization we mean finding the best solution to some problem from a set of alternatives. So, a definition of the basic components of a mathematical optimization model should be given at the beginning. These components are:

1. *Variables:* include decision variables, state variables, or independent variables. These variables represent the factors to manipulate to achieve the desired objective.
2. *Objective function*: which represents the measure of effectiveness or the value of utility associated with some particular combination of variables.
3. *Constraints* (feasibility conditions): usually represented by a set of algebraic or differential equations or even inequalities that the variables have to satisfy.

The principal idea of dynamic programming is based on the *Principle of Optimality* introduced by Bellman [42] which states:-
"*An optimal policy has the property that, whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with respect to the state which results from the initial decisions*". Simply, an optimal policy should consist of optimal sub-policies.

4.2.1. Dynamic Programming Basic Theory and Functional Equations

Problems to which dynamic programming can be applied, are usually called, *Sequential Decision Processes* [43]. The sequential decision processes represent a class of systems that progress through a sequence of consecutive stages. At each stage, the system can be described by a relatively small set of parameters called, *state variables* or *state vector*. At each stage, one or more *decisions* must be made. These decisions may depend on either the stage and/or the state of the system.

A basic characteristic of the systems that dynamic programming deals with is that the past states of the system have no effect on the current state. This means that the decisions are made based only on the current state of the system. When a decision is made, the system undergoes a *transition* from one state to another and a *cost* is associated with each transition. This cost is determined by a single-valued function of the input state variables. Meanwhile, the inter-state transition results from a single-valued function of the decision variables and the current state.

Thus, and based on the above introduction, dynamic programming involves some conceptual terms; *state*, *stage*, *transition*, *decision* (policy), and *cost* that have to be defined clearly before going in more details.

- *State*: The state space is a non-empty set $\mathbf{X}$ whose elements $x \in \mathbf{X}$ are state variables that describe the condition of the system or process under study. Obviously, the state space consists of all the possible states the system can be in.
- *Stage*: This concept is introduced such that the decisions can be ordered. Therefore, the stage variable is usually discrete. Also, it should be noted that dynamic programming is considered to be discrete or continuous based on the nature of the stage variables not the state variables. Usually, in discrete dynamic programming, states and stages are the same.
- *Decision*: Based on each state variable $x \in \boldsymbol{X}$, there is a corresponding non-empty set $\boldsymbol{U_x}$ called the decision set for x, such that every element (decision variable) $u(x) \in \boldsymbol{U_x}$ represents one of the choices the controller can make when the system is in state $x \in \boldsymbol{X}$.
- *Transitions*: Assuming that the process under study is in state $x_1 \in \boldsymbol{X}$, and a decision $u(x_1) \in \boldsymbol{U_{x1}}$ is made, then the process will be transformed to another state $x_2 \in \mathbf{T}(x_1, u(x_1))$ where, $\mathbf{T}(x, u(x))$ is the set of all possible states the process can move from state $x \in \boldsymbol{X}$ using decision $u(x) \in \boldsymbol{U_x}$. This transition operator $\mathbf{T}(x, u(x))$ is of two different types; *deterministic*, and *stochastic*.
- *Policy*: It is defined as the set of all decision sets $\boldsymbol{U_x}$ corresponding to all state variables $x \in \boldsymbol{X}$.

- *Cost*: The cost is a single-valued function, $c_u(x)$, defined on the current state variable and the corresponding decision variable as the return that would be obtained if the process is at state $x \in \boldsymbol{X}$ and a decision $u(x) \in \boldsymbol{U_x}$ is made.

Therefore, the basic problem can be mathematically formulated as:

$$\max/\min V = f(x_1, x_2, x_3, ..., x_n) \qquad 4.1$$

subjected to $\quad h_i(x_1, x_2, x_3, ..., x_n) = 0 \qquad i = 1,2,....m$

Applicability of dynamic programming to certain class of problems requires two main requirements in order that the principle of optimality to be invoked. These 2 conditions are:

1. *Separability of the Objective Function*: which means that, $\forall k$, the effect of the final k stages of an n-stage process on the objective function only depends on state x_{n-k} and the final k decisions.
2. *State Separation Property*: by this we mean that, $\forall k$, transition from state x_k to state x_{k+1} only depends on state x_k and decision $u_{k+1}(x_k)$ but not on any previous states. This property is also known as Markovian state property and systems having these type of properties are called, memory-less systems.

These two requirements basically represent the necessary and sufficient conditions for the principle of optimality to be applied for the model to which dynamic programming technique is to be applied [44-45].

4.2.2. Deterministic Dynamic Programming Algorithm

Before introducing the dynamic programming algorithm, we'll give some more detailed mathematical interpretation of the main components of the basic problem formulation [46];

- A *discrete system* whose state transition is governed by;

  $$x_{k+1} = f(x_k, u_k)$$

- A *control constraints* $u_k(x_k) \in U_x$
- An *additive cost function* of the form;

  $$V = c_N(x_N) + \sum_{k=1}^{N-1} c_k(x_k, u_k)$$

  where c_k is some function that gives the interstate transition cost, and c_N is the cost associated with the final state.
- Optimization over polices which means the rules applied, $u_k(x_k)$, for each possible state at time instant k to optimize the cost function $\boldsymbol{V}$.

Thus, the dynamic programming algorithm can be stated as following [46];

Denoting $\boldsymbol{V}^*(x_0)$ *to be the optimal value of the cost function* $\boldsymbol{V}(x)$, *then*;

$\boldsymbol{V}^*(x_0) = \boldsymbol{V}_0(x_0)$

where, the value function $\boldsymbol{V}_0(x_0)$ *is given by the last step of the following algorithm, which proceeds backward in time from time instant* N-1 *to instant* 0;

$\boldsymbol{V}_N(x_N) = c_N(\mathrm{x}_N)$

$$V_k(x_k) = \min_{u_k \in U}\{c_k(x_k, u_k) + V_{k+1}[f(x_k, u_k)]\}, \qquad k = 0,1,2,\ldots,N-1$$

Moreover,

$$u^*{}_k(x_k) = \arg\{\min_{u_k \in U} V^*{}_k(x_k)\}$$

is the optimal control policy.

After this brief introduction about dynamic programming foundation, we are ready to present a dynamic programming based solution to the pursuer evader herding problem, where the pursuer and the evader are playing a non-cooperative deterministic game. We present the dynamics of the problem and then provide the dynamic programming solution to the problem. The solution is proven to be correct and then simulations are performed to illustrate some example runs.

4.3. A NxN Grid Pursuer-Evader Problem

We consider the pursuit-evasion herding problem in a $N \times N$ grid as shown in Figure 4.1. The pursuer can occupy one of the $N \times N$ positions and so can the evader. Therefore, there are N^2 states in the system. The aim of the pursuer is to make the evader go to the pen, which is the (0,0) state for the evader. The following shows the nomenclature used in this paper.

$x_p(k)$	x coordinate of the pursuer position at time instance k.
$y_p(k)$	y coordinate of the pursuer position at time instance k.
$x_e(k)$	x coordinate of the evader position at time instance k.
$y_e(k)$	y coordinate of the evader position at time instance k.
$\mathbf{x}(k)$	state vector given by $\mathbf{x}(k) = [x_e(k)\ y_e(k)\ x_p(k)\ y_p(k)]$ at time instance k.

For the NxN pursuer evader problem, we have $\forall k \quad x_p(k) \in \{0,1,2,...,N\}$, $y_p(k) \in \{0,1,2,....,N\}$, $x_e(k) \in \{0,1,2,...,N\}$ and $y_e(k) \in \{0,1,2,....,N\}$. However, the pursuer and the evader can not have the same location on the grid as their initial positions. It can be proven that if they have different initial positions, then based on the allowable actions of both (as described later), they can never end up on the same location. There is a cost of one unit for each step (horizontal or vertical or diagonal) of a pursuer as well as of a evader. The aim of the pursuer is to move the evader to the pen i.e. to the (0,0) coordinate, with the least cost. Figure 4.1 below shows the 3x3 grid for the pursuer-evader problem.

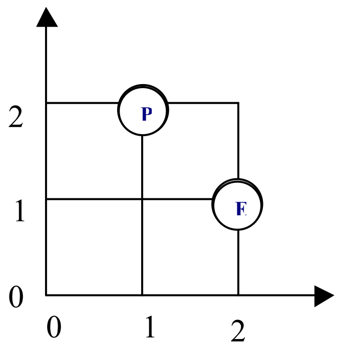


Fig. 4.1 A 3x3 example of Pursuer-evader Problem Grid

<u>Definition1</u>: *Equilibrium state of the evader*: The evader is in an equilibrium state when given a time instant T the following condition is satisfied:

$\forall k \geq T$

if

$$x_p(k) = x_p(T) \text{ and}$$

$$y_p(k) = y_p(T)$$

then

$$x_e(k) = x_e(T) \text{ and}$$

$$y_e(k) = y_e(T)$$

<u>Definition 2</u>: *Final equilibrium state of the evader*: The evader is in the final equilibrium state when given a time instant T the following condition is satisfied:

$\forall k \geq T$

if

$$x_p(k) = x_p(T) \text{ and}$$

$$y_p(k) = y_p(T)$$

then

$$x_e(k) = 0 \text{ and}$$

$$y_e(k) = 0$$

Definition 3: *Positive successor function*: Positive successor function is a function given by

$PS(.) : X = \{0,1,2,...,N\} \rightarrow Y = \{1,2,...,N\}$ i.e.,

$\forall y \in Y, x \in X,$

$$
\begin{aligned}
y = PS(x) &= x + 1 \qquad && \text{if } x \neq N \\
&= N && \text{if } x = N
\end{aligned}
$$

Definition 4: *Negative successor function*: Negative successor function is a function given by

$NS(.) : X = \{0,1,2,.....,N\} \rightarrow Z = \{0,1,2,.........,N-1\}$ i.e.,

$\forall z \in Z, x \in X,$

$$
\begin{aligned}
z = NS(x) &= x - 1 \qquad && \text{if } x \neq N \\
&= N && \text{if } x = N
\end{aligned}
$$

The following rules generate the dynamics of the evader and pursuer movements:

1. $\forall k \quad x_p(k) \in \{0,1,2,....N\}$, $y_p(k) \in \{0,1,2,...,N\}$, $x_e(k) \in \{0,1,2,...N\}$ and $y_e(k) \in \{0,1,2,...N\}$
2. The pursuer can only move when evader is in an equilibrium state
3. The pursuer can only move one step in one time instant. That step can be in horizontal, vertical or diagonal direction. The evader can also only move one step in horizontal, vertical or diagonal direction.
4. The evader moves based on the following rules:
   a) Far condition:

      If

      $x_e(k) < NS(x_p(k))$ or $x_e(k) > PS(x_p(k))$ or $y_e(k) < NS(y_p(k))$ or $y_e(k) < NS(y_p(k))$

      then

      $x_e(k+1) = x_e(k)$ and $y_e(k+1) = y_e(k)$

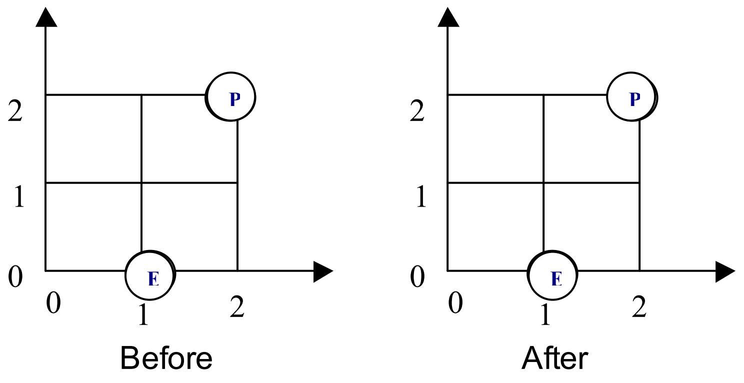


Fig 4.2 Example of a far condition

b) Left top corner pursuer right condition:

If

$x_e(k) = 0$ and $x_p(k) = PS(x_e(k))$ and $y_e(k) = y_p(k) = N$

then

$x_e(k+1) = x_e(k)$ and $y_e(k+1) = NS(y_e(k))$

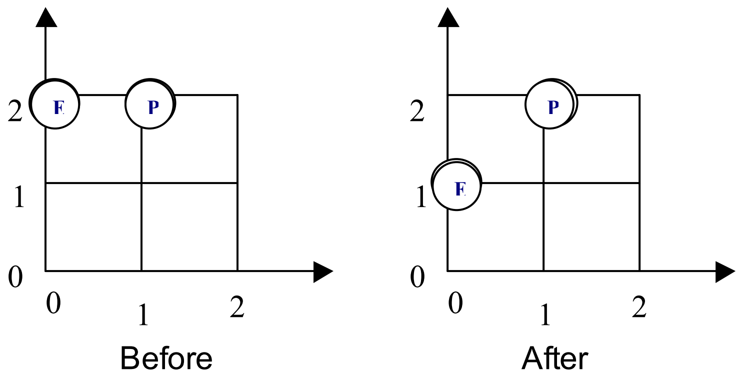


Fig 4.3 Example of the left top corner pursuer right condition

c) Left top corner pursuer down condition:

If

$x_e(k) = x_p(k) = 0$ and $y_e(k) = 2$ and $y_p(k) = NS(y_e(k))$

then

$x_e(k+1) = PS(x_e(k))$ and $y_e(k+1) = y_e(k)$

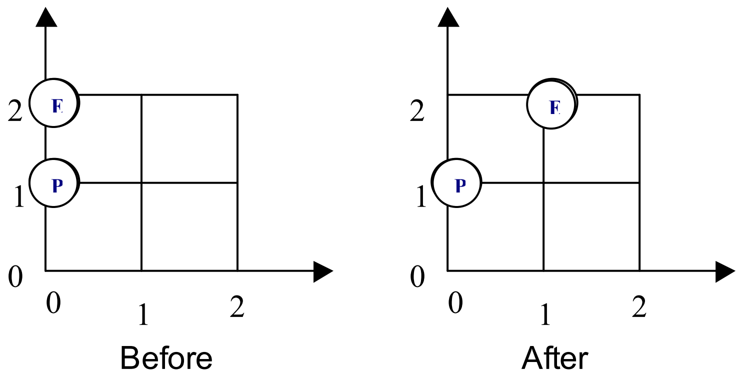


Fig. 4.4. Example of the left top corner pursuer down condition

d) Right top corner pursuer left condition:

If

$x_e(k) = 2$ and $x_p(k) = NS(x_e(k))$ and $y_e(k) = y_p(k) = 2$

then

$x_e(k+1) = x_e(k)$ and $y_e(k+1) = NS(y_e(k))$

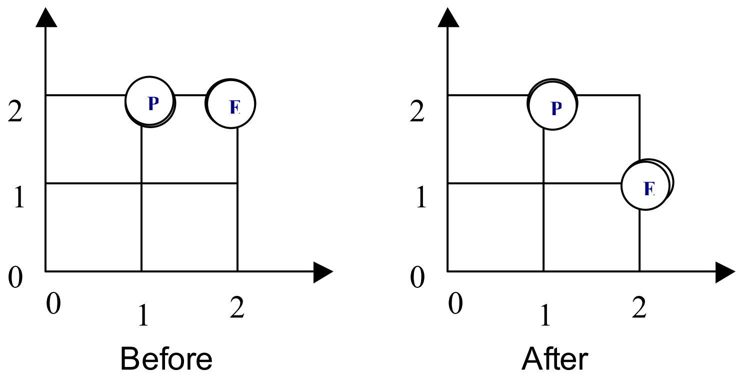


Fig. 4.5 Example of the right top corner pursuer left condition

e) Right top corner pursuer down condition:

If

$x_e(k) = x_p(k) = 2$ and $y_e(k) = 2$ and $y_p(k) = NS(y_e(k))$

then

$x_e(k+1) = NS(x_e(k))$ and $y_e(k+1) = y_e(k)$

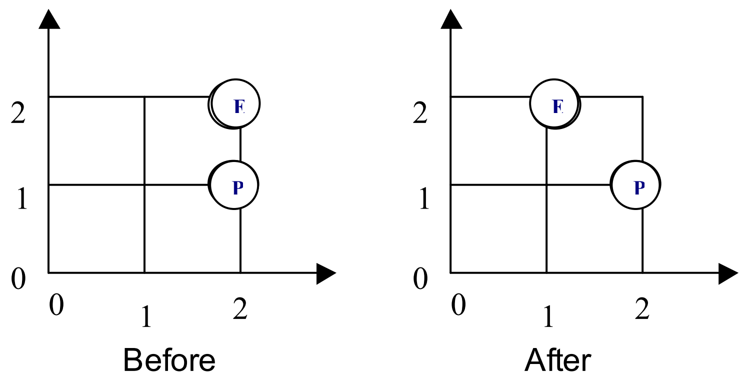


Fig. 4.6 Example of the right top corner pursuer down condition

f) Left bottom corner pursuer right condition:

If

$x_e(k) = 0$ and $x_p(k) = PS(x_e(k))$ and $y_e(k) = y_p(k) = 0$

then

$x_e(k+1) = x_e(k)$ and $y_e(k+1) = PS(y_e(k))$

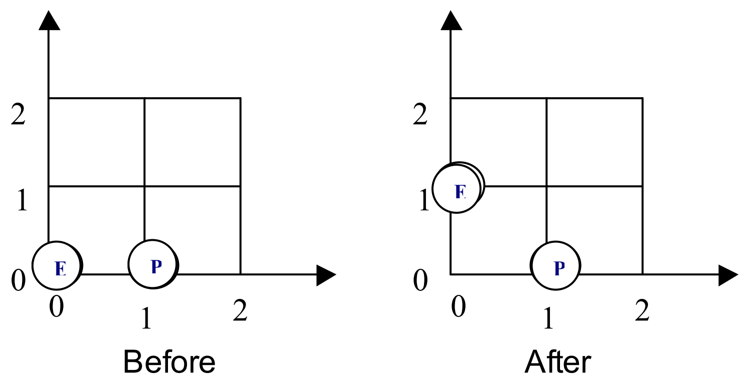


Fig. 4.7 Example of the left bottom corner pursuer right condition

g) Left bottom corner pursuer up condition:

If

$x_e(k) = x_p(k) = 0$ and $y_e(k) = 0$ and $y_p(k) = PS(y_e(k))$

then

$x_e(k+1) = PS(x_e(k))$ and $y_e(k+1) = y_e(k)$

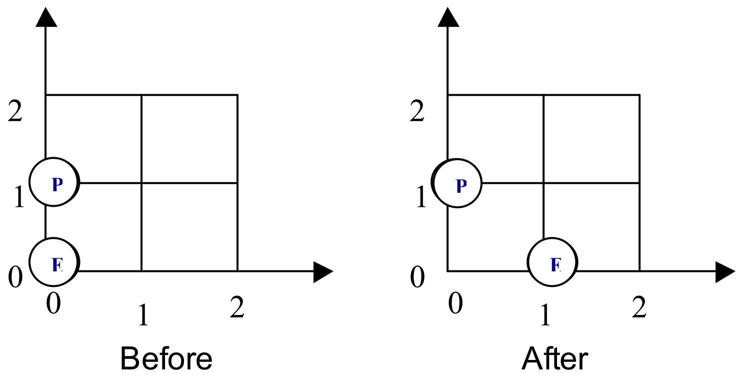


Fig.4.8 Example of the left bottom corner pursuer up condition

h) Right bottom corner pursuer left condition:

If

$x_s(k) = 2$ and $x_d(k) = NS(x_s(k))$ and $y_s(k) = y_d(k) = 0$

then

$x_s(k+1) = x_s(k)$ and $y_s(k+1) = PS(y_s(k))$

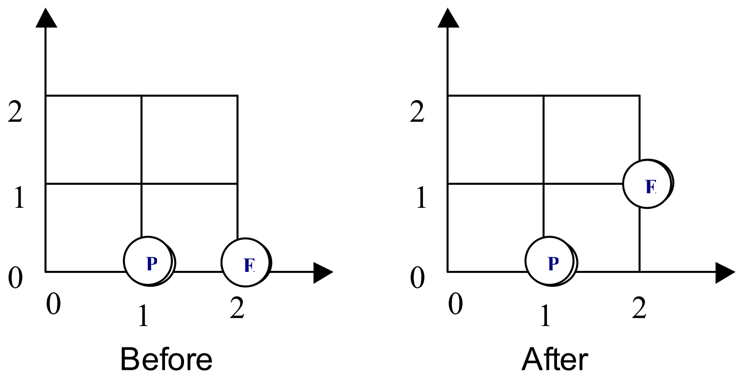


Fig. 4.9 Example of the right bottom corner pursuer left condition

i) Right bottom corner pursuer up condition:

If

$x_e(k) = x_p(k) = 2$ and $y_e(k) = 0$ and $y_p(k) = PS(y_e(k))$

then

$x_e(k+1) = NS(x_e(k))$ and $y_e(k+1) = y_e(k)$

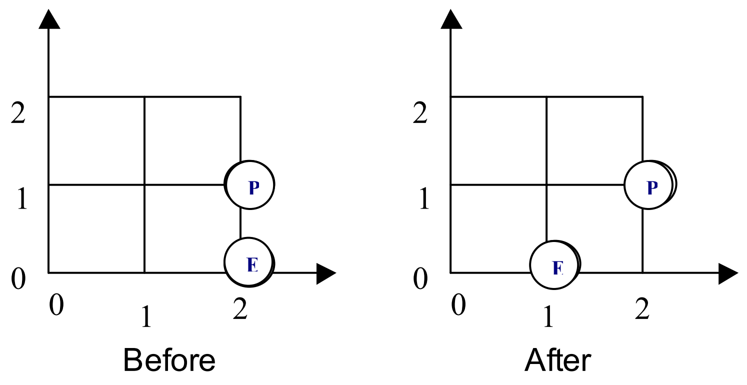


Fig. 4.10 Example of the right bottom corner pursuer up condition

j) <u>Other conditions</u>:

If (a) to (i) are not satisfied and

$$x_e(k) = NS(x_p(k))$$

then

$$x_e(k+1) = NS(x_e(k))$$

If (a) to (i) are not satisfied and

$$x_e(k) = PS(x_p(k))$$

then

$$x_e(k+1) = PS(x_e(k))$$

If (a) to (i) are not satisfied and

$$y_e(k) = NS(y_p(k))$$

then

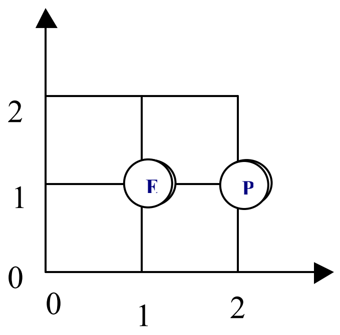
$$y_e(k+1) = NS(y_e(k))$$

If (a) to (i) are not satisfied and

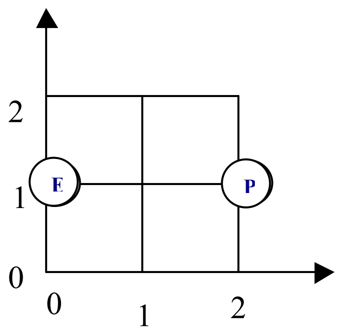
$$y_e(k) = PS(y_p(k))$$

then

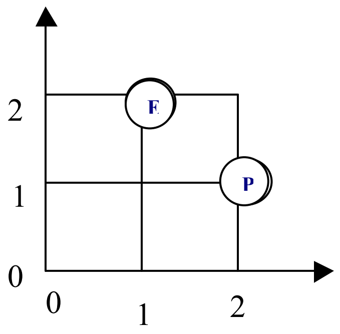
$$y_e(k+1) = PS(y_e(k))$$



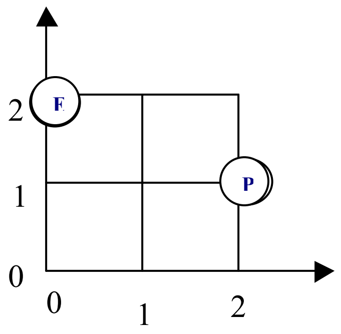
Before



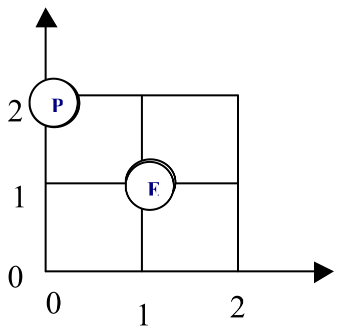
After



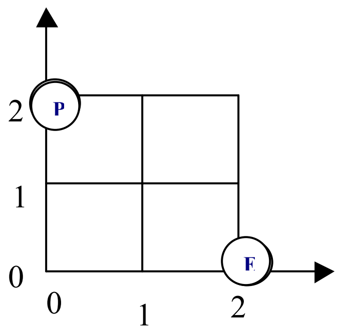
Before



After



Before



After

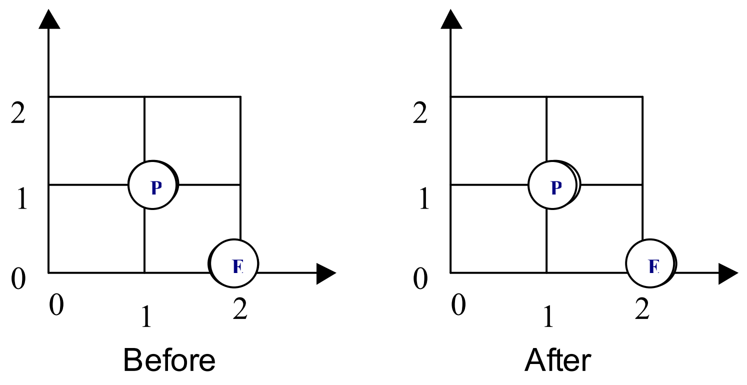


Fig. 4.11 Some examples of other conditions

<u>*Theorem 1*</u>: There are $N\times N$-3 final equilibrium states of the NxN pursuer-evader problem.

<u>*Proof*</u>: Figure 4.12 shows an example of the six final equilibrium states in case of N=3. The figure implies that the pursuer can be in any of the six positions to obtain a final equilibrium state. We can prove that the state when the pursuer is in (1,1) position is a final equilibrium state because of the 4(j) rule of the dynamics. For the other ones we can prove the same by using 4(a) rule.

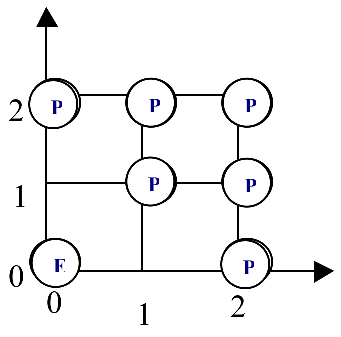


Fig. 4. 12 The six final equilibrium states for 3×3 grid.

For each given equilibrium-state the pursuer is free to choose its next move based on a finite set of possible actions. This finite set is a function of the state ***x***. If the state is a non-equilibrium state then the pursuer is not allowed to move in that time instant. The evader will move from the non-equilibrium-state to another state that could be (in general) another non-equilibrium-state or an equilibrium-state.

***Theorem* 2**: If for any positive integer k, $x_e(k) \neq x_p(k)$, and $y_e(k) \neq y_p(k)$, then for any k $t \geq k$, the following two statements can not be simultaneously false: $x_e(t) \neq x_p(t)$,and $y_e(t) \neq y_p(t)$.

Proof: This can be easily proven by noting that in order for both statements to be false at time t, the system would have to be in equilibrium condition, and then the pursuer would have to move to acquire the same coordinates as those of the evader. However, due to the constraint on the motion of the pursuer, that the pursuer can only move when the system is in a non-equilibrium state, the theorem is proven.

Let U be the discrete set of actions available to the pursuer when the system is in the state $\boldsymbol{x}$. The pursuer defines a *policy* $\mu : \mathbf{x} \rightarrow U$ that is a function from the state to actions. This defines a feedback control policy. We also define a *value function* $V_\mu(\mathbf{x})$, which is the sum of all future instantaneous costs given that the initial state of the system is **x** and the system follows the policy μ. We define *instantaneous cost* as:

$$c_{\mathbf{x}}(u) = \min\{|x_p(k) - x_p(k-1)| + |y_p(k) - y_p(k-1)| + |x_e(k) - x_e(k-1)| + |y_e(k) - y_e(k-1)|, 1\}$$

Notice that given any state $\boldsymbol{x}$(k), we can find out the next state if the control action u is known. The value function $V_\mu(\mathbf{x})$ is given by

$$V_\mu(i) = \sum_{k=0}^{\infty} c_{\mathbf{x}(k)}(\mu(\mathbf{x}(k))) \quad \textit{where} \quad \mathbf{x}(0) = i$$

Problem Statement: Find the *optimal policy* that minimizes the value function:

$$V^*(i) = \min_\mu V_\mu(i)$$

This gives us the *optimal value function*. In general, optimal value function is unique but an optimal policy might not be.

4.4. Properties of the Digraph Associated with the Pursuer-evader Problem

We can represent the pursuer-evader problem described above as a digraph **G** = (***V***, ***E***) that consists of a finite set ***V*** of vertices or nodes representing the states of the system, and a finite set E of edges. ***V*** consists of all the possible values of the state ***x***. The cardinality of ***V*** denoted by ***N***(***V***) is $(N\times N)^2$. There exists an edge *e* from a state-value (node) *v* to *w* if for some *k*, $v = \boldsymbol{x}(k)$ and $w = \boldsymbol{x}(k+1)$ following the dynamics generated by the rules in section 2. The digraph is a directed network or a weighted-digraph since we associate a cost with each edge using the instantaneous cost formula from section 2. The digraph is also simple, since there are no loops or multiple edges. The adjacency matrix of the digraph is an $N(V)\times N(V)$ matrix whose diagonal elements are all zeros.

Theorem 2: The instantaneous cost associated with each edge in the digraph of the pursuer-evader problem is 1.

Proof:

The proof of the theory comes directly from calculating the cost for the notion of the pursuer and the evader, according to their dynamics as given in section 2.

a) Far condition;

In this case, and by definition 1, the evader is at equilibrium state, so, only the pursuer is allowed to move;

$x_e(k+1) = x_e(k)\ \&\ y_e(k+1) = y_e(k)$

Since only the pursuer is allowed to move, and for only one step. Then, the minimum distance the pursuer can move, will result from moving it one step in either the *x* direction or the *y* direction;

$x_p(k+1) - x_p(k) = 1 \textit{ or } y_p(k+1) - y_p(k) = 1$

Substitute with these in the cost equation⇒

$$\begin{aligned} c_x(\mu) = \min(&|x_p(k+1) - x_p(k)| + |y_p(k+1) - y_p(k)| + \\ &|x_e(k+1) - x_e(k)| + |y_e(k+1) - y_e(k)|, 1) \end{aligned}$$

Thus;

$c_x(\mu) = \min(1+0+0+0,1) = 1 \textit{ or } c_x(\mu) = \min(0+1+0+0,1) = 1$

b) Left tope corner, pursuer right

In this case, the evader is not in an equilibrium state and therefore, only the evader is moving while not the pursuer;

$x_e(k) = 0 \, and \; y_e(k) = N$

$x_e(k+1) = 0 \, and \; y_e(k+1) = NS(y_e(k)) = NS(N) = N-1$

$x_p(k) = PS(x_e(k)) = PS(0) = 1 \, and \; y_p(k) = N$

$x_p(k+1) = x_p(k) = 1 \, and \; y_p(k+1) = y_p(k) = N$

Thus,

$\therefore c_x(\mu) = \min(0+0+0+1,1) = 1$

c) Left top corner, pursuer down condition

$x_e(k) = 0 \, and \; y_e(k) = N$

$x_p(k) = 0 \, and \; y_p(k) = NS(N) = N-1$

$x_e(k+1) = PS(x_e(k)) = 1 \, and \; y_e(k+1) = y_e(k) = N$

$x_p(k+1) = x_p(k) = 0 \, and \; y_p(k+1) = y_p(k) = N-1$

Thus,

$\therefore c_x(\mu) = \min(0+0+1+0,1) = 1$

d) Right top corner, pursuer left condition

$x_e(k) = N \, and \; y_e(k) = N$

$x_p(k) = NS(x_e(k)) = N-1 \; and \; y_p(k) = N$

$x_e(k+1) = x_e(k) = N \; and \; y_e(k+1) = NS(y_e(k)) = N-1$

$x_p(k+1) = x_p(k) = N-1 \; and \; y_p(k+1) = y_p(k) = N$

Thus,

$\therefore c_x(\mu) = \min(0+0+1+0,1) = 1$

e) Right top corner, pursuer down condition

$x_e(k) = N \; and \; y_e(k) = N$

$x_p(k) = x_e(k) = N \; and \; y_p(k) = NS(y_e(k)) = N-1$

$x_e(k+1) = NS(x_e(k)) = N-1 \; and \; y_e(k+1) = y_e(k) = N$

$x_p(k+1) = x_p(k) = N \; and \; y_p(k+1) = y_p(k) = N-1$

Thus,

$\therefore c_x(\mu) = \min(0+0+1+0,1) = 1$

f) Left bottom corner, pursuer right condition

$x_e(k) = 0 \text{ and } y_e(k) = 0$
$x_p(k) = PS(x_e(k)) = 1 \text{ and } y_p(k) = 0$
$x_e(k+1) = x_e(k) = 0 \text{ and } y_e(k+1) = PS(y_e(k)) = 1$
$x_p(k+1) = x_p(k) = 1 \text{ and } y_p(k+1) = y_p(k) = 0$

Thus,

$\therefore c_x(\mu) = \min(0+0+0+1,1) = 1$

g) Left bottom corner, pursuer up condition

$x_s(k) = 0 \text{ and } y_s(k) = 0$
$x_d(k) = x_s(k) = 0 \text{ and } y_d(k) = PS(y_s(k)) = 1$
$x_s(k+1) = PS(x_s(k)) = 1 \text{ and } y_s(k+1) = y_s(k) = 0$
$x_d(k+1) = x_d(k) = 0 \text{ and } y_d(k+1) = y_d(k) = 1$

Thus,

$\therefore c_x(\mu) = \min(0+0+1+0,1) = 1$

h) Right bottom corner, pursuer left condition

$x_e(k) = N \text{ and } y_e(k) = 0$
$x_p(k) = NS(x_e(k)) = N-1 \text{ and } y_p(k) = 0$
$x_e(k+1) = x_e(k) = N \text{ and } y_e(k+1) = PS(y_e(k)) = 1$
$x_p(k+1) = x_p(k) = N-1 \text{ and } y_p(k+1) = y_p(k) = 0$

Thus,

$\therefore c_x(\mu) = \min(0+0+0+1,1) = 1$

i) Right bottom corner, pursuer up condition

$x_e(k) = N \text{ and } y_e(k) = 0$
$x_p(k) = x_e(k) = N \text{ and } y_p(k) = PS(y_e(k)) = 1$
$x_e(k+1) = NS(x_e(k)) = N-1 \text{ and } y_e(k+1) = y_e(k) = 0$
$x_p(k+1) = x_p(k) = N \text{ and } y_p(k+1) = y_p(k) = 1$

Thus,

$\therefore c_x(\mu) = \min(0+0+1+0,1) = 1$

j) Other conditions:

For all the cases mentioned in section 2 to *j*, the evader takes only one step at a time away from the pursuer which is not allowed to move since the evader in not in an equilibrium state, as illustrated in figure 4.11. Therefore, in all cases we have;

$x_e(k+1) - x_e(k) = 1 \; or \; y_e(k+1) - y_e(k) = 0 \Rightarrow$

$\therefore c_x(\mu) = \min(0+0+1+0,1) = 1$

or $x_e(k+1) - x_e(k) = 0 \; or \; y_e(k+1) - y_e(k) = 1 \Rightarrow$

$\therefore c_x(\mu) = \min(0+0+1+1,1) = 1$

or $x_e(k+1) - x_e(k) = 1 \; or \; y_e(k+1) - y_e(k) = 1 \Rightarrow$

$\therefore c_x(\mu) = \min(0+0+1+1,1) = 1$

Theorem 3: The digraph of the pursuer-evader problem is not a strongly connected digraph, but is weakly connected. Moreover, it is not a unilaterally connected digraph.
Proof: It can be shown that starting from any allowable (all states except the ones with co-incident positions for pursuer and evader) state, one of the final equilibrium states can be reached. All the final equilibrium states have paths connecting them together. To see this, consider a final equilibrium state, and then move the pursuer back (to increase he distance between the pursuer and the evader). This action will not result in any evader movement. Then we can move the pursuer in positions that have distance more than one from the evader (at the pen). Then the pursuer can be moved to a different position corresponding to another final equilibrium position. This shows that starting from any initial allowable state, there is a path to all the final equilibrium states. This proves that the underlying graph of the digraph is connected. It is also a unilaterally connected digraph for the same reason. To show that it is not strongly connected, consider any final equilibrium state. From these states, there is no pursuer action that can take the evader from the boundary of the two dimensional space into the interior.

Some additional properties of the pursuer-evader digraph are given below:

1. The number of nodes that are adjacent from a node representing an equilibrium state depends on the location of the pursuer position in the grid. There are the following three possibilities on the number of adjacent states.

a) There are eight states adjacent from the equilibrium state node if the pursuer position is in the interior. Only seven out of the eight are allowed since pursuer and evader are not allowed to have the same location.

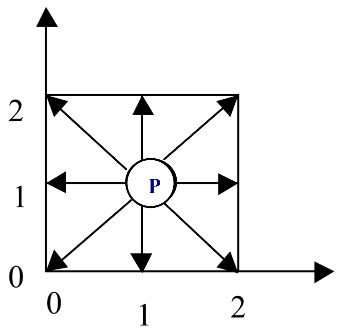


Fig. 4.13 Adjacent states for pursuer in the interior

b) There are five states adjacent from the equilibrium state node if the pursuer position is in the side but not in a corner.

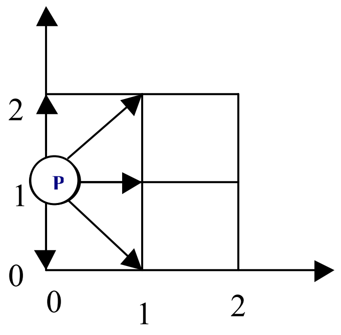


Fig. 4.14 Adjacent states for pursuer in the side (not corner)

c) There are three states adjacent from the equilibrium state node if the pursuer position is in the corner.

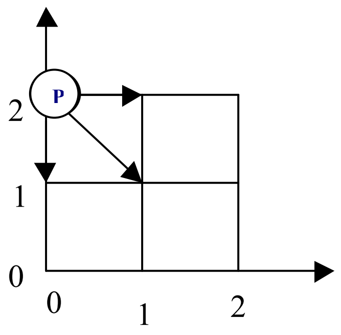


Fig. 4.15 Adjacent states for pursuer in the corner

2. The number of nodes adjacent to a node representing a non-equilibrium state is one. The state adjacent from the non-equilibrium node can be another non-equilibrium node or an equilibrium node.

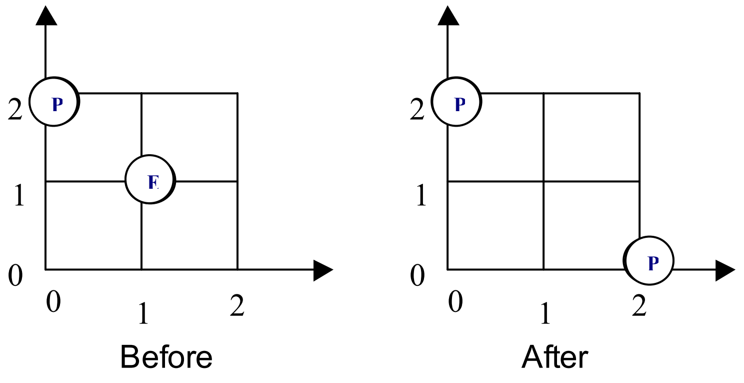


Fig. 4.16 Adjacent states for non-equilibrium initial state

4.5. Proposed Techniques of Solution to the NxN Grid Pursuer-Evader Problem

The dynamic programming solution is based on Bellman's equation, which for our problem would look like the following:

$$V^*(\mathbf{x}(k)) = \min_{u\in\mu(\mathbf{x})}\{c_i(u) + V^*(\mathbf{x}(k+1))\}$$

This equation indicates how the feedback controller can make decisions once the value function is available. This equation can also be used to find the value function using the boundary conditions from the problem.

We provide the solution to the problem using two different algorithms directions. In the first direction and based on the fact that finite state systems can always be represented by an acyclic graph with finite number of nodes, then the deterministic systems problem becomes equivalent to finding the shortest path from an initial node x_0 to a terminal node x_N. Unfortunately, since non of the most common shortest path techniques, like label correcting techniques [47], and auction algorithms [48], deals with cases like ours, where we have a multiple terminal states whose number depends on the grid size. Therefore, we modified one of the most popular shortest path algorithms to fit our case. Two modified versions of Dijkstra's shortest path algorithm are considered to deal with the case of multi-terminal state case in our problem. The first algorithm uses Dijkstra's algorithm for each final equilibrium state and then uses minimization over all final equilibrium states to obtain the value function. It is so obvious that such an algorithm will increase the complexity of the original Dijkstra's by the number of the final equilibrium states, as will be illustrated later, which is impractical specially when dealing with large grid size. This gave us a motive to introduce another modification of Dijkstra's algorithm to suit our case. The basic idea of that one is to introduce a new definition of the distance of a node from a set of nodes, as it will be explained in the following section. Finally, the last algorithm directly uses dynamic programming to sequentially obtain the value function. The three algorithms are described after we define some terminology and some sub-algorithms that will be used by the three main algorithms.

The following terminology is adapted from [49]. For the digraph **G**=(***V***,***E***), weight function maps edges to weights as $w: E \rightarrow 1$. If a node v is adjacent from node u, we show that as $u \rightarrow v$. If there exists a path between a node u and node v possibly through other nodes, it is shown as $u \xrightarrow{p} v$. Weight of a path $p = (v_0, v_1, ..., v_k)$ is the sum of all the included edge weights, given as:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

The shortest path weight from a node u to node v is $\delta(u,v)$ defined by:

If a path from u to v exists, then

$$\delta(u,v) = \min_p (w(p) : u \xrightarrow{p} v)$$

else

$$\delta(u,v) = \infty$$

Any path p from node u to node v in G(V,E) with weight $w(p) = \delta(u,v)$ is the shortest path from u to v.

<u>Algorithm</u> <u>1</u>: INITIALIZE (**G**,s)

Given a source node $s \in V[G]$

```
        $\forall v \in V[G]$
        do $d[v] := \infty$
            $d[s] := 0$
```

Here $:=$ is the assignment operator.

<u>Algorithm</u> <u>2</u>: RELAX(u,v)

```
        if $d[v] > d[u] + w(u,v)$
        then $d[v] := d[u] + w(u,v)$
```

<u>Algorithm</u> <u>3</u>: DIJKSTRA(**G**,s)

Note: $S \subset V, S = \{v : d[v] = \delta(s,v)\}$

```
        INITIALIZE (G,s)
        S := ∅
        Q : = V[G]
```

```
while $\mathbf{Q} \neq \varnothing$
    do u := EXTRACT-MIN(Q)
    $\mathbf{S} := \mathbf{S} \cup \{u\}$
    for each vertex $v \in \mathrm{Adj}[u]$
        do RELAX($u$,$v$)
```

In this algorithm, $u \in \boldsymbol{V} - \boldsymbol{S}$ is the vertex with the shortest path estimate in Q that contains all vertices in $\boldsymbol{V} - \boldsymbol{S}$ sorted by their d values.

4.5.1. Algorithm based on Dijkstra's Shortest Path Solution

For convenience, we label the nodes of the digraph as follows. Notice that the state value are given as $\mathbf{x}(k) = [x_e(k)\ y_e(k)\ x_p(k)\ y_p(k)]$ where we have $\forall k \quad x_p(k) \in \{0,1,2,....N\}$, $y_p(k) \in \{0,1,2,.....,N\}$, $x_e(k) \in \{0,1,2,.....,N\}$ and $y_e(k) \in \{0,1,2,.....,N\}$. We can consider the state to be a N-digit ternary number with the least significant digit being $y_p(k)$, first being $x_p(k)$, second being $y_e(k)$, and the most significant one being $x_e(k)$. The label for the node is simply the decimal value of the ternary number. That is, if we use the variable n for the node label with state given by $\mathbf{x} = [x_e\ y_e\ x_p\ y_p]$, then the following is true.

$$n = x_e.N^3 + y_e.N^2 + x_p.N^1 + y_p.N^0$$

4.5.1.1. *Modified Dijkstra's* 1

Assuming that the number of the final equilibrium states is m, and the overall number of states is n, another way of utilizing Dijkstra's algorithm to solve the problem in hand may be introduced. This technique is based on calculating the shortest path between any state $v_i \in V\text{-}S\ \forall i=1,2,\ldots n\text{-}m$, and all the final equilibrium states $s_j \in S\ \forall j=1,2,\ldots,m$, then taking from the M calculated distances, the one with minimum path. In other words, the algorithm is repeated m times for each final equilibrium state and the path with least weight is assigned to that state v_i and the corresponding final equilibrium state m_j.

Let $S = \{s(i,1) : i = 3,5,6,...,m\}$ …i.e. the set of all the final equilibrium states.

Then,

$$d[v_j] = \min.\{d[v_j, s_i] \forall i = 1,2,.....m\}$$

Like Dijkstra's algorithm, this algorithm produces a set of vertices ***X*** whose final shortest path distances from the source set S is determined. That is, for all $v_I \in \boldsymbol{X}$, we have $d[v_j]=\delta(S, v_j)$. The algorithm repeatedly selects the vertex $v_i \in V\text{-}S$ with minimum shortest path estimate, inserts v_j into ***X*** and relaxes all the edges leaving v_j. At the end, we'll have a queue ***Q*** that contains all the vertices in *V-S* with their corresponding distance value from the source set S.

- ***Algorithm***

```
Initialize source set $d[s_i]=0 \ \forall i=1,2,.....,m;$
For i=1:m
    Initialize the weights of the V-S=$\infty$.
    Set $\boldsymbol{Q}=\boldsymbol{V}\text{-}\boldsymbol{S}$
    While $\boldsymbol{Q} \neq \phi$
        If $d[v,s_i] \leq d[s_i]+d[v, s_i]$,
          then $X=Xuv$
           $d[v]=d[v,S]$
        for each vertex $u \in ADJ(v)$
        Relax (u,v,d)
    End
End
For i=1:n-m
 $d_{min}=min.d[v_I,s_k] \ \forall k=1,,2,....m$
$d[v_i,s_k]=d_{min}$
End
```

4.5.1.2. *Modified Dijkstra's 2*

Since Dijkstra's algorithm solves the single source shortest paths problem on a weighted directed graph **G**=(***V***,***E***) for non-negative weights case. A modified version of this algorithm can be used to deal with our problem after introducing the definition of a vertex "state" from a set of vertices "states".

Let $S = \{s(i,1) : i = 3,5,6,..., N \times N\}$ …i.e. the set of all the final equilibrium states.

<u>*Definition*</u>: The distance of a vertex v_I from a set of vertices S is defined as:

$$d[\mathbf{S}, v_i] = \min.\{d[v_i, s_j] \forall v_i \in \mathbf{V} - \mathbf{S}, s_j \in \mathbf{S}\}$$

Like Dijkstra's algorithm, this algorithm produces a set of vertices ***X*** whose final shortest path distances from the source set ***S*** is determined. That is, for all $v_i \in \boldsymbol{X}$, we have $d[v_j, \mathbf{S}] = \delta(\mathbf{S}, v_j)$. The algorithm repeatedly selects the vertex $v_i \in \boldsymbol{V}\text{-}\boldsymbol{S}$ with minimum shortest path estimate, inserts v_i into ***X*** and relaxes all the edges leaving v_j. At the end, we'll have a queue ***Q*** that contains all the vertices in ***V-S*** with their corresponding distance value from the source set ***S***.

➤ <u>***Algorithm***</u>

```
Initialize source set $d[\mathbf{S}]=0$;
Initialize the weights of the $\mathbf{V}\text{-}\mathbf{S}=\infty$.
Set $\mathbf{Q}=\mathbf{V}\text{-}\mathbf{S}$
While $\mathbf{Q} \neq \phi$
    If $d[v,\mathbf{S}] \leq d[\mathbf{S}]+d[v,\mathbf{S}]$,
      then $\mathbf{X}=\mathbf{X} \cup v$
       $d[v]=d[v,\mathbf{S}]$
    for each vertex $u \in ADJ(v)$
    Relax (u,v,d)
END
```

4.5.2. Direct Dynamic Programming Solution

This approach utilizes the dynamic programming tool directly to evaluate the minimum cost-to-go value function from any of the non-equilibrium states to the final equilibrium (terminal) states over the set of admissible control actions. Assume

$c_{ij}(x_i,u_j)$ to be the cost for transition from state x_i to state x_j by applying control u_j.

$c(x_N)$ to be the v-value of the final equilibrium states, (zero in our case).

Then, working backward form the final equilibrium states to all the other states, and applying the following dynamic programming algorithm at each state, we can evaluate the minimum value or minimum cost to go value function of any state to the terminal ones.

$V(x_i)=c(x_N)$

$$V(x_i) = \min_{u_{ij}\in U(x_i)} \{c_{ij}(x_i,u_{ij}) + V(x_j)\} \qquad \forall i = 1,2,\ldots\ldots, N\times N-(N-3)$$

where x_j…represent the set of all adjacent states to x_i with the application of control u_{ij}.

4.6. Simulation Results

Simulating our system begins by computing the value of the cost function of each state using one of the solution techniques mentioned above. Table 4.1, shows the values of the cost function for a 3×3-grid using modified Dijkstra's 2, V_{dj}, and direct dynamic programming techniques V_{dp}. As shown from the results that both techniques gives exactly the same cost function value. The state number corresponds to every possible combination of the x and y coordinates of the pursuer and evader positions. The evader movements are controlled by the dynamics defined in section 2. Meanwhile, the pursuer makes its transitions based on the cost function value of the adjacent states to the current state of the system. The pursuer moves to the state of the lowest cost of all adjacent states, then it checks whether the system is at equilibrium or not to make its next move. This process continues till the system reaches one of the final equilibrium states defined in section 2.

A graphical user interface is used to simulate the system where the user is to choose the grid size, N, from a drop box. The intial position of the pursuer and the evader is supplied to the simulation programm using an edit box. Then, the technique used to calculate the cost to go (value) function of each state is choosen using a check box. Finally, simulation starts by pressing the START button.

S	V_{dj}	V_{dp}	S	V_{dj}	V_{dp}	S	V_{dj}	V_{dp}	S	V_{dj}	V_{dp}	S	V_{dj}	V_{dp}	S	V_{dj}	V_{dp}	S	V_{dj}	V_{dp}
1	*Inf*	*Inf*	*13*	*5*	*5*	*25*	*4*	*4*	*37*	*0*	*0*	*49*	*2*	*2*	*61*	*Inf*	*Inf*	*73*	*0*	*0*
2	*5*	*5*	*14*	*6*	*6*	*26*	*5*	*5*	*38*	*2*	*2*	*50*	*3*	*3*	*62*	*7*	*7*	*74*	*2*	*2*
3	*4*	*4*	*15*	*7*	*7*	*27*	*6*	*6*	*39*	*3*	*3*	*51*	*Inf*	*Inf*	*63*	*6*	*6*	*75*	*3*	*3*
4	*5*	*5*	*16*	*4*	*4*	*28*	*6*	*6*	*40*	*2*	*2*	*52*	*3*	*3*	*64*	*0*	*0*	*76*	*2*	*2*
5	*8*	*8*	*17*	*5*	*5*	*29*	*5*	*5*	*41*	*Inf*	*Inf*	*53*	*4*	*4*	*65*	*2*	*2*	*77*	*1*	*1*
6	*6*	*6*	*18*	*6*	*6*	*30*	*4*	*4*	*42*	*5*	*5*	*54*	*5*	*5*	*66*	*3*	*3*	*78*	*4*	*4*
7	*4*	*4*	*19*	*0*	*0*	*31*	*Inf*	*Inf*	*43*	*3*	*3*	*55*	*0*	*0*	*67*	*1*	*1*	*79*	*3*	*3*
8	*6*	*6*	*20*	*1*	*1*	*32*	*6*	*6*	*44*	*5*	*5*	*56*	*3*	*3*	*68*	*3*	*3*	*80*	*4*	*4*
9	*7*	*7*	*21*	*Inf*	*Inf*	*33*	*5*	*5*	*45*	*6*	*6*	*57*	*4*	*4*	*69*	*4*	*4*	*81*	*Inf*	*Inf*
10	*6*	*6*	*22*	*3*	*3*	*34*	*8*	*8*	*46*	*0*	*0*	*58*	*1*	*1*	*70*	*2*	*2*			
11	*Inf*	*Inf*	*23*	*5*	*5*	*35*	*7*	*7*	*47*	*1*	*1*	*59*	*5*	*5*	*71*	*Inf*	*Inf*			
12	*8*	*8*	*24*	*7*	*7*	*36*	*6*	*6*	*48*	*2*	*2*	*60*	*5*	*5*	*72*	*5*	*5*			

Table 4.1 The cost function values for a 3×3 grid, using modified Dijkstra's 2, V_{dj}, and dynamic programming technique, V_{dp}.

Figures 4.17 (a) and (b), along with Figure 4.18 (a) and (b) show the used graphical user interface provided to simulate the system where Figure 4.17 (a), and (b) shows a single run with intial position of the pursuer is at (1,1) and the intial position of the evader is at (3,4) where the cost values is calculated using modified Dijkestra 2 and dynamic programming techinques recpectively. Meanwhile, Figure 4.18 (a), and (b) illustrates different same intial positions for the pursuer and the evader, and again the cost value is calculated this time using modified Dijkestra 2 and dynamic programming techinques recpectively.

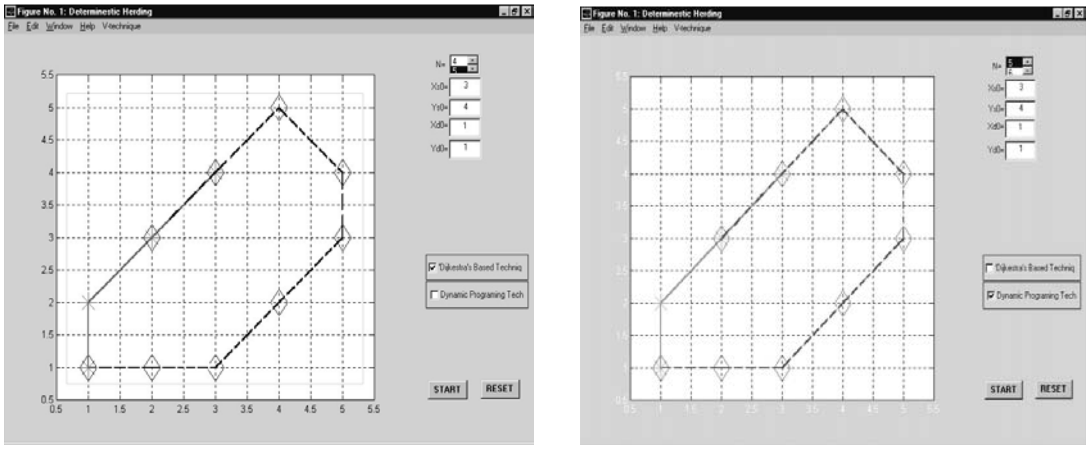


Figure No. 1: Deterministic Herding
File Edit Window Help V-technique
START
RESET

N= 4
Xs0= 3
Ys0= 4
Xd0= 1
Yd0= 1
[✓ Dijkestra's Based Techniq]
Dynamic Programing Tech

N= 5
Xs0= 3
Ys0= 4
Xd0= 1
Yd0= 1
Dijkestra's Based Techniq
[✓ Dynamic Programing Tech]

(a) (b)

Fig 4.17 GUI for simulating the system with value function calculated using modified Dijkstra's 2 in (a) and dynamic programming in (b)

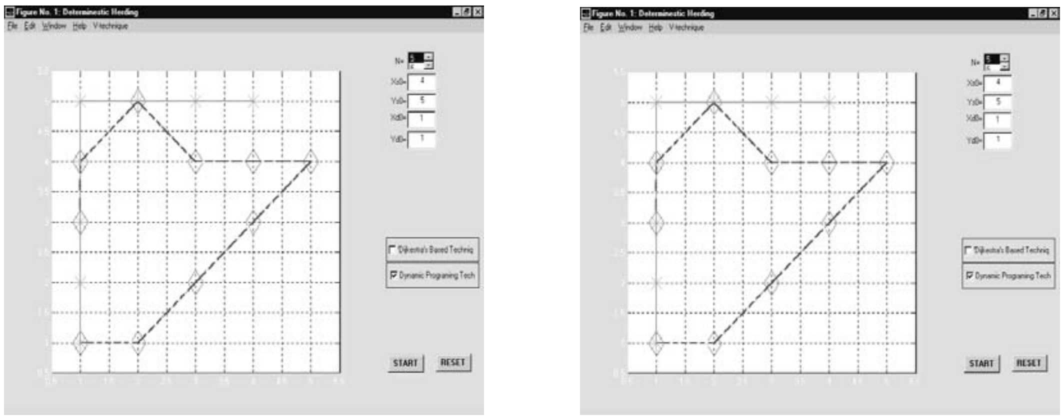


Figure No. 1: Deterministic Herding
File Edit Window Help V-technique
START
RESET

N= 5
Xs0= 4
Ys0= 5
Xd0= 1
Yd0= 1
Dijkestra's Based Techniq
[✓ Dynamic Programing Tech]

N= 5
Xs0= 4
Ys0= 5
Xd0= 1
Yd0= 1
Dijkestra's Based Techniq
[✓ Dynamic Programing Tech]

Fig 4.18 GUI for simulating the system with value function calculated using modified Dijkstra's 2 in (a) and dynamic programming in (b) for different initial positions

4.7. Summary

In this chapter we introduced the main building block of the research, in its simplest form, single evader and passive single pursuer in a completely deterministic system.

Also, after mathematically formulating the problem, the proposed solution techniques are introduced and finally some simulation results that support the theoretical solution techniques are shown.

Chapter (5)

Pursuit Evasion: The Stochastic Model

5.1. Introduction

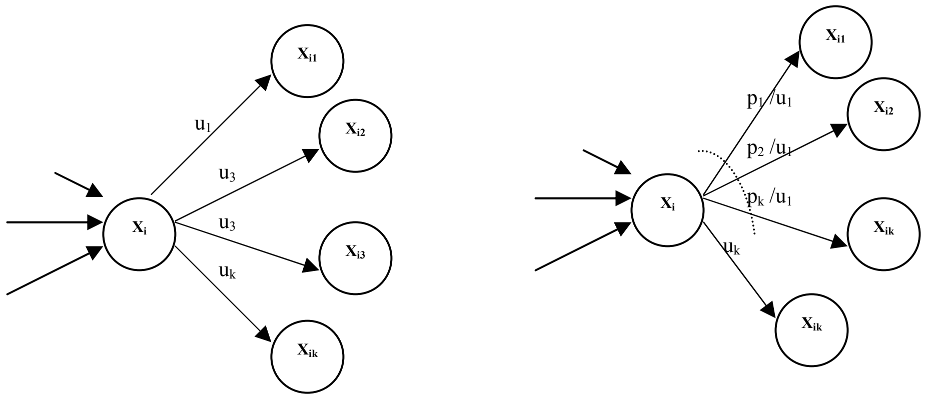
This chapter is a continuation to the previous one concentrating on studying the pursuit evasion problem. The modified problem we study in this chapter involves a "pursuer" agent herding an "evader" agent -moving stochastically- in order to drive it to a pen. The problem is stated in terms of allowable sequential actions of the two agents. The solution is obtained by applying the principles of stochastic dynamic programming. Three algorithms for solution are presented with their accompanying results.

Other techniques have been considered by researchers, for some similar problems. For instance, the hamstrung squad car game and the homicidal chauffeur game where the reduced space technique was used to evaluate the value of the cost function have been studied in [50]. The reduced space technique proved to create an increase in complexity not only with the increase of the number of players, but with the increase of systems dynamics complexity also [50].

This chapter provides a brief introduction to the principles of stochastic dynamic programming since it is the candidate optimization technique that is going to be used. Then we introduce the system dynamics of the problem. After that, properties of the di-graph representing the problem model are explained. Later, the problem statement is specified. Finally, the proposed solution techniques are introduced with the supporting simulation results.

5.2. Principles of Stochastic Dynamic Programming

Stochastic dynamic programming principles are introduced in analogy with those of deterministic dynamic programming to illustrate the variations between both algorithms. These differences come from the stochastic nature of transitions between states. In deterministic dynamic programming, application of any control $u_i \in \boldsymbol{U}$ results in transition from one state, say x_i to state x_j such that x_i & $x_j \in \boldsymbol{X}$. Meanwhile, in stochastic model application of any control $u_i \in \boldsymbol{U}$ results in transition from x_j to a set $\boldsymbol{s}_i \subset \boldsymbol{X}$ with a pre-determined probability distribution $\boldsymbol{P}$. Therefore, the cost of each state is dependent not only on the set of adjacent states but also on the transition probabilities. This can be illustrated by figures 5.1 (a) and (b).



(a) Deterministic (b) Stochastic

Fig. 5.1. Deterministic and Stochastic transitions

The stochastic version of dynamic programming can be obtained from the deterministic one by introducing a stochastic variable w_k into the transition operator as follows;

- A discrete system whose state transition is given by;

  $x_{k+1} = f(x_k, u_k, w_k)$

- An independent random disturbance parameter w_k with given probability distribution.
- A control constraint $u_i \in \boldsymbol{U}$.

- An additive cost function of the form;

  $$V = E\{c_N(x_N) + \sum_{k=1}^{N-1} \tilde{c}_k(x_k, u_k, w_k)\}$$

  where $\tilde{c}_k$ is a function that gives the stochastic inter-state transition cost and c_N is the cost associated with the terminal state.

Thus, the stochastic dynamic programming algorithm may be stated as follows;
Denoting the optimal cost value by $V^*(x_0)$, then

$$V^*(x_0) = V_0(x_0)$$

whereas, the value of the cost function $V_0(x_0)$ is given by the last step of the following algorithm which proceeds backward in time form instant *N*-1 to 0;

$$V_N(x_N) = c_N(x_N)$$

$$V_k(x_k) = \min_{u \in U_k} \underset{w_k}{E}\left(c_x(x_k, u_k, w_k) + \sum_{j=1}^{N-1} p_{kj} V_j(f(x_k, u_k, w_k)) \right), \qquad k=0,1,2,\ldots, N-1$$

Moreover,

$$u^*{}_k(x_k) = \arg\{\min_{u_k \in U} V^*{}_k(x_k)\}$$

is the optimal control policy.

The sequential interpretation of the algorithm can be illustrated by the figure 5.2

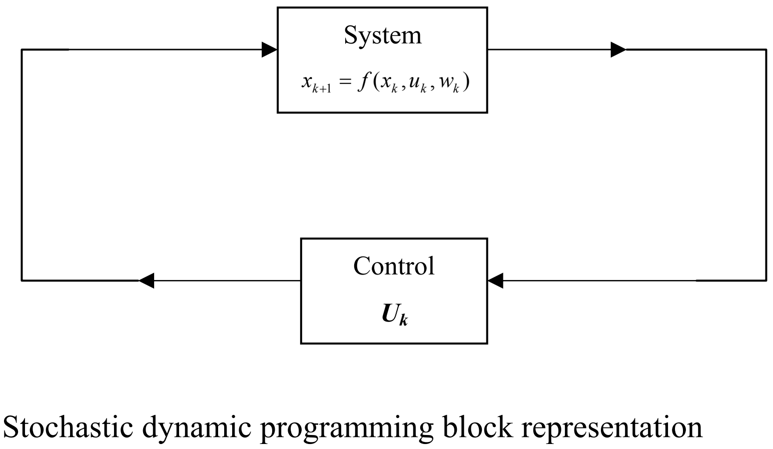


Fig. 5.2 Stochastic dynamic programming block representation

5.3. A N×N Stochastic Pursuer-Evader problem

We consider the pursuer-evader problem in an $N \times N$ to introduce the dynamics assumptions based on them the solution is attained. The pursuer can occupy one of the N positions and so may the evader with probabilities that depend on the pursuer location, such that both of them are not allowed to be in the same position at the same time. The ultimate objective of the pursuer is to drive the evader to the pen, (0,0) position, in minimum expected time. Therefore, the state vector at time k, $\boldsymbol{x}(k)$, is determined by the position of the evader and the pursuer, i.e

$\boldsymbol{x}(k)=[x_e(k)\ \ y_e(k)\ x_p(k)\ y_p(k)\ ]$

where,

$x_e(k)$ …. The x coordinate of the evader at time k.

$y_e(k)$ …. The y coordinate of the evader at time k.

$x_p(k)$ …. The x coordinate of the pursuer at time k.

$y_p(k)$ …. The y coordinate of the pursuer at time k.

So, at any time k, we have $x_p \in \{0,1,2….N\}$, $y_p \in \{0,1,2…N\}$, $x_e \in \{0,1,2,…..N\}$,and $y_e \in \{0,1,2,………N\}$. However, based on the dynamics and as it will be illustrated later, with the pursuer and the evader not being in the same initial state, they never can be in the same location. A cost of one unit is assigned for each step (horizontal, vertical, or diagonal) for the pursuer as well as the evader. Fig. 5.3 below illustrates the $N \times N$ spatial grid of the pursuer-evader problem.

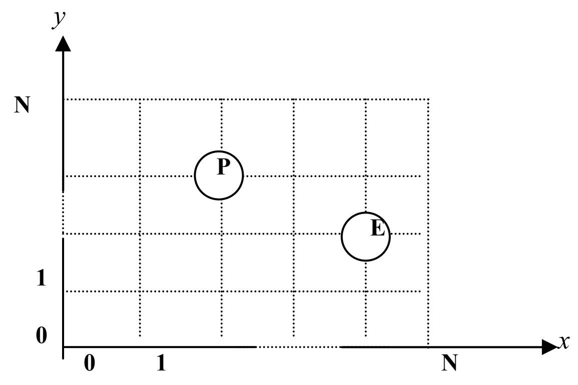


Fig. 5.3. The N ×N pursuer-evader problem grid.

Definition* 1;Positive successor function:* Positive successor function is given by:

$$PS(X(k)) = \begin{cases} X(k) & \text{if } X(k) = N \\ X(k)+1 & \text{if } 0 \le X(k) \le N \end{cases} \tag{1}$$

where,

$x(k)$ is the x or y coordinate of either the pursuer or the evader.

Thus, $PS(.)$: $X=\{0,1,2,\ldots N\} \rightarrow Y=\{1,2,\ldots N\}$

Definition* 2; Negative successor function:* Negative successor function is given by:

$$NS(X(k)) = \begin{cases} X(k) & \text{if } X(k) = 0 \\ X(k)-1 & \text{if } 0 \le X(k) \le N \end{cases} \tag{2}$$

$x(k)$ is the x or y coordinate of either the pursuer or the evader.

Thus, $NS(.)$: $X=\{0,1,2,\ldots N\} \rightarrow Y=\{0,1,2,\ldots N\text{-}1\}$

Definition* 3; Equilibrium state of the evader:* The evader is in equilibrium state when given a time instant T, if one of the following conditions is satisfied;

a. Far condition: $x_p(T)$-$x_e(k)$ and/or $y_p(T)$-$y_e(k)$>1.
b. Left boundary condition: $x_e(k)=0$, $0<y_e(k)<N$ with $y_p(k)= y_s(k)$, and $x_p(k)=PS(x_e(k))$.
c. Right boundary condition: $x_e(k)=2$, $0<y_e(k)<N$ with $y_p(k)= y_e(k)$, and $x_p(k)=NS(x_e(k))$.
d. Low boundary condition: $y_e(k)=0$, $0<x_e(k)<N$ with $x_p(k)= x_e(k)$, and $y_p(k)=PS(y_e(k))$.
e. Upper boundary condition: $y_e(k)=N$, $0<x_e(k)<N$ with $x_p(k)= x_e(k)$, and $y_p(k)=NS(y_e(k))$.
f. Upper left corner condition: $(x_e(k), y_e(k))=(0,N)$, and $x_p(k)=PS(x_e(k))$, and $y_p(k)=NS(y_e(k))$.
g. Upper right corner condition $(x_e(k), y_e(k))=(N,N)$, and $x_p(k)=NS(x_e(k))$, and $y_p(k)=NS(y_e(k))$.
h. Low right corner condition $(x_e(k), y_e(k))=(N,0)$, and $x_p(k)=NS(x_e(k))$, and $y_p(k)=PS(y_e(k))$.

***Definition* 4**; *Final equilibrium state of the evader:* The evader is in final equilibrium state at time instant T, when the following condition is satisfied:

$x_p(k)=x_d(T)$ and $y_p(k)=y_p(T)$, then $x_e(k)=0$ and $y_e(k)=0$. $\forall k>T$.

The following rules generate the pursuer-controlling movements and assign probabilities to the evader transitions based on its relative location with respect to the pursuer:

d. $\forall k\, x_p(k), y_p(k), x_e(k)$, and $y_e(k) \in \{0,1,2,\ldots\ldots N\}$.

e. The pursuer moves when the evader is at an equilibrium state only.

f. The pursuer can move one step at a time, depending on its position in the grid, and its relative location with respect to evader position as illustrated in figures 5.4 and 5.5 below:

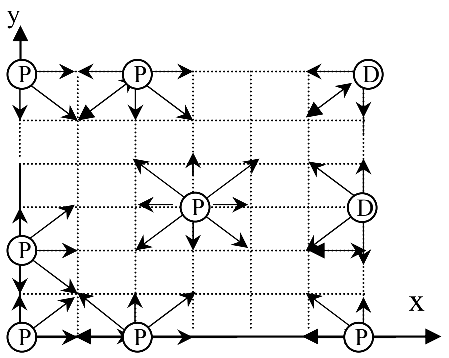


Fig. 5.4. Pursuer movements with distance between pursuer and evader>2.

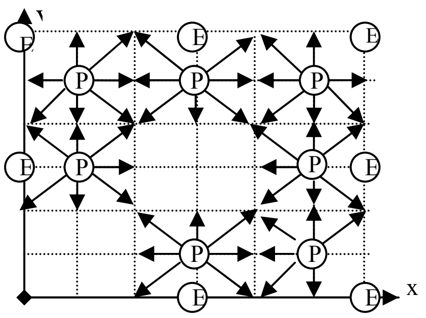


Fig.5.5. Pursuer movements with evader at equilibrium state.

The evader transition probabilities depend on its relative position to the pursuer as following:

a. Far Condition: If $x_e(k)<NS(x_p(k))$ or $x_e(k)>PS(x_p(k))$ or $y_e(k)<NS(y_p(k))$ or $y_e(k)>PS(y_p(k))$, then $x_e(k+1)=x_p(k)$ and $y_e(k+1)=y_p(k)$.

b. Left top corner pursuer right condition : If $x_e(k)=0$ and $x_p(k)=PS(x_e(k))$ and $y_e(k)=y_p(k)=N$ then ;

$P\{x_e(k+1)=x_e(k)$ and $y_e(k+1)=NS(y_e(k))|\ x_p(k)$and $y_p(k)\}=p.$

$P\{x_e(k+1)=PS(x_e(k))$ and $y_e(k+1)=NS(y_e(k))|\ x_p(k)$and $y_p(k)\}=(1-p).$

c. <u>Left top corner pursuer down condition</u>: If $x_e(k)= x_p(k)=0$ and $y_p(k)=NS(y_e(k))$ and $y_e(k)= N$ then ;

$P\{x_e(k+1)=PS(x_e(k))$ and $y_e(k+1)= y_e(k)\ |\ x_p(k)$and $y_p(k)\}=p.$

$P\{x_e(k+1)=NS(x_e(k))$ and $y_e(k+1)=NS(y_e(k))|\ x_p(k)$and $y_p(k)\}=(1-p).$

d. <u>Right top corner pursuer left condition:</u> If $x_e(k)=N$ and $x_p(k)=NS(x_e(k))$ and $y_e(k)=y_p(k)=N$ then ;

$P\{x_e(k+1)=x_e(k)$ and $y_e(k+1)=NS(y_e(k))|\ x_p(k)$and $y_p(k)\}=p.$

$P\{x_e(k+1)=NS(x_e(k))$ and $y_e(k+1)=NS(y_e(k))|\ x_p(k)$and $y_p(k)\}=(1-p).$

e. <u>Right top corner pursuer down condition:</u> If $x_e(k)= x_p(k)=N$ and $y_p(k)=NS(y_e(k))$ then ;

$P\{x_e(k+1)=NS(x_e(k))$ and $y_e(k+1)= y_e(k)\ |\ x_p(k)$and $y_p(k)\}=p.$

$P\{x_e(k+1)=NS(x_e(k))$ and $y_e(k+1)=NS(y_e(k))|\ x_p(k)$and $y_p(k)\}=(1-p).$

f. <u>Left bottom corner pursuer right condition:</u> If $x_e(k)= 0$ and $x_p(k)=PS(x_e(k)\ )$ and $y_p(k)=y_e(k)=0$ then ;

$P\{x_e(k+1)=x_e(k)$ and $y_e(k+1)= PS(y_e(k))\ |\ x_p(k)$and $y_p(k)\}=p.$

$P\{x_e(k+1)=PS(x_e(k))$ and $y_e(k+1)=PS(y_e(k))|\ x_p(k)$and $y_p(k)\}=(1-p)$

g. <u>Left bottom corner pursuer up condition:</u> If $x_e(k)= x_p(k)= 0$ and $y_p(k)=PS(y_e(k)\ )$ then ;

$P\{x_e(k+1)=PS(x_e(k))$ and $y_e(k+1)= y_e(k)\ |\ x_p(k)$and $y_p(k)\}=p.$

$P\{x_e(k+1)=PS(x_e(k))$ and $y_e(k+1)=NS(y_e(k))|\ x_p(k)$and $y_p(k)\}=(1-p)$

h. <u>Right bottom corner pursuer left condition</u>: If $x_e(k)= N$ and $x_p(k)=NS(x_e(k))$ and $y_p(k)= y_e(k)=0$ then ;

$P\{x_e(k+1)= x_e(k)$ and $y_e(k+1)=PS(\ y_e(k))\ |\ x_p(k)$and $y_p(k)\}=p.$

$P\{x_e(k+1)=NS(x_e(k))$ and $y_e(k+1)=PS(y_e(k))|\ x_p(k)$and $y_p(k)\}=(1-p).$

i. <u>Right bottom corner pursuer up condition</u>: If $x_e(k)= x_p(k)= N$ and $y_e(k)=0$ and $y_p(k)=PS(y_e(k))$ then ;

$P\{x_e(k+1)= NS(x_e(k))$ and $y_e(k+1)= y_e(k) \mid x_p(k)$and $y_p(k)\}=p$.

$P\{x_e(k+1)=NS(x_e(k))$ and $y_e(k+1)=PS(y_e(k))\mid x_p(k)$and $y_p(k)\}=(1-p)$.

j. <u>Other conditions</u>: If (a) to (k) are not satisfied and;

i. $x_p(k)= NS(x_e(k))$ & $y_p(k)=PS(y_e(k))$ then;

$P\{x_e(k+1)= PS(x_e(k))$ and $y_e(k+1)=NS(y_e(k) \mid x_p(k)$and $y_p(k)\}=p$.

$P\{x_e(k+1)= x_e(k)$ and $y_e(k+1)=NS(y_e(k))\mid x_p(k)$and $y_p(k)\}=(1-p)/2$.

$P\{x_e(k+1)=PS(x_e(k))$ and $y_e(k+1)=y_e(k)\mid x_p(k)$and $y_p(k)\}=(1-p)/2$.

ii. $x_p(k)= x_e(k)$ & $y_p(k)=PS(y_e(k))$ then;

$P\{x_e(k+1)= x_e(k)$ and $y_e(k+1)=NS(y_e(k) \mid x_p(k)$and $y_p(k)\}=p$.

$P\{x_e(k+1)= NS(x_e(k))$ and $y_e(k+1)=NS(y_e(k))\mid x_p(k)$and $y_p(k)\}=(1-p)/2$.

$P\{x_e(k+1)=PS(x_e(k))$ and $y_e(k+1)=NS(y_e(k))\mid x_p(k)$and $y_p(k)\}=(1-p)/2$.

iii. $x_p(k)= PS(x_e(k))$ & $y_p(k)=PS(y_e(k))$ then;

$P\{x_e(k+1)= NS(x_e(k))$ and $y_e(k+1)=NS(y_e(k) \mid x_p(k)$and $y_p(k)\}=p$.

$P\{x_e(k+1)= x_e(k)$ and $y_e(k+1)=NS(y_e(k))\mid x_p(k)$and $y_p(k)\}=(1-p)/2$.

$P\{x_e(k+1)=NS(x_e(k))$ and $y_e(k+1)=y_e(k)\mid x_p(k)$and $y_p(k)\}=(1-p)/2$.

iv. $x_p(k)= PS(x_e(k))$ & $y_p(k)=y_e(k)$ then;

$P\{x_e(k+1)= NS(x_e(k))$ and $y_e(k+1)= y_e(k) \mid x_p(k)$and $y_p(k)\}=p$.

$P\{x_e(k+1)=NS(x_e(k))$ and $y_e(k+1)=NS(y_e(k))\mid x_p(k)$and $y_p(k)\}=(1-p)/2$.

$P\{x_e(k+1)=NS(x_e(k))$ and $y_e(k+1)=PS(y_e(k))\mid x_p(k)$and $y_p(k)\}=(1-p)/2$.

v. $x_p(k)= PS(x_e(k))$ & $y_p(k)=NS(y_e(k))$ then;

$P\{x_e(k+1)= NS(x_e(k))$ and $y_e(k+1)=PS(y_e(k) \mid x_p(k)$and $y_p(k)\}=p$.

$P\{x_e(k+1)= x_e(k)$ and $y_e(k+1)=PS(y_e(k))\mid x_p(k)$and $y_p(k)\}=(1-p)/2$.

$P\{x_e(k+1)=NS(x_e(k))$ and $y_e(k+1)=y_e(k)\mid x_p(k)$and $y_p(k)\}=(1-p)/2$.

vi. $x_p(k)= x_e(k)$ & $y_p(k)=NS(y_e(k))$ then;

$P\{x_e(k+1)= x_e(k))$ and $y_e(k+1)=PS(y_e(k) \mid x_p(k)$and $y_p(k)\}=p$.

$P\{x_e(k+1)= NS(x_e(k))$ and $y_e(k+1)=PS(y_e(k))\mid x_p(k)$and $y_p(k)\}=(1-p)/2$.

$P\{x_e(k+1)=PS(x_e(k))$ and $y_e(k+1)=PS(y_e(k))\mid x_p(k)$and $y_p(k)\}=(1-p)/2$.

vii. $x_p(k)= NS(x_e(k))$ & $y_p(k)=NS(y_e(k))$ then;

$P\{x_e(k+1)= PS(x_e(k))$ and $y_e(k+1)=PS(y_e(k) \mid x_p(k)$and $y_p(k)\}=p$.

$P\{x_e(k+1)= x_e(k)$ and $y_e(k+1)=PS(y_e(k))|\, x_p(k)$and $y_p(k)\}=(1\text{-}p)/2$.

$P\{x_e(k+1)=PS(\, x_e(k))$ and $y_e(k+1)=y_e(k)|\, x_p(k)$and $y_p(k)\}=(1\text{-}p)/2$.

viii. $x_p(k)= x_e(k)$ & $y_p(k)=NS(y_e(k))$ then;

$P\{x_e(k+1)= PS(x_e(k))$ and $y_e(k+1)= y_e(k) \mid x_p(k)$and $y_p(k)\}=p$.

$P\{x_e(k+1)= PS(x_e(k))$ and $y_e(k+1)=NS(y_e(k))|\, x_p(k)$and $y_p(k)\}=(1\text{-}p)/2$.

$P\{x_e(k+1)=PS(\, x_e(k))$ and $y_e(k+1)=PS(y_e(k))|\, x_p(k)$and $y_p(k)\}=(1\text{-}p)/2$.

5.3. Properties of The Stochastic Digraph Associated with the Pursuer-Evader Problem:

After inspecting our model representation of the problem, it was found that it has the following characterizing properties;

a. The number of states of the system is finite.

b. The system is stationary, which means the probability distribution of transition between states, the instantaneous cost, and the system dynamics are independent of the states.

c. There are (N-3) final equilibrium states of the $N\times N$ stochastic pursuer-evader problem.

d. The cost value of the final equilibrium states is zero.

e. At any time instance t, $(x_e, y_e)\neq (x_p, y_p)$.

f. The di-graph representing the stochastic pursuer-evader model is *pseupo-stochastic*, which means the transition between states is stochastic when the evader is to move. Meanwhile, when the pursuer is to move, the transition between states is deterministic since the pursuer is allowed to go to certain locations based on the dynamics i.e. the probability of pursuer transitions is always 1.

g. The estimated cost associated with each edge is the pseudo-stochastic pursuer-evader di-graph is 1.

5.4. Problem Statement

Based on the previous investigations of the mathematical model of the problem at hand, its characterizing properties, and the proposed solution technique, we can formulate the problem as following:

Given a finite state space $S=\{1,2\ldots N,t\}$ with transition probability between states:

$p_{ij}(u)=P(s_{k+1}=j|\ s_k=i,\ u_k=u)$

Where $u_k=u\in U(i)$, the control set ,is finite at each state i.

• The instantaneous cost $c(i,u)$ of a state is incurred-when the control $u\in U(i)$ is selected-as the expected cost/stage as:

$$c(i,u)=\sum_{j=1}^{n}p_{ij}(u)\,\tilde{g}(i,u,j) \qquad (3)$$

where, $g^{\sim}(i,u,j)$ is the estimated cost to move from state i using control u to go to state j.

Then, the cost value of each state is given by:

$$J(i)=[c(i,u)+\sum_{j=1}^{n}p_{ij}(u)J(j)] \qquad i=1,2,\ldots\ldots,n \qquad (4)$$

Our objective is to find the set of optimal control policy U^* that gives minimum expected cost value for the pursuer to drive the evader to the pen, i.e.,

$$J^*(i)=\min_{u\in U}\ E\{g(i,u)+\sum_{j=1}^{n}p_{ij}(u)J(j)\} \qquad i=1,2,\ldots\ldots,n \qquad (5)$$

5.5. Methods of Solution

Solving the problem outlined above mainly depends on calculating the cost function values. Although it may look like solving a set of $(N\times N)^2$ [*]linear algebraic equations, it became obvious that this is not the case since the policy based on which the cost function values yet to be determined first. Once this policy is determined, then the problem may be considered as solving a system of linear equations and even with order less that

[*] 2 here because the number of players is 2, otherwise the power should be replaced by the number of players.

$(N\times N)^2$. Bearing this in mind, solving for the cost value function while searching for the optimum control policy involves 3 techniques. These techniques are:

a. *Apmissible Policy Search Technique*

Solving for $\mathbf{J}^*$ mainly depends on the characteristics of the probability state transition matrix (*PSTM*) **P**. Close examination of the *PSTM* showed that if the transition between states results from pursuer movements, the entities of the corresponding row for the current state are either 0, where there is no path to go to the corresponding state, or 1, which corresponds to the next state, the pursuer can derive the system to. Meanwhile, if the transition results from evader's movement, the entities of the *PSTM* is either 0, or the probability of the system to go the corresponding state due to evader movement. Therefore, the minimization process in equation (5) has one argument when carried to calculate the corresponding cost of evader movement and results in picking one of the states that results in minimum cost when the pursuer moves. In other words we can put equation 5 in the matrix form as:

$$\mathbf{J}^* = (\mathbf{I} - \tilde{\mathbf{P}})^{-1}\mathbf{SC} \tag{6}$$

where;

S… is the $N\times N$ state transition matrix.

C… is the $N\times 1$ instantaneous cost matrix.

$\tilde{\mathbf{P}}$ …is the $N\times N$ modified probability state transition matrix, obtained from P by picking the only 1 that corresponds to the state that results in the minimum cost value and replacing all the other 1's by zeros. So the problem now is how to find $\tilde{\mathbf{P}}$?

This can be accomplished by searching all the possible combinations of 1's in the *PSTM* till we get the pattern that results in the minimum value of **J.**

Despite its accuracy in calculating the values of the cost function, this technique has proven to be very time consuming since it searches all the space of admissible control policies.

b. *Value Iteration Technique*

Assuming that there is an integer number of stages, *m*, represent the maximum number of stages till the final equilibrium state is reached, then there is a positive probability that the final equilibrium state will be reached in less than *m* stages [16]. Based on this assumption, an iterative technique can be used to calculate the cost value of each state beginning form any initial values $J_0(1)$……… $J_0(n)$. So, applying the DP algorithm;

$$J_{k+1}(i) = \min_{u \in U(i)} \left[c(i,u) + \sum_{j=1}^{n} p_{ij}(u) J_k(j) \right] \qquad i = 1,2,\ldots\ldots n \qquad (7)$$

The sequence J_{k+1} (i) will converge to the optimal cost, $J^*_{k+1}(i)$ given by Bellman's equation, after finite number of iterations.

c. *Policy Iteration Technique*

This technique depends on searching the admissible policy subspace in a steepest decent way, beginning form any admissible policy. It has three phases:

i. Initialization step: Start with one of the admissible policies, u^0.

ii. Policy evaluation step: Solve the linear system of equations given by eq. (6) to get the cost values for this policy, $J_{u^0}(i),\ i = 1,2,\ldots n$.

iii. Policy improvement step: in which we compute a new policy u^{k+1} which minimizes the expected cost calculated in step (ii), i.e.

$$u^{k+1}(i) = \arg \min_{u \in U(i)} \left[c(i,u) + \sum_{j=1}^{n} p_{ij}(u) J_{u^k}(j) \right], \qquad i = 1,2,\ldots n \qquad (8)$$

iv. If $J_{u^{k+1}}(i) = J_{u^k}(i),\ i = 1,2,\ldots n$, terminates or set $J_{u^{k+1}}(i) = J_{u^0}(i)$ and go to step (i).

5.6. Simulation Results

Simulating our system begins by computing the value of the cost function of each state using one of the techniques mentioned above. Table 1, shows the values of the cost function for a 3×3 grid, and interstate transition probability of 0.8 using value iteration, V_v, and policy iteration, V_p, techniques. As shown from the results that both techniques converge to exactly the same cost function value. The state number corresponds to every possible combination of the x and y coordinates of the pursuer and evader positions. The evader movements are controlled by a random number generator where it can move randomly according to the dynamics defined in part II. The pursuer makes its transitions based on the cost function value of the adjacent states to the current state of the system. The pursuer moves to the state of the lowest cost of all adjacent states, then it checks whether the system is at equilibrium or not to make its next move. This process continues till the system reaches one of the final equilibrium states defined in part II.

A graphical user interface is used to simulate the system where the user is to choose the grid size, *N*, from a drop box. The intial position of the pursuer and the is supplied to the simulation programm using an edit box. The probability, *p*, of transition is set using a slider. Finally, the technique used to calculate the cost of each state is choosen using a check box, then simulation starts by pressing the *START* button.

Figures 5.6, and 5.7 shows the used graphical user interface provided to simulate the system where figure 5.6 shows a single run with intial position of the pursuer is at (1,1) and the intial position of the evader is at (4,4) and the probability of transition is 0.8 where the cost values is calculated using value iteration technique. Meanwhile, figure 5.7 illustrates the same the same intial condition and the same probability of transition system , but the cost value is calculated this time using policy iteration technique. It should be noticed that the differences between the two systems, although having the same intial states, comes from the stochastic motion of the evader. Also, it can be noticed that the pursuer's movements are the same from the intial state till the state where the evader makes its first move. This is due to the determinstic nature of the pursuer's motion.

S	V_v	V_p	S	V_v	V_p	S	V_v	V_p	S	V_v	V_p	S	V_v	V_p	S	V_v	V_p	S	V_v	V_p
1	9.00	9.00	13	6.02	6.02	25	4.88	4.88	37	0.00	0.00	49	2.50	2.50	61	9.00	9.00	73	0.00	0.00
2	6.22	6.22	14	6.61	6.61	26	5.60	5.60	38	2.50	2.50	50	3.39	3.39	62	7.21	7.21	74	2.50	2.50
3	4.88	4.88	15	7.21	7.21	27	6.36	6.36	39	3.88	3.88	51	9.00	9.00	63	6.36	6.36	75	3.88	3.88
4	6.22	6.22	16	4.88	4.88	28	7.14	7.14	40	2.50	2.50	52	3.88	3.88	64	0.00	0.00	76	2.50	2.50
5	8.21	8.21	17	5.60	5.60	29	6.02	6.02	41	9.00	9.00	53	4.60	4.60	65	2.50	2.50	77	1.50	1.50
6	6.60	6.60	18	6.36	6.36	30	4.88	4.88	42	5.60	5.60	54	5.36	5.36	66	3.88	3.88	78	4.60	4.60
7	4.88	4.88	19	0.00	0.00	31	9.00	9.00	43	3.88	3.88	55	0.00	0.00	67	1.50	1.50	79	3.88	3.88
8	6.60	6.60	20	1.70	1.70	32	6.61	6.61	44	5.60	5.60	56	3.50	3.50	68	3.39	3.39	80	4.60	4.60
9	7.36	7.36	21	9.00	9.00	33	5.60	5.60	45	6.36	6.36	57	4.88	4.88	69	4.60	4.60	81	9.00	9.00
10	7.14	7.14	22	3.50	3.50	34	8.09	8.09	46	0.00	0.00	58	1.70	1.70	70	2.88	2.88			
11	9.00	9.00	23	5.81	5.81	35	7.21	7.21	47	1.50	1.50	59	5.81	5.81	71	9.00	9.00			
12	8.09	8.09	24	7.21	7.21	36	6.36	6.36	48	2.88	2.88	60	5.60	5.60	72	5.36	5.36			

Table 1. The cost function values for a 3×3 grid, and interstate transition probabilty 0.8, using value iteration technique, V_{vitr}, and policy iteration technique, V_{pitr}.

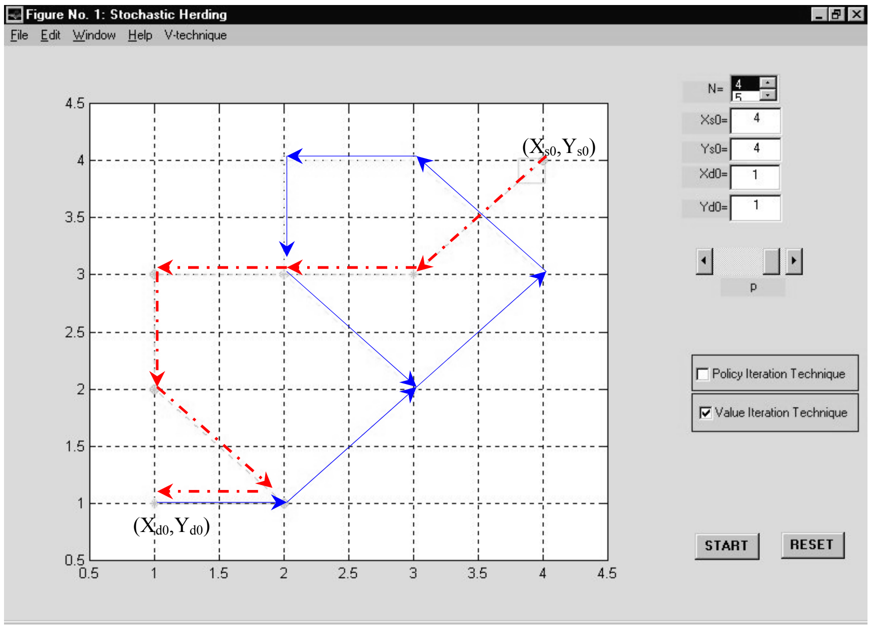


Figure No. 1: Stochastic Herding
File Edit Window Help V-technique
(Xs0,Ys0)
(Xd0,Yd0)
N= 4 5
Xs0= 4
Ys0= 4
Xd0= 1
Yd0= 1
p
Policy Iteration Technique
Value Iteration Technique
START
RESET

Settings: N= 4; Xs0= 4; Ys0= 4; Xd0= 1; Yd0= 1; [✓ Value Iteration Technique]

Fig. 5.6 Graphical user interface model for the herding problem with intial position of the evader =(4,4), *p*=0.8, and cost values calculated by value iteration technique

.

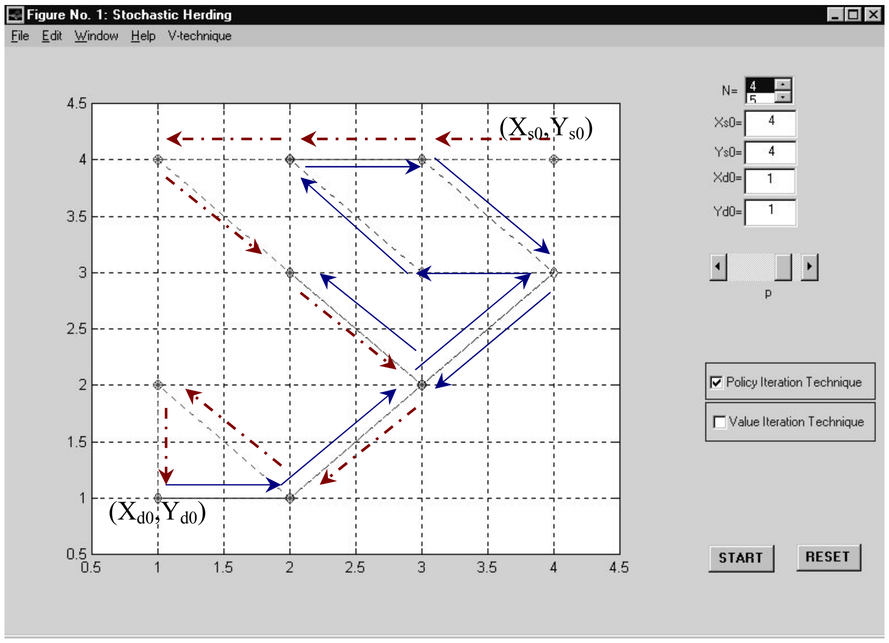


Figure No. 1: Stochastic Herding
File Edit Window Help V-technique
(Xs0,Ys0)
(Xd0,Yd0)
N= 4 5
Xs0= 4
Ys0= 4
Xd0= 1
Yd0= 1
p
Policy Iteration Technique
Value Iteration Technique
START
RESET

Settings: N= 4; Xs0= 4; Ys0= 4; Xd0= 1; Yd0= 1; [✓ Policy Iteration Technique]

Fig. 5.7. Graphical user interface model for the herding problem with intial position of the evader =(4,4), *p*=0.8, and cost values calculated by policy iteration technique

5.7. Summary

In this chapter we provided a stochastic model of the herding problem in the context of the stochastic dynamic programming. The stochastic dynamics of the model were introduced in the beginning of the chapter. Then, analysis of the pseudo-stochastic di-graph representing the system is given. After that, three solution algorithms are introduced and analyzed. Finally, samples of the simulation results for the proposed solution algorithms are introduced to conclude the chapter.

Chapter (6)

Pursuit Evasion: The Herding Non-cooperative Dynamic Game

6.1. Introduction

In this final chapter we present a dynamic programming based solution to the pursuer evader herding problem, where the pursuer and the evader are playing a non-cooperative deterministic game The evader here is not assumed to be a passive evader, rather it is an active evader so that it tries to maximize the cost value of the current state. As followed in the previous chapters, we present the dynamics of the problem then provide the dynamic programming solution to the problem. The solution is proven to be correct and then simulations are performed to illustrate some example runs.

6.2. **A *N*×*N* Grid Pursuer-Evader Herding Problem**

We consider the pursuer-evader problem in a *N*×*N* grid as shown in figure 6.1. The pursuer can occupy one of the *N*×*N* positions and so can the evader. Therefore, there are N^2 states in the system. The aim of the pursuer is to make the evader go to the pen that is the (0,0) position for the evader. The following shows the nomenclature used in this paper.

$x_p(k)$	x coordinate of the pursuer position at time instance *k*
$y_p(k)$	y coordinate of the pursuer position at time instance *k*
$x_e(k)$	x coordinate of the evader position at time instance *k*
$y_e(k)$	y coordinate of the evader position at time instance *k*
x(*k*)	state vector given by $\mathbf{x}(k) = [x_e(k)\ y_e(k)\ x_p(k)\ y_p(k)]$ at time instance *k*

For the *N*x*N* pursuer evader problem, we have $\forall k \quad x_p(k) \in \{0,1,2,....,N\}$, $y_p(k) \in \{0,1,2\}....,N$, $\forall k \quad x_e(k) \in \{0,1,2,....,N\}$ and $y_e(k) \in \{0,1,2\}....,N$. However, the pursuer and the evader can not have the same location on the grid as their initial positions. It can be proven that if they have different initial positions, then based on the allowable actions of both (as described later), they can never end up on the same location. There is a cost of one unit for each step (horizontal or vertical or diagonal) of a pursuer as well as of a evader. The aim of the pursuer is to move the evader to the pen i.e. to the (0,0) coordinate, with the least cost. Fig. 6.1 below shows the $N \times N$ grid for the pursuer-evader problem.

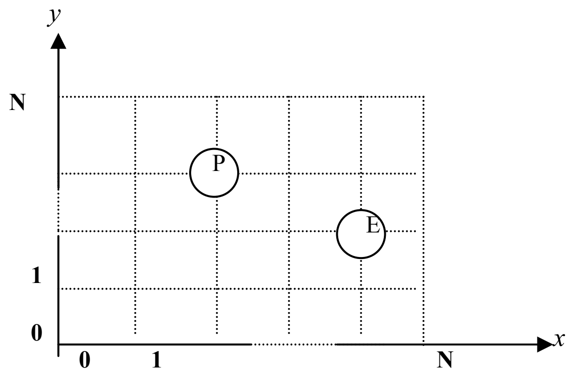


Fig. 6.1. The $N \times N$ pursuer-evader problem grid

Definition* 1;Positive successor function:* Positive successor function is given by:

$$PS(X(k)) = \begin{cases} X(k) & \text{if } X(k) = N \\ X(k)+1 & \text{if } 0 \le X(k) \le N \end{cases} \tag{6.1}$$

where,
$x(k)$ is the x or y coordinate of either the pursuer or the evader.
Thus, $PS(.)$: $X=\{0,1,2,\ldots N\} \rightarrow Y=\{1,2,\ldots N\}$

Definition* 2; Negative successor function:* Negative successor function is given by:

$$NS(X(k)) = \begin{cases} X(k) & \text{if } X(k) = 0 \\ X(k)-1 & \text{if } 0 \le X(k) \le N \end{cases} \tag{6.2}$$

$x(k)$ is the x or y coordinate of either the pursuer or the evader.
Thus, $NS(.)$: $X=\{0,1,2,\ldots N\} \rightarrow Y=\{0,1,2,\ldots N-1\}$

Definition* 3; Equilibrium state of the evader:* The evader is in equilibrium state when given a time instant T, if one of the following conditions is satisfied;

a. Far condition: $x_p(T)$-$x_e(k)$ and/or $y_p(T)$-$y_e(k)$>1.
b. Left boundary condition: $x_e(k)=0$, $0<y_e(k)<N$ with $y_p(k)= y_e(k)$, and $x_p(k)=PS(x_e(k))$.

c. Right boundary condition: $x_e(k)=N$, $0<y_e(k)<N$ with $y_p(k)= y_e(k)$, and $x_p(k)=NS(x_e(k))$.

d. Low boundary condition: $y_e(k)=0$, $0<x_e(k)<N$ with $x_p(k)= x_e(k)$, and $y_p(k)=PS(y_e(k))$.

e. Upper boundary condition: $y_e(k)=N$, $0<x_e(k)<N$ with $x_p(k)= x_e(k)$, and $y_p(k)=NS(y_e(k))$.

f. Upper left corner condition: $(x_e(k), y_e(k))=(0,N)$, and $x_p(k)=PS(x_e(k))$, and $y_p(k)=NS(y_e(k))$.

g. Upper right corner condition $(x_e(k), y_e(k))=(N,N)$, and $x_p(k)=NS(x_e(k))$, and $y_p(k)=NS(y_e(k))$.

h. Low right corner condition $(x_e(k), y_e(k))=(N,0)$, and $x_p(k)=NS(x_e(k))$, and $y_p(k)=PS(y_e(k))$.

***Definition* 4**; *Final equilibrium state of the evader:* The evader is in final equilibrium state at time instant T, when the following condition is satisfied:
$x_p(k)=x_p(T)$ and $y_p(k)=y_p(T)$, then $x_e(k)=0$ and $y_e(k)=0$. $\forall k>T$.

The following rules generate the pursuer-controlling movements and assign probabilities to the evader transitions based on its relative location with respect to the pursuer:

a. $\forall k\ x_p(k), y_p(k), x_e(k)$, and $y_e(k) \in \{0,1,2,\ldots\ldots N\}$.

b. The pursuer moves when the evader is at an equilibrium state only.

c. The pursuer can move one step at a time, depending on its position in the grid, and its relative location with respect to evader position as illustrated in Fig.6.2 and 6.3 below:

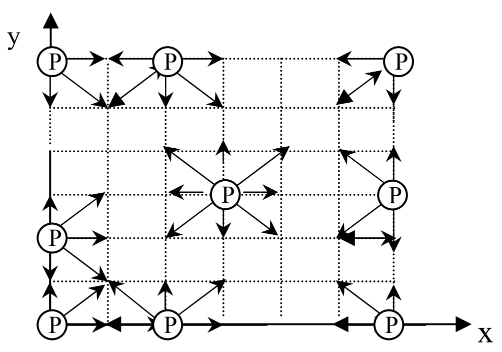


Fig. 6.2. Pursuer movements with distance between pursuer and evader>2.

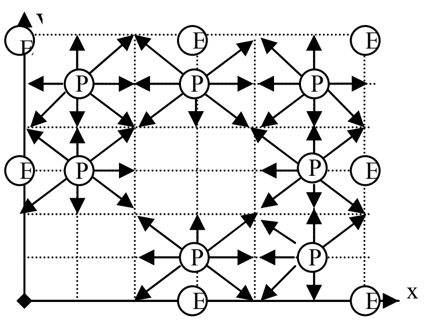


Fig.6.3. Pursuer movements with evader at equilibrium state.

The evader transition depends on its relative position to the pursuer as following:

a. <u>Far Condition</u>: If $x_e(k)<NS(x_p(k))$ or $x_e(k)>PS(x_p(k))$ or $y_e(k)<NS(y_p(k))$ or $y_e(k)>PS(y_p(k))$, then $x_e(k+1)=x_p(k)$ and $y_e(k+1)=y_p(k)$.

b. <u>Left top corner pursuer right condition</u> : If $x_e(k)=0$ and $x_p(k)=PS(x_e(k))$ and $y_e(k)=y_p(k)=N$ then ;

$$(x_e(k+1)\ ,\ y_e(k+1))\ |(\ x_p(k),\ y_p(k)) = \arg\max_{v\in V_j} \{(\ x_e(k),\ NS(y_e(k))),\ (PS(x_e(k)),\ NS(y_e(k)))\}$$

c. <u>Left top corner pursuer down condition</u>: If $x_e(k)=\ x_p(k)=0$ and $y_p(k)=NS(y_e(k))$ and $y_e(k)=\ N$ then ;

$$(x_e(k+1)\ ,\ y_e(k+1))\ |(\ x_p(k),\ y_p(k)) = \arg\max_{v\in V_j} \{(\mathrm{PS}(\ x_e(k)),y_e(k)),\ (NS(x_e(k)),\ NS(y_e(k)))\}$$

d. <u>Right top corner pursuer left condition:</u> If $x_e(k)=N$ and $x_p(k)=NS(x_e(k))$ and $y_e(k)=y_p(k)=N$ then ;

$$(x_e(k+1)\ ,\ y_e(k+1))\ |(\ x_p(k),\ y_p(k)) = \arg\max_{v\in V_j} \{(\ x_e(k),\ NS(y_e(k))),\ (NS(x_e(k)),\ NS(y_e(k)))\}$$

e. <u>Right top corner pursuer down condition:</u> If $x_e(k)=\ x_p(k)=N$ and $y_p(k)=NS(y_e(k))$ then ;

$$(x_e(k+1)\ ,\ y_e(k+1))\ |(\ x_p(k),\ y_p(k)) = \arg\max_{v\in V_j} \{(NS(\ x_e(k)),\ (y_e(k))),\ NS(x_e(k)),$$

$$NS(y_e(k)))\}$$

f. <u>Left bottom corner pursuer right condition:</u> If $x_e(k)=\ 0$ and $x_p(k)=PS(x_e(k)\ )$ and $y_p(k)=y_e(k)=0$ then ;

$$(x_e(k+1)\ ,\ y_e(k+1))\ |(\ x_p(k),\ y_p(k)) = \arg\max_{v\in V_j} \{(\ x_e(k),\ PS(y_e(k))),\ (PS(x_e(k)),\ PS(y_e(k)))\}$$

g. <u>Left bottom corner pursuer up condition:</u> If $x_e(k)=\ x_p(k)=\ 0$ and $y_p(k)=PS(y_e(k)\ )$ then ;

$$(x_e(k+1)\ ,\ y_e(k+1))\ |(\ x_p(k),\ y_p(k)) = \arg\max_{v\in V_j} \{(PS(\ x_e(k)),\ y_e(k)),\ (PS(x_e(k)),\ NS(y_e(k)))\}$$

h. <u>Right bottom corner pursuer left condition</u>: If $x_e(k)=\ N$ and $x_p(k)=NS(x_e(k))$ and $y_p(k)=\ y_e(k)=0$ then ;

$$(x_e(k+1)\,,y_e(k+1))\,|(\,x_p(k),y_p(k)) = \arg\max_{v\in V_j}\{(\,x_e(k), PS(y_e(k))), (NS(x_e(k)), PS(y_e(k)))\}$$

i. <u>Right bottom corner pursuer up condition</u>: If $x_e(k)= x_p(k)= N$ and $y_e(k)$=0 and $y_p(k)=PS(y_e(k))$ then ;

$$(x_e(k+1)\,,y_e(k+1))\,|(\,x_p(k),y_p(k)) = \arg\max_{v\in V_j}\{(\,NS(x_e(k)), y_e(k)), (NS(x_e(k)), PS(y_e(k)))\}$$

j. <u>Other conditions</u>: If (a) to (i) are not satisfied and;

   i. $x_p(k)= NS(x_e(k))$ & $y_p(k)=PS(y_e(k))$ then;

   $$(x_e(k+1)\,,y_e(k+1))\,|(\,x_p(k),y_p(k)) = \arg\max_{v\in V_j}\{(PS(\,x_e(k), NS(y_e(k))),$$

   $$(x_e(k),NS(y_e(k)), (PS(\,x_e(k)), y_e(k))\}$$

   ii. $xd(k)= xs(k)$ & $yd(k)=PS(ys(k))$ then;

   $$(xs(k+1)\,,ys(k+1))\,|(xd(k),yd(k)) = \arg\max_{v\in V_j}\{(PS(\,x_e(k), NS(y_e(k))),$$

   $$(x_e(k),NS(y_e(k)), (PS(\,x_e(k)), y_e(k))\}$$

   iii. $x_p(k)= PS(x_e(k))$ & $y_p(k)=PS(y_e(k))$ then;

   $$(xs(k+1)\,,ys(k+1))\,|(xd(k),yd(k)) = \arg\max_{v\in V_j}\{(NS(\,x_e(k), NS(y_e(k))),$$

   $$(x_e(k),NS(y_e(k)), (NS(\,x_e(k)), y_e(k))\}$$

   iv. $x_p(k)= PS(x_e(k))$ & $y_p(k)=y_e(k)$ then;

   $$(xs(k+1)\,,ys(k+1))\,|(xd(k),yd(k)) = \arg\max_{v\in V_j}\{(NS(\,x_e(k),(y_e(k)),$$

   $$(NS(x_e(k)),NS(y_e(k)), (NS(\,x_e(k)), (PSy_e(k)))\}$$

   v. $x_p(k)= PS(x_e(k))$ & $y_p(k)=NS(y_e(k))$ then;

   $$(xs(k+1)\,,ys(k+1))\,|(xd(k),yd(k)) = \arg\max_{v\in V_j}\{(NS(\,x_e(k), PS(y_e(k))),$$

   $$(x_e(k),(PS(y_e(k))), (NS(\,x_e(k)), y_e(k))\}$$

   vi. $x_p(k)= x_e(k)$ & $y_p(k)=NS(y_e(k))$ then;

   $$(xs(k+1)\,,ys(k+1))\,|(xd(k),yd(k)) = \arg\max_{v\in V_j}\{(\,x_e(k), PS(y_e(k))),$$

   $$(NS(x_e(k)),PS(y_e(k)), (PS(\,x_e(k)), (PS(y_e(k))\}$$

vii. $x_p(k) = NS(x_e(k))$ & $y_p(k) = NS(y_e(k))$ then;

$$(xs(k+1), ys(k+1)) \,|(xd(k), yd(k)) = \arg\max_{v \in V_j} \{(PS(x_e(k), PS(y_e(k))),$$
$$(x_e(k), PS(y_e(k)), (PS(x_e(k)), y_e(k))\}$$

viii. $x_p(k) = x_e(k)$ & $y_p(k) = NS(y_e(k))$ then;

$$(xs(k+1), ys(k+1)) \,|(xd(k), yd(k)) = \arg\max_{v \in V_j} \{(PS(x_e(k), (y_e(k)),$$
$$(PS(x_e(k)), NS(y_e(k)), (PS(x_e(k)), PS(y_e(k))\}$$

Theorem* 1:* There are ($N\times N$-3) final equilibrium states of the $N\times N$ pursuer-evader problem.

Proof: Figure 6.4 shows the six final equilibrium states of a 3×3 grid. The figure implies that the pursuer can be in any of the six positions to obtain a final equilibrium state. We can prove that the state when the pursuer is in (1,1) position is a final equilibrium state because of the 4(j) rule of the dynamics. For the other ones we can prove the same by using 4(a) rule.

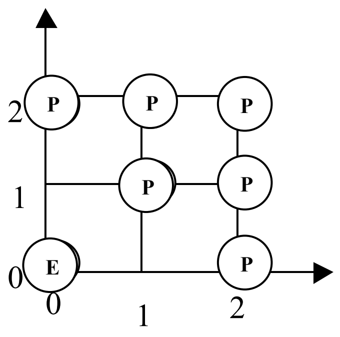


Fig. 6.4 The six final equilibrium states for a 3×3 grid.

For each given equilibrium-state, the pursuer is free to choose its next move based on a finite set of possible actions. This finite set is a function of the state **x**. If the state is a non-equilibrium state then the pursuer is not allowed to move in that time instant. The evader will move from the non-equilibrium-state to another state that could be (in general) another non-equilibrium-state or an equilibrium-state.

By induction, the same result can be proved for any *N*×*N* grid.

Theorem* 2*: If for any positive integer k, $x_e(k) \neq x_p(k)$ and $y_e(k) \neq y_p(k)$ then for any $t \geq k$, the following two statements can not be simultaneously false: $x_e(t) \neq x_p(t)$ and $y_e(t) \neq y_p(t)$.

Proof: This can be easily proven by noting that in order for both statements to be false at time t, the system would have to be in equilibrium condition, and then the pursuer would have to move to acquire the same coordinates as those of the evader. However, due to the constraint on the motion of the pursuer, that the pursuer can only move when the system is in a non-equilibrium state, the theorem is proven.

Let *U* be the discrete set of actions available to the pursuer (pursuer) when the system is in the state **x**. The pursuer (pursuer) defines a *policy* $\mu : \mathbf{x} \to U$ that is a function from the state to pursuer actions. This defines a feedback control policy for the pursuer. Let *W* be the discrete set of actions available to the evader (evader) when the system is in the state **x**. The evader (evader) defines a *policy* $\omega : \mathbf{x} \to W$ that is a function from the state to evader actions. This defines a feedback control policy for the evader. We also define a *value function* $V_{\mu\omega}(\mathbf{x})$, which is the sum of all future instantaneous costs given that the initial state of the system is **x** and the system follows the policies μ and ω. We define *instantaneous cost* as:

$$c_{\mathbf{x}}(u,w) = \min(|x_d(k) - x_d(k-1)| + |y_d(k) - y_d(k-1)| + \\ |x_s(k) - x_s(k-1)| + |y_s(k) - y_s(k-1)|, 1)$$

The value function $V_{\mu\omega}(\mathbf{x})$ is given by

$$V_{\mu\omega}(i) = \sum_{k=0}^{\infty} c_{\mathbf{x}(k)}(\mu(\mathbf{x}(k))) \quad where \quad \mathbf{x}(0) = i$$

Problem Statement* 1*:

(a) Find the *policies* μ_* and ω_*. that produce the following lower value function:

$$V_*(i) = \min_{\mu} \max_{\omega} V_{\mu\omega}(i)$$

(b) Find the *policies* μ^* and ω^*. that produce the following upper value function:

$$V^*(i) = \max_{\omega} \min_{\mu} V_{\mu\omega}(i)$$

6.3. Properties of the Diagraph Associated with the Pursuer-Evader, Herding Problem

We can represent the pursuer-evader problem described above as a digraph ***G*** = (***V, E***) that consists of a finite set ***V*** of vertices or nodes representing the states of the system, and a finite set *E* of edges. ***V*** consists of all the possible values of the state **x**. The cardinality of ***V*** denoted by *N*(***V***) is N×N. There exists an edge e from a state-value (node) *v* to *w* if for some *k*, *v* = **x**(k) and *w* = **x**(k+1), following the dynamics generated by the rules in section 2. The digraph is a directed network or a weighted-digraph since we associate a cost with each edge using the instantaneous cost formula from section 2. The digraph is also simple, since there are no loops or multiple edges. The adjacency matrix of the digraph is an $N(V)\times N(V)$ matrix whose diagonal elements are all zeros. The reader is referred to [51] for background reference.

***Theorem* 3:** The instantaneous cost associated with each edge in the digraph of the pursuer-evader problem is 1.

Proof: The proof of the theorem comes directly from calculating the cost for the motion of the pursuer and the evader, according to their dynamics as given in section 2.

a) Far condition;

In this case, and by definition 1, the evader is at equilibrium state, so, only the pursuer is allowed to move;

$x_e(k+1) = x_e(k)$ & $y_e(k+1) = y_e(k)$

Since only the pursuer is allowed to move, and for only one step. Then, the minimum distance the pursuer can move, will result from moving it one step in either the x direction or the y direction;

$$x_p(k+1) - x_p(k) = 1 \quad or \quad y_p(k+1) - y_p(k) = 1$$

Substitute with these in the cost equation$\Rightarrow$

$$c_x(\mu) = \min(|x_p(k+1) - x_p(k)| + |y_p(k+1) - y_p(k)| + \\ |x_e(k+1) - x_e(k)| + |y_e(k+1) - y_e(k)|, 1)$$

$$c_x(\mu) = \min(1+0+0+0,1) = 1 \quad or \quad c_x(\mu) = \min(0+1+0+0,1) = 1$$

b) <u>Left top corner, pursuer right</u>

In this case, the evader is not in an equilibrium state and therefore, only the evader is moving while not the pursuer;

$$x_e(k) = 0 \ and \ y_e(k) = N; x_e(k+1) = 0 \ and \ y_e(k+1) = NS(y_e(k)) = NS(N) = N-1$$
$$x_p(k) = PS(x_e(k)) = PS(0) = 1 \ and \ y_p(k) = N$$
$$x_p(k+1) = x_p(k) = 1 \ and \ y_p(k+1) = y_p(k) = N$$

$$\therefore c_x(\mu) = \min(0+0+0+1,1) = 1$$

c) <u>Left top corner, pursuer down condition</u>

$$x_e(k) = 0 \ and \ y_e(k) = N; x_p(k) = 0 \ and \ y_p(k) = NS(N) = N-1$$
$$x_e(k+1) = PS(x_e(k)) = 1 \ and \ y_e(k+1) = y_e(k) = N$$
$$x_p(k+1) = x_p(k) = 0 \ and \ y_p(k+1) = y_p(k) = N-1$$

$$\therefore c_x(\mu) = \min(0+0+1+0,1) = 1$$

d) <u>Right top corner, pursuer left condition</u>

$$x_e(k) = 2 \ and \ y_e(k) = 2; x_p(k) = NS(x_e(k)) = 1 \ and \ y_p(k) = 2$$
$$x_e(k+1) = x_e(k) = 2 \ and \ y_e(k+1) = NS(y_e(k)) = 1$$
$$x_p(k+1) = x_p(k) = 1 \ and \ y_p(k+1) = y_p(k) = 2$$

$$\therefore c_x(\mu) = \min(0+0+1+0,1) = 1$$

e) Right top corner, pursuer down condition

$$x_e(k) = N \text{ and } y_e(k) = N; x_p(k) = x_e(k) = N \text{ and } y_p(k) = NS(y_e(k)) = N-1$$
$$x_e(k+1) = NS(x_e(k)) = N-1 \text{ and } y_e(k+1) = y_e(k) = N$$
$$x_p(k+1) = x_p(k) = N \text{ and } y_p(k+1) = y_p(k) = N-1$$

$\therefore c_x(\mu) = \min(0+0+1+0,1) = 1$

f) Left bottom corner, pursuer right condition

$$x_e(k) = 0 \text{ and } y_e(k) = 0; x_p(k) = PS(x_e(k)) = 1 \text{ and } y_p(k) = 0$$
$$x_e(k+1) = x_e(k) = 0 \text{ and } y_e(k+1) = PS(y_e(k)) = 1$$
$$x_p(k+1) = x_p(k) = 1 \text{ and } y_p(k+1) = y_p(k) = 0$$

$\therefore c_x(\mu) = \min(0+0+0+1,1) = 1$

g) Left bottom corner, pursuer up condition

$$x_e(k) = 0 \text{ and } y_e(k) = 0; x_p(k) = x_e(k) = 0 \text{ and } y_p(k) = PS(y_e(k)) = 1$$
$$x_e(k+1) = PS(x_e(k)) = 1 \text{ and } y_e(k+1) = y_e(k) = 0$$
$$x_p(k+1) = x_p(k) = 0 \text{ and } y_p(k+1) = y_p(k) = 1$$

$\therefore c_x(\mu) = \min(0+0+1+0,1) = 1$

h) Right bottom corner, pursuer left condition

$$x_e(k) = N \text{ and } y_e(k) = 0; x_p(k) = NS(x_e(k)) = N-1 \text{ and } y_p(k) = 0$$
$$x_e(k+1) = x_e(k) = N \text{ and } y_e(k+1) = PS(y_e(k)) = 1$$
$$x_p(k+1) = x_p(k) = N-1 \text{ and } y_p(k+1) = y_p(k) = 0$$

$\therefore c_x(\mu) = \min(0+0+0+1,1) = 1$

i) Right bottom corner, pursuer up condition

$$x_e(k) = N \text{ and } y_e(k) = 0; x_p(k) = x_e(k) = N \text{ and } y_p(k) = PS(y_e(k)) = 1$$
$$x_e(k+1) = NS(x_e(k)) = N-1 \text{ and } y_e(k+1) = y_e(k) = 0$$
$$x_p(k+1) = x_p(k) = N \text{ and } y_p(k+1) = y_p(k) = 1$$

$\therefore c_x(\mu) = \min(0+0+1+0,1) = 1$

Other conditions:

For all the cases mentioned in rule 4-j, the evader takes only one step at a time away from the pursuer which is not allowed to move since the evader in not in an equilibrium state. Therefore, in all cases we have;

$x_e(k+1) - x_e(k) = 1 \text{ or } y_e(k+1) - y_e(k) = 0 \Rightarrow$

$\therefore c_x(\mu) = \min(0+0+1+0,1) = 1$

or $x_e(k+1) - x_e(k) = 0$ *or* $y_e(k+1) - y_e(k) = 1 \Rightarrow$

$\therefore c_x(\mu) = \min(0+0+0+1,1) = 1$

or $x_e(k+1) - x_e(k) = 1$ *or* $y_e(k+1) - y_e(k) = 1 \Rightarrow$

$\therefore c_x(\mu) = \min(0+0+1+1,1) = 1$

Theorem* 4:* The digraph of the pursuer-evader problem is not a strongly connected, but is weakly connected.

Proof: It can be shown that starting from any allowable (all states except the ones with co-incident positions for pursuer and evader) state, one of the final equilibrium states can be reached. All the final equilibrium states have paths connecting them together. To see this, consider a final equilibrium state, and then move the pursuer back (to increase he distance between the pursuer and the evader). This action will not result in any evader movement. Then we can move the pursuer in positions that have distance more than one from the evader (at the pen). Then the pursuer can be moved to a different position corresponding to another final equilibrium position. This shows that starting from any initial allowable state, there is a path to all the final equilibrium states. This proves that the underlying graph of the digraph is connected. To show that it is not strongly connected, consider any final equilibrium state. From these states, there is no pursuer action that can take the evader from the boundary of the two dimensional space into the interior.

Some additional properties of the pursuer-evader digraph are given below:

1. The number of nodes that are adjacent from a node representing an equilibrium state depends on the location of the pursuer position in the grid. There are the following three possibilities on the number of adjacent states.
   a) There are eight states adjacent from the equilibrium state node if the pursuer position is in the interior. Only seven out of the eight are allowed since pursuer and evader are not allowed to have the same location.

b) There are five states adjacent from the equilibrium state node if the pursuer position is in the side but not in a corner. This can become four if one of the five is taken by the evader.

c) There are three states adjacent from the equilibrium state node if the pursuer position is in the corner. This can become two if one of the five is taken by the evader.

2. The number of nodes that are adjacent from a node representing a non-equilibrium state is either one or two. The state adjacent from the non-equilibrium node can be another non-equilibrium node or an equilibrium node. An example is shown in Fig.6.5.

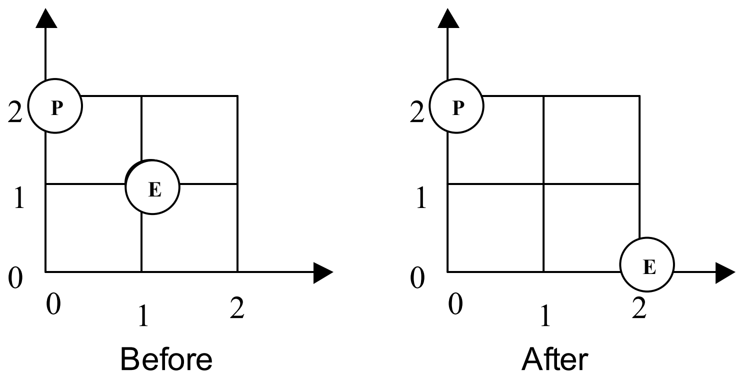


Fig. 6.5 Adjacent states for non-equilibrium initial state

***Theorem* 5** *(<u>Necessary and sufficient condition for existence of value</u>):* The necessary and sufficient condition for the existence of the value of the pursuit-evasion game (starting at any state) as described above is that there should be no cycles completable by the maximizing player (evader) in the digraph of the game.

Proof: If a cycle is completable by the pursuer, obviously, it will not complete the cycle during a game, since going into a cycle would increase the cost. On the other hand, if the evader can complete a cycle, it definitely will, since that will increase the cost and in fact make the cost to go to infinity. The reason for this is that, the control actions (of the pursuer as well as the evader) are constrained to be functions of the current state alone. Therefore, whatever action is taken by the evader at the state, where the cycle can be completed, that action will be repeated every time that same state is reached. Following the same logic, the states that were visited during the first traversal of the cycle will be repeated infinitely many times. Therefore, the total cost in that case will be infinite, and therefore, no value will exist for the game for all the states. It is a necessary condition, because since if cycles are present there are states for which there are no values. It is also a sufficient condition, because the total number of states in the system is infinite.

Theorem* 6*: The lower and upper values of the game are the same.
Proof: This is true because the set of all the states can be decoupled into two subsets, one subset containing the states that can be followed by pursuer action, and the other subset containing the states that can be followed by evader action. Therefore, for any initial state, the sequence of the pursuer and the evader actions is fixed. Therefore, the lower and upper values are the same, since changing the min operation before max or vice versa does not affect the value.

6.4. Topology of the Game and the Associated Properties

Let X be the set of all the states of the system. We define a real-valued function d on $X \times X$ for all the ordered pairs of X [52]. The function d is an anti-symmetric metric or a distance function and satisfies the following axioms. For every $a,b,c \in X$

axiom 1) $d(a,b) \geq 0$ and $d(a,a)=0$

axiom 2) $d(a,c) = d(a,b) + d(b,c)$

For the pursuer-evader game we are studying, we define a cost distance function $\delta(a,b)$ to be the cost that takes the game to evolve from state a to state b where the cost has been minimized by the pursuer and maximized by the evader. Let the set **0** be the subset of X containing all the final equilibrium states.

***Definition* 5 :** The cost distance between a point $x \in X$ and a subset A of X is given by

$$\delta(x, A) = \inf\{\delta(x,a) : a \in A\}$$

We define *d*(*a*,*b*) as

$$d(a,b) = \left|\delta(a,\mathbf{0}) - \delta(b,\mathbf{0})\right|$$

***Definition* 6 :** The anti-symmetric distance between a point $x \in X$ and a subset A of *X* is given by

$$d(x,A) = \inf\{d(x,a) : a \in A\}$$

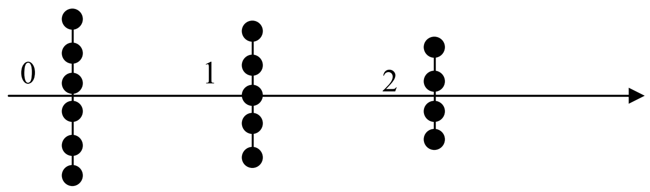


Fig. 6.6 Representation of the points of the set on a real line based on *d*(*x*,**0**)

Fig. 6.6 shows how the points of the set *X* can be represented on a real line based on the distances from the final equilibrium set **0**. Notice that on the real line, the sets occupy the positions of non-negative integers only. Notice also that there can be multiple points on the same location. Notice that the function *d*(*a*,*b*) as we defined satisfies the two axioms of a distance.

We can define an open sphere in order to develop a topology for the problem. For any point $x \in X$ and any real number r>0, we define an open ball as:

$$B(x,r) = \{y : d(y,x) < r\}$$

Lemma* 1*: Let *B* be an open ball with center *a* and radius *r*. For every point $b \in B$, there exists an open ball *C* centered at *b*, so that *C* is contained in the ball *B*.

Proof: There can be various different possibilities for the ball. In one case the ball might contain no points of the set *X*, or it may contain one or more points. We will proceed with the proof analyzing the various possibilities as follows:

1. <u>Ball containing no points of *X*:</u>

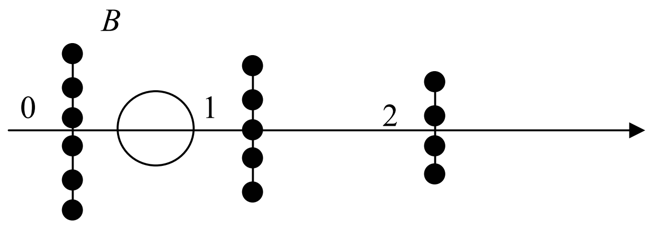


Fig. 6.7 Ball containing no points of the set *X*.

For this case, the lemma is satisfied in a trivial manner, since the ball contains no points of the set *X*.

2. Ball containing only one positive integer when represented on a real line:

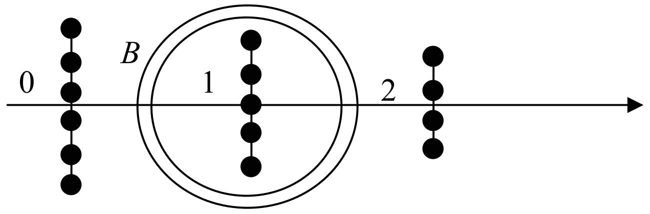


Fig. 6.8 Ball containing only one positive integer when represented on a real line.

In this case, the ball might contain multiple points, but all the points are located at the same location on the real number line. Therefore, it is very easy to find a ball centered at these points that is contained in the ball *B*.

3. Ball containing multiple positive integers when represented on a real line:

This case can also be represented on the real line. In this case we can again easily construct balls around each of the integer locations contained in the ball *B*, so that those balls are also contained in *B*.

Lemma 2*:* Let B_1 and B_2 be two open balls and x be a point belonging to both, such that $x \in B_1 \cap B_2$. Then, there exists an open ball B_x centered at x, such that $x \in B_x \subset B_1 \cap B_2$.

Proof: To obtain the proof, let us consider the real line representation as shown in Fig. 6.9.

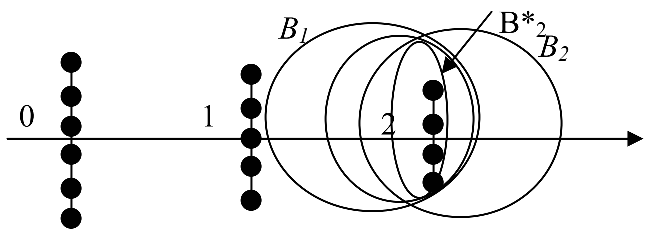


Fig. 6.9 A point belonging to two balls.

Let B_1 and B_2 be two open balls and x be a point belonging to both, such that $x \in B_1 \cap B_2$ as shown in Fig. 6.9. Since $x \in B_1$, according to Lemma 1, there exists a ball B*$_1$ centered at *x* that is contained in B$_1$. Similarly, since $x \in B_2$, according to Lemma 1, there exists a ball B*$_2$ centered at *x* that is contained in B$_2$. Now, we have two concentric balls, B*$_1$ and B*$_2$. Therefore, one should be contained in the other. In this example, we take B*$_2$ contained in B*$_1$. Therefore, we have obtained B*$_2$ as B_x centered at x, such that $x \in B_x \subset B_1 \cap B_2$.

Theorem* 7:* The class of open balls in set *X* of all the states of the game with the function *d* is a base for a topology on *X*.

Proof: Lemma 1 and Lemma 2 combined applied to Theorem 6.1 in [52] proves this theorem.

We define a norm in a slightly different way than the standard norm definitions to deal with different structure of the problem.

***Definition* 7** (*Norm*)*:* Norm on *X* is a function that assigns to each member *x* of *X* a real value $\|x\|$ and follows the following axiom.

***axiom* 3**: $\|x\| \geq 0$ and $\|x\| = 0$ iff x is a final equilibrium state

Theorem* 8:* The function *d*(*x*,**0**) is a norm function. From now on we denote this function as *d*(*x*). Note that this function is the same as the value function $V_{\mu\omega}(\mathbf{x})$ when the policies μ and ω have been applied.

Proof: Axiom 3 is satisfied by using the function definition in axiom 1.

***Theorem* 9**(*Lyapunov-like Theorem*)*:* When the policies μ and ω have been applied to the pursuit evasion game, and starting from any initial state *x*(0) if

$$d(x(k+1)) - d(x(k)) < 0 \text{ if } x(k) \neq 0$$

then $\exists T, s.t. x(T) \in \mathbf{0}$

Proof: Since there are only finite number of states, the system will keep traversing through different states (via the available links) in order to reduce *d*(*x*(*k*). In finite number of steps, the system will reach the **0** set.

***Theorem* 10**(*Boundedness Theorem*)*:* When the policies μ and ω have been applied to the pursuit evasion game, and starting from any initial state x(0) if

$$d(x(k+1)) - d(x(k)) \leq 0 \text{ if } x(k) \neq 0$$

then;

$$\forall k, d(x(k)) \leq d(x(0))$$

Dynamic programming solution to obtaining the pursuer and the evader policies can be presented as the solution of the following problem:

Problem Statement* 2:* Find the *policies* μ_* and ω_*. that produce the following function:

$$d_*(x(k)) = \min_{\mu} \max_{\omega} [2d(x(k+1)) - d(x(k))]$$

This problem statement is same as the problem statement 1. This will be restated in the standard dynamic programming terminology in the next section of this paper.

6.5. Dynamic Programming Solution to the NxN Grid Pursuer-Evader Herding Problem

The dynamic programming solution is based on Bellman's equation, which for our problem would look like the following:

$$V^*(\mathbf{x}(k)) = \min_{u\in\mu(\mathbf{x})} \max_{w\in\omega(x)} \{c_i(u,w) + V^*(\mathbf{x}(k+1))\}$$

This equation indicates how the feedback pursuer and evader can make decisions once the value function is available. The pursuer and the evader use the same function. This equation can also be used to find the value function using the boundary conditions from the problem. Small-scale problems can be solved by hand, but for large scale ones, a computer program can be written to apply the dynamic programming algorithm. The algorithm is applied similar to the problem of Fig 1.4 in [27].

6.6. **Simulation Software**

We have developed a Multiple Document Interface (MDI) windows application using Visual Basic for performing experiments. This software is available by sending an email to the first author of the paper. This program allows us to run many simulations at the same time, in different modes. The three different modes are (a) automatic, (b) user-assisted, and (c) manual.

In the automatic mode, the simulation runs by itself once started. We just observe the behavior of the pursuer and the evader. The simulation stops when the evader has reached its final position. In the user-assisted mode, the user needs to click *"Next button*" to make the evader or the pursuer move one step. This option allows more time to the user to - for example - think about the problem between consecutive moves. Finally in the manual mode, the user is controlling the pursuer and the evader movements using drag and drop.

The software produces a printable text history of the evader and pursuer moves as shown in Fig. 6.10.

```
Automatic Mode running ...

1) Evader (2, 2) --> (2, 2)
2) Pursuer (1, 1) --> (1, 2)
3) Evader (2, 2) --> (2, 1)
4) Pursuer (1, 2) --> (1, 2)
5) Evader (2, 1) --> (2, 0)
6) Pursuer (1, 2) --> (2, 1)
7) Evader (2, 0) --> (1, 0)
8) Pursuer (2, 1) --> (2, 1)
9) Evader (1, 0) --> (0, 0)

Evader has reached the home (0,0)
Simulation Ends ...
```

Fig. 6.10 Text Description of a Simulation Run

This simulation run can be graphically represented as shown in Fig. 6.11.

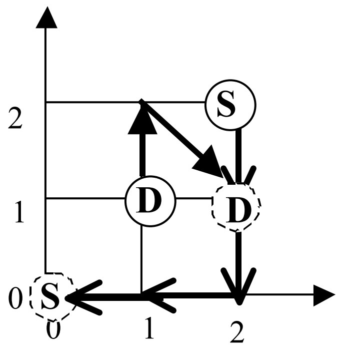


Fig. 6.11 Representation of a simulation run

6.7. Conclusion

In this chapter we studied a class of pursuit evasion problems that is different than the traditional problems in that the aim of the pursuer is to force the evader into a pen. We studied the properties of the problem and formulated the dynamic programming solution for the problem. We also presented a software package that has been developed to experiment with the problem.

References

1. Geoffrey F. Miller and Dave Cliff, “Technical Report” CSRP311, Aug. 1994.
2. Futuyama_ D. J. & Slatkin M. (Eds), ” Co-evolution” Sinauer, Sunderland, Massachusetts,1983.
3. Driver P., & Humphries N. “ Protean behavior; The biology of unpredictability” Oxford University Press, , 1988.
4. Dawkins R. &Krebs J.R., “Animal Signals; Information or Manipulation!”, In Krebs J. R.& Davies N. B. (Eds) “Behavioral ecology; An evolutionary approach”, pp: 282-309, Blackwell Scientific Oxford, 1984.
5. Harper D. “Communication”. In Krebs J. R. Davies N. B. (Eds) “Behavioral ecology; An evolutionary approach” third edition, Blackwell Scientific Press, Oxford, 1993.
6. Endler J. “Interactions between predators and prey” In Krebs J. R. Davies N. B. (Eds) , “Behavioral ecology; An evolutionary approach” 3rd Ed, pp 169-196., Blackwell Scientific, 1991.
7. Fagen R. , “Animal play behavior” Oxford U Press, 1981.
8. Hoyle G. “The scope of Neuroethology; *Behavioral and Brain Sciences*” 7, 367-412, 1984.
9. (e.g. Camhi, 1988; Krasne & Wine, 1987; Eaton, 1984) Camhi_ J. M. & Levy A. “Organization of a complex movement Fixed and variable components of the cockroach escape behavior”, Journal of Comparative Physiology; A Sensory Neural and Behavioral Physiology, 163 (3), 317-328.
10. Krasne F. &Wine J., “ Evasion responses of the crayfish” In Guthrie D. M.(Ed) Aims and Methods in Neuroethology, Manchester Univ. Press, 1987.
11. Yavin Y. & Pachter M. (Eds.) “Pursuit Evasion Differential Games”, Pergamon Press, 1987.
12. Isaacs R. The past and some bits of the future In Grote J.(Ed), “The Theory and Application of Differential Games”, pp 1-11, D. Reidel, 1975.
13. Richard S. Sutton and Andrew G. Barto, “Reinforcement Learning:An Introduction” MIT Press, Cambridge, MA, 1998,A Bradford Book.

14. von Neumann J. Morgenstern "Theory of Games and Economic Behavior", Princeton University Press, 1944.
15. Fudenberg D. & Tirole J., "Game Theory", MIT Press, 1991.
16. Brogan W. L., "Modern Control Theory", Prentice Hall, 1991.
17. Berkovitz L. D., "Two Person Zero Sum Differential Games; An overview", In Grote J. D. (Ed) "The theory and Application of Differential Games", pp:12-22, D. Reidel, 1975.
18. Rapoport A. "Two Person Game Theory", University of Michigan Press, 1966.
19. Ermolov A., Kryakovskii B. and Maslov E., " Differential Game with mixed strategies", Automatika i Telemekhanika, 47 (10), pp:1336-1349, 1986.
20. Houston A. and McNamara J. "Singing to attract a mate; A stochastic dynamic game", J. Theoretical Biology, 129, 57-68.
21. Clark C. W. and Ydenberg. R. C., "The risk of parenthood" I. general theory and applications. Evolutionary Ecology, 4 (1), pp:21-34, 1990.
22. Rodin E. Y., Lirov Y., Mittnik, S., McElhaney B. G., and Wilbur L., " Artificial intelligence in air combat games". In Yavin Y. and Pachter M. (Eds) "Pursuit evasion differential games", pp: 261-274 , Pergamon Press, 1987.
23. Joao P. Hespanha, Hyoun Jin Kim, and Shankar Sastray, "Multiple Agent Probabilistic Pursuit Evasion Games" Proceeding of the IEEE 38th conference on design and control, Phoenix, Arizona, USA, 1999.
24. Joao P. Hespanha, Maria Prandini, and Shankar Sastray, "Probabilistic Pursuit Evasion Games; A One Step Nash Approach" Proceeding of the IEEE 39th conference on design and control, Sydney, Australia, 2000.
25. Tuoxas Raivio, and Harri Ehtamo, "Applying Non-linear programming to a Complex Pursuit Evasion Problem", IEEE transaction on Automatic Control, 1997.
26. Boris Stilman, "Heuristic Networks for Concurrent Pursuit Evasion Systems", IEEE, 1995.
27. Pushkin Kachroo, SamyA. Shedied, J. S. Bay, and Hugh Vanlandingham, "Dynamic Programming Solution for a Class of Pursuit Evasion Problems: The Herding Problem," *IEEE Trans. Systems, Man and Cybernetics, Part C, Feb., 2001.*

28. Pushkin Kachroo, Samy A. Shedied, and Hugh Vanlandingham "Pursuit Evasion: The Herding Non-cooperative Dynamic Game," *Transactions of SDPS, (to be published).*
29. Pushkin Kachroo, Samy A. Shedied, and Hugh Vanlandingham "Pursuit Evasion: The Herding Non-cooperative Dynamic Game: The Stochastic Model," *IEEE Trans. Systems, Man and Cybernetics, Part C (to be published).*
30. Rufus Isaacs, "Differential Games: A Mathematical Theory with Application to Warfare and Pursuit Control and Optimization", Dover Publications Inc., NY, , 1965.
31. Berkovitz, L. D. and W. H. Flemming, "A Variational Approach to Differential Games," Annals of Mathematics Study 39, Princeton University Press, Princeton, N. J. 1957.
32. Sutton, Richard S., and Barto, Andrew G, "Reinforcement Learning: An Introduction," M.I.T. Press, 1999.
33. George M. Ewing, *Calculus of Variations with Applications*, Dover Publication Inc, 1969
34. Frank L, Lewis, Vassilis L. Syrmos, *Optimal Control*, New York : J. Wiley, c1995.
35. George Leitmann., *The calculus of variations and optimal control : an introduction*, New York : Plenum Press, c1981.
36. J. J. E. Slotine and Weiping Li, *Applied Nonlinear Control*, Prentice Hall, 1990.
37. Jean-Paul Laumond, *Robot Motion Planning and Control*, Springer, 1998.
38. A. Bellaiche, J.P. Laumond, and J. Jacobes, "Controllability of Car Like Robots and Complexity of Motion Planning Problem", International Symposium on intelligent Robotics, 322-337, Banagalore, India, 1991.
39. W. L Chow "Uber Systeme von Linearen Partiellen Differentialgleichungen erster Ordnung," Math Ann., 117, 98-115,1940.
40. Leslie Pack Kaelbling Michael L. Littman , "Reinforcement Learning: A Survey", 1996
41. Mance E. Harmon, Stephanie S. Harmon, "Reinforcement Learning: A Tutorial", Wright State University, 1999.
42. R. E Bellman, "Dynamic Programming", Princeton University Press, Princeton 1969.

43. Leon Cooper, and Marry W Cooper, “Introduction to Dynamic Programming”, 1981, Pregamon Press Ltd.
44. S.E. Dreyfus, and L.G. Mitten, “Elements of Sequential Decision Process”, Journal of Industrial Engineering, 18, 106-112, 1965.
45. R.M. Karp, and M. Held, “Finite State Process and Dynamic Programming’, SIAM Journal of Applied Mathematics, 15, 693-718, 1967.
46. Dimitri P. Bertesekas, “Dynamic Programming and Optimal Control”, Vol 1Athena scientific , Belmont, Massachusetts, 1995.
47. Dimitri P. Bertesekas,” Linear Network Optimization; Algorithms and Codes”, MIT press, MA, 1991.
48. Dimitri P. Bertesekas,” The Auction Algorithms for Shortest paths”, SIAM journal on optimization, Vol. 1, pp 425-447.
49. T. H. Cormen, C. E. Lieserson, and R. L. Rivest, “Introduction to Algorithms”, MIT press, 1998.
50. Basar Tamer, and Olsder Greet Jan, “Dynamic non-cooperative Game Theory”, 2nd Ed., Academic Press, 1987.
51. A. Kaufmann, *Graphs, Dynamic Programming, and Finite Games*, Academic Press, New York, 1967.
52. V. K. Balakrishnan, *Introductory Discrete Mathematics*, Dover Publications, Inc., New York, 1991.
53. Lipschutz, Seymour, *General Topology*, McGraw Hill, 1965.

VITA

Samy A Shedied was born in El-Bagour Menofia, Egypt on the 4th of September 1968. He received a Bachelor of Science degree in electrical engineering from MTC in Cairo, Egypt in 1990. After graduation, he joined the teaching staff of the MTC and was employed as a lecturer assistant while working toward his master degree. Mr. Shedied received his Master of Science degree in 1996. He was awarded a scholarship to continue his academic career and joined Virginia Polytechnic Institute and State University (Virginia Tech) in August 1998 to get the PhD in Electrical Engineering. Since August 1998, Mr. Shedied has been a member of the Multi-Agent Biological Learning (MABL) group in the Bradley Department of Electrical and Computer Engineering in VA Tech. On his track to get the PhD degree, he received a Master of Science degree in Mathematics form Virginia Tech in Aug. 2001. As a member of the MABL group, Mr. Shedied was involved in the design and development of optimal control policies for dynamic systems and pursuit evasion games.

His research interests include speech recognition and enhancement, robotics, systems and control, and digital signal processing.