Numerical Solution Methods for Differential Game Problems

by

Philip A. Johnson
B.S. Systems Engineering
United States Naval Academy, 2007

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Master of Science in Aeronautics and Astronautics
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2009

Author .....
Department of Aeronautics and Astronautics
May 22, 2009

Certified by .....
Steven R. Hall, Ph.D.
Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by ....
Russell Smith, Ph.D.
Principal Member of Technical Staff, C. S. Draper Laboratory, Inc.
Thesis Supervisor

Accepted by .....
David L. Darmofal, Ph.D.
Associate Department Head Chair, Committee on Graduate Students

Numerical Solution Methods for Differential Game Problems

by

Philip A. Johnson

Submitted to the Department of Aeronautics and Astronautics
on May 22, 2009, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Differential game theory provides a potential means for the parametric analysis of combat engagement scenarios. To determine its viability for this type of analysis, three frameworks for solving differential game problems are evaluated. Each method solves zero-sum, pursuit-evasion games in which two players have opposing goals. A solution to the saddle-point equilibrium problem is sought in which one player minimizes the value of the game while the other player maximizes it. The boundary value method is an indirect method that makes use of the analytical necessary conditions of optimality and is solved using a conventional optimal control framework. This method provides a high accuracy solution but has a limited convergence space that requires a good initial guess for both the state and less intuitive costate. The decomposition method in which optimal trajectories for each player are iteratively calculated is a direct method that bypasses the need for costate information. Because a linearized cost gradient is used to update the evader's strategy the initial conditions can heavily influence the convergence of the problem. The new method of neural networks involves the use of neural networks to govern the control policy for each player. An optimization tool adjusts the weights and biases of the network to form the control policy that results in the best final value of the game. An automatic differentiation engine provides gradient information for the sensitivity of each weight to the final cost. The final weights define the control policy's response to a range of initial conditions dependent upon the breadth of the state-space used to train each neural network. The neural nets are initialized with a normal distribution of weights so that no information regarding the state, costate, or switching structure of the controller is required. In its current form this method often converges to a suboptimal solution. Also, creative techniques are required when dealing with boundary conditions and free end-time problems.

Thesis Supervisor: Steven R. Hall, Ph.D.
Title: Professor of Aeronautics and Astronautics

Thesis Supervisor: Russell Smith, Ph.D.
Title: Principal Member of Technical Staff, C. S. Draper Laboratory, Inc.

Acknowledgments

I would like to express my sincere thanks to [redacted] MIT, and Draper Laboratory for providing this incredible opportunity. Specifically, I would like to thank my advisor, Russell Smith, for his insight and guidance during my time as a Draper Fellow. I am in debt as well to the faculty and staff of MIT's Department of Aeronautics and Astronautics who have academically challenged and stretched me beyond my expectations. Special thanks are due to my MIT advisor, Steven Hall, for his commitment to this work and for introducing me to differential games. This thesis would not have been possible without his consistent input and direction. Additionally, I would like to thank all the current and former Draper Fellows who provided research ideas, classwork collaboration, and friendship over the past two years. I would like to thank my friends and family for their encouragement and prayers throughout the development and writing of this thesis. Finally, I would like to thank my wife, Christie, for her unending support, patience, and love.

Acknowledgment

May 22, 2009

This thesis was prepared at The Charles Stark Draper Laboratory, Inc., under contract number N00030-08-C-0010.

Publication of this thesis does not constitute approval by Draper or the sponsoring agency of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

Philip A. Johnson

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

An original motivation for this thesis was to discover what techniques could provide an offline, parametric assessment of engagement scenarios between a hostile watercraft and a surfaced submarine guarded by an escort ship. Given a set of performance characteristics, should the escort pursue the hostile ship or stay close to the submarine and try to block? How much time will it take for the hostile ship to get around the escort? What performance enhancements could ensure the submarine does not get hit? Differential game theory emerged as a technique potentially suited to answer these questions. This thesis tests the viability of the differential game approach by surveying the current state of differential game theory and evaluating several numerical solution methods using standard engineering software. The methods are employed on a variety of test problems to provide insight into the future use of differential game theory for realistic problems such as the submarine engagement scenario.

1.1 Differential Games

Rufus Isaacs pioneered differential game theory while working for the RAND Corporation in the early 1950s [1, 2, 3, 4, 5]. His series of research memorandums not only established the study of differential games but also introduced concepts of control theory that would not be independently discovered until years later. In a review of Isaacs' 1965 publication of *Differential Games* [6], a compilation of his previous research memorandums and some new examples, Yu-Chi Ho remarked, "[Isaacs] thus anticipated practically all the important control-theoretic results known to date by nearly a decade. In fact, in several instances the book went beyond present control theory knowledge" [7].

Applications of differential games to problems of conflict and warfare were sought from the beginning [6, p. 305]. Pursuit-evasion games are a subset of differential game theory that describe a conflict, usually between two players, in which one player acts as the pursuer, P, and another player as the evader, E. This scenario is a zero sum game in which one player's gain is an equivalent loss to the opposing player. Pursuit-evasion games can be used to model numerous combat scenarios such as the movement

of ground troops, ships, missiles, satellites, or aircraft.

Using the theory established in early chapters, the majority of Isaacs' work is devoted to analytically solving specific differential game problems that are posed both for the reader's *instruction* and *enjoyment* [6, p. 24]. This process can be mathematically rigorous and often requires *a priori* intuition regarding the switching structure of the solution. Throughout the discipline's early decades, nearly all work in differential games was done analytically using simplified dynamics to describe pursuit-evasion scenarios [8, 9, 10, 11, 12].

1.2 Problem Formulation

Considering Isaacs did not make his work widely available until 1965, his notation and terminology were not commonly used and have since been abandoned in favor of conventional optimal control terminology and problem formulation. Bryson and Ho [13, 14, 7] derive the necessary conditions required in order for the cost function J to have a stationary value (not explicitly the maximization or minimization of the cost function). In a similar work, Başar and Oldser [15] provide a set of necessary conditions for an open-loop representation of the feedback saddle-point solution.

The differential games examined in this work will adhere to the following basic framework. Begin with the dynamic system describing an uncoupled, forced pair of state equations relating to each player[1]

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{x}}_P \\ \dot{\mathbf{x}}_E \end{bmatrix} = \begin{bmatrix} \mathbf{f}_P\left(\mathbf{x}_P, \mathbf{u}_P, t\right) \\ \mathbf{f}_E\left(\mathbf{x}_E, \mathbf{u}_E, t\right) \end{bmatrix} \tag{1.1}$$

with initial conditions $\mathbf{x}(t_0) = \mathbf{x}_0$ and $t_0 \leq t \leq t_f$. The initial time t_0 is fixed (typically $t_0 = 0$) and the final time t_f can either be fixed or free. Not all components of $\mathbf{x}(t_0)$ must be specified. The terminal constraints are compiled into the equation

$$\boldsymbol{\Psi}\left(\mathbf{x}_0, \mathbf{x}_{t_f}, t_f\right) = 0 \tag{1.2}$$

$\boldsymbol{\Psi}$ can be a vector of multiple terminal constraints. However, for the purposes of this thesis $\boldsymbol{\Psi}$ is only composed of the the capture condition

$$l(\mathbf{x}(t_f), t_f) = 0 \tag{1.3}$$

In most pursuit-evasion games l expresses the distance or capture radius between the two players at the end of the game.

The performance criterion or cost functional is

$$J = \phi\left(\mathbf{x}\left(t_f\right), t_f\right) + \int_{t_0}^{t_f} L(\mathbf{x}, \mathbf{u}_P, \mathbf{u}_E, t)\, dt \tag{1.4}$$

and the goal of the zero sum differential game is to find the optimal controls (denoted

[1]Boldface denotes vector-valued quantities

by an asterisk) $\mathbf{u}_P^*$ and $\mathbf{u}_E^*$ such that

$$J(\mathbf{u}_P^*, \mathbf{u}_E) \leq J(\mathbf{u}_P^*, \mathbf{u}_E^*) \leq J(\mathbf{u}_P, \mathbf{u}_E^*) \tag{1.5}$$

The pursuer seeks to minimize the cost functional, J, whereas the evader seeks to maximize it. Any deviation from the optimal control set will be to the other player's advantage.

Two open-loop feedback strategies, γ_P and γ_E, are introduced so that the control policies may be expressed as functions of the initial state variables as

$$\mathbf{u}_P(t) = \boldsymbol{\gamma}_P(\mathbf{x}_P(0), t) \quad \text{and} \quad \mathbf{u}_E(t) = \boldsymbol{\gamma}_E(\mathbf{x}_E(0), t) \tag{1.6}$$

The *value* of the game

$$V = \min_{\boldsymbol{\gamma}_P^*} \max_{\boldsymbol{\gamma}_E^*} J = \max_{\boldsymbol{\gamma}_E^*} \min_{\boldsymbol{\gamma}_P^*} J = J(\mathbf{u}_P^*, \mathbf{u}_E^*) \tag{1.7}$$

is defined as the outcome of the objective function when both players execute their respective optimal strategies given the initial state condition $\mathbf{x}_0$. The existence of the minimax value is dependent upon the system dynamics being separable [13, p. 278].

Bryson and Ho [13, p. 277] and Başar and Olsder [15, p. 433] provide the necessary conditions for the existence of an open-loop representation of the saddle-point solution. The Hamiltonian H is defined as

$$H \equiv \boldsymbol{\lambda}^T \mathbf{f} + L \tag{1.8}$$

where the costate and derivative vectors are

$$\boldsymbol{\lambda} = \begin{bmatrix} \boldsymbol{\lambda}_P \\ \boldsymbol{\lambda}_E \end{bmatrix} \quad \text{and} \quad \mathbf{f} = \begin{bmatrix} \mathbf{f}_P \\ \mathbf{f}_E \end{bmatrix} \tag{1.9}$$

respectively, and the terminal conditions are represented as

$$\boldsymbol{\Phi} \equiv \phi + \mathbf{v}^T \boldsymbol{\Psi} \tag{1.10}$$

The term ϕ is the terminal cost, and $\mathbf{v}$ is the Lagrange multiplier for terminal constraint $\boldsymbol{\Psi}$ [13, p. 65]. Raivio and Ehtamo [16] use the multiplier α to describe the Lagrange multiplier that corresponds to the capture condition l.

The following variational equations hold for the costate $\boldsymbol{\lambda}$, assuming no integral cost L,

$$\dot{\boldsymbol{\lambda}} = -H_{\mathbf{x}} = -\left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]^T \boldsymbol{\lambda} \tag{1.11}$$

with the costate terminal (or capture) condition

$$\boldsymbol{\lambda}^T(t_f) = \frac{\partial \phi(\mathbf{x}(t_f), t_f)}{\partial \mathbf{x}} \tag{1.12}$$

The saddle-point solution must satisfy the conditions

$$\begin{aligned} \mathbf{u}_P &= \arg\min_{\mathbf{u}_P} H &&= \arg\min_{\mathbf{u}_P} \left(\boldsymbol{\lambda}_P^T \mathbf{f}_P\right) \\ \mathbf{u}_E &= \arg\max_{\mathbf{u}_E} H &&= \arg\max_{\mathbf{u}_E} \left(\boldsymbol{\lambda}_E^T \mathbf{f}_E\right) \end{aligned} \tag{1.13}$$

In the case of unconstrained controls this argument becomes

$$\begin{aligned} H_{\mathbf{u}_P}^T &= \left[\frac{\partial \mathbf{f}_P^T}{\partial \mathbf{u}_P}\right] \boldsymbol{\lambda}_P = 0 \\ H_{\mathbf{u}_E}^T &= \left[\frac{\partial \mathbf{f}_E^T}{\partial \mathbf{u}_E}\right] \boldsymbol{\lambda}_E = 0 \end{aligned} \tag{1.14}$$

with the second order conditions relating to the Hessian matrix of a saddle-point solution

$$\begin{aligned} H_{\mathbf{u}_P \mathbf{u}_P} &= \frac{\partial^2 H}{\partial \mathbf{u}_P^2} \geq 0 \\ H_{\mathbf{u}_E \mathbf{u}_E} &= \frac{\partial^2 H}{\partial \mathbf{u}_E^2} \leq 0 \end{aligned} \tag{1.15}$$

$H_{\mathbf{u}_P \mathbf{u}_P}$ must be positive semi-definite and $H_{\mathbf{u}_E \mathbf{u}_E}$ must be negative semi-definite. The transversality condition for this free final time problem is

$$H\left(\mathbf{x}^*\left(t_f\right), \mathbf{u}_P^*, \mathbf{u}_E^*, \boldsymbol{\lambda}\left(t_f\right), t_f\right) = -\frac{\partial \boldsymbol{\Phi}}{\partial t_f} = -\frac{\partial \phi}{\partial t_f} - \mathbf{v}^T \frac{\partial \boldsymbol{\Psi}}{\partial t_f} \tag{1.16}$$

Equations 1.1, 1.3, 1.11, 1.12, 1.14, 1.15, and 1.16, form a two-point boundary value problem (TPBVP) with unknowns $\mathbf{x}^*(t)$, $\mathbf{u}^*(t)$,$\boldsymbol{\lambda}^*(t)$, and $\mathbf{v}$.

Starting with the initial state $\mathbf{x}_0$, the pursuer attempts to move the evader towards the terminal surface where capture takes place such that $l(\mathbf{x}(t_f), t_f) = 0$. A barrier separates the state space into two different zones: the *capture zone* and *evasion zone* [6, p. 203]. In the capture zone, the pursuer will always be able to capture the evader assuming that the pursuer acts optimally. Conversely, in the evasion zone no amount of effort by the pursuer will result in a capture unless the evader cooperates and acts non-optimally.

1.3 Numerical Solutions for Pursuit Evasion Games

Early work in pursuit-evasion games took place before the wide availability of computers and software packages that were capable of solving realistic problems using modern numerical methods. Consequently, the application of differential game theory to useful problems has been limited, since all calculations had to be done analytically. Breakwell and Merz [17] calculated the minimum capture radius required in order to guarantee capture of the evader under the assumption of constant speeds and specified turn rates. Farber and Shinar [18] developed an approximate feedback solution to a variable speed, coplanar aerial pursuit-evasion game. Rajan, et al. [19] examined the coplanar, two aircraft problem through the use of the individual aircraft's

extremal trajectory maps. Guelman, et al. [20] were able to do a similar coplanar, aerial pursuit-evasion game that included aerodynamic drag as a function of the angle of attack.

The solution of pursuit-evasion games with realistic dynamics must be found numerically. As is the case for optimal control problems, numerical solutions for differential games often fall into two categories, *direct* methods and *indirect* methods [21]. The basic procedure for indirect methods is to take the system dynamics, form the Hamiltonian, derive the necessary conditions, then solve the boundary value problem numerically using information about the states and costates at the boundaries. The theory behind this approach was outlined in Section 1.2. Conversely, direct method solvers need only be provided with the system dynamics, control and state constraints, and the objective function without the need for costate information.

Indirect (boundary value) methods are characterized by their use of the analytical necessary conditions, the production of which can be a time consuming and difficult process as the dimensionality of the problems increase. Various solvers exist that can solve two point or multi-point boundary value problems. Often, these solvers are based on the multiple shooting method [22] or the Simpson's method [23]. Indirect methods are marked by their high accuracy and the production of a solution that satisfies the necessary conditions of optimality. They are, however, very difficult to solve without providing a plausible initial guess not only for the state trajectories but also for the costates. With constraints on the states and/or controls, the user must provide the solver with a guess that incorporates knowledge of the switching structure between constrained and unconstrained operation. Determining these subarcs requires an intuition about the solution that can be difficult when realistic, nonlinear dynamics are implemented.

Direct methods are so named because they do not make use of the analytical necessary conditions. The continuous differential game problem is converted into a discrete non-linear programming (NLP) problem. Direct Collocation with NLP or DCNLP solvers discretize the state and control trajectories and use splines for interpolation between them. Pseudospectral Methods are another collocation method that expand the state and controls as a finite sum of Lagrange polynomials evaluated at Chebyshev-Gauss-Lobatto points [24]. Though they may achieve less accuracy than indirect methods, direct methods provide a means to avoid the analytical derivation of the necessary conditions. A guess for the costate and its switching structure is no longer needed. Costate information is not provided by the solver and would require *ex post facto* computation.

Early attempts at solving differential game problems numerically used differential dynamic programming. Järmark, Merz, and Breakwell [25] solved a variable speed, tail-chase, aerial combat problem that took place in the horizontal plane. Hillberg and Järmark examined a similar problem using steady turn rates and realistic thrust and drag data. However, this approach is plagued by the "curse of dimensionality" [26] and has been abandoned in favor of more advanced numerical techniques.

In 1993, Breitner, Pesch, and Grimm [22, 27] sought to narrow the gap in the complexity of optimal control problems being solved at the time compared to the meager application of differential game theory to realistic problems. Using a numerical

solver that employed the indirect, multiple shooting method, they solve a pursuit-evasion game between a realistically modeled air-to-air missile and high performance aircraft in the vertical plane. As is the case with indirect methods, this solution requires extensive analysis regarding the necessary conditions and switching structure of unconstrained and constrained sub-arcs in order to form the multi-point boundary value problem. Breitner states that direct methods cannot be applied to pursuit-evasion games and that dynamic programming and indirect methods are the only appropriate methods to find numerical differential game solutions [22].

In a series of papers written in the early 2000s, Raivio, et al. [28, 16] proposed the decomposition of the saddle point problem into two optimal control problems that could be solved iteratively using either indirect or direct methods. Raivio opted to use discretization and NLP to provide solutions to a variety of problems, including a visual aircraft identification pursuit-evasion game [29] that incorporated realistic, nonlinear dynamics.

More recently, Horie and Conway [30] proposed a hybrid method called *semi-DCNLP*. This method attempts to reformulate the differential game problem into a form that can be handled by a numerical optimizer designed to solve optimal control problems. This transformation is accomplished by augmenting the state vector with the costate of one of the players and adjusting the terminal constraints. A detailed description of the process can be found in [31, 32, 33].

Virtually all numerical solutions to differential game problems mentioned have relied on customized software packages to solve BVPs and NLP problems. Several of these solvers have fallen out of popular use in the decades following their introduction. One goal of the current work is to evaluate how well MATLAB can implement the methods mentioned above. Though not as fast as an equivalent purpose-built solver, MATLAB offers an ease of use that lowers the barrier of entry for anyone trying to solve differential game problems.

1.4 Two Games of Interest

As previously mentioned, surface transit protection for submarines was an original motivation for this thesis. Due to their lack of maneuverability and armament, submarines are vulnerable to *USS Cole*-style attacks from hostile watercraft. Of specific interest are scenarios in which a slower escort ship, the pursuer, attempts to intercept a significantly faster hostile ship, the evader, at a maximum distance from the submarine. The evader would attempt to minimize the distance to the submarine at the intercept time while the pursuer would try to maximize it. This thesis examines previously established problems that contain elements similar to the submarine problem in order test the viability of three differential game solution methods for future use during complex engagement scenario analysis.

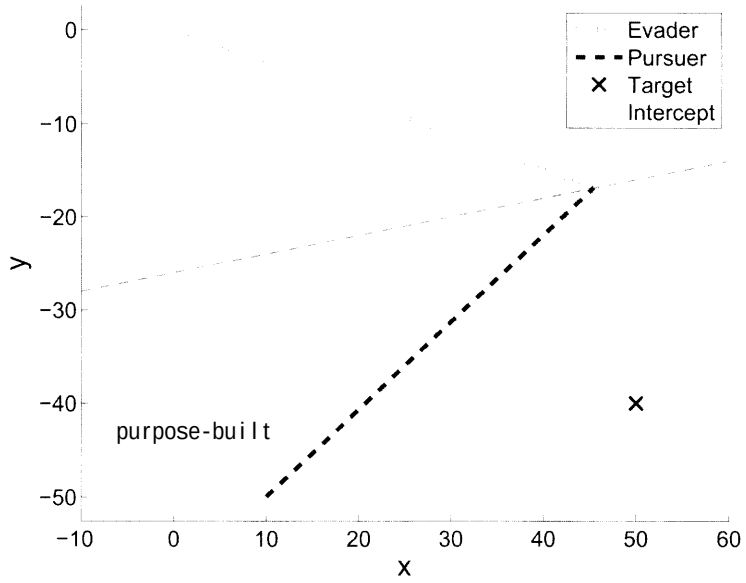

Figure 1-1: The Target Guarding Problem

1.4.1 The Target Guarding Problem

In the target guarding problem as described by Isaacs [6, p. 19], P and E have an equal, constant speed and *simple motion*, which means there are no restrictions on how fast each player can change direction. E aims to get as close as possible to the target C while P, who is guarding the target, attempts to intercept E as far as possible from the target. Thus, the objective function is the distance away from the target at the terminal (intercept) time.

The solution of this game can be found geometrically. The dashed line in the center of Figure 1-1 represents a centered, perpendicular line between PE that Isaacs calls the *perpendicular bisector* [6, p. 19]. The area above the bisector represents all locations which E could reach before P, and the area below the bisector represents locations that P could reach before E. The optimal solution for E is to aim for the point along the bisector that is the closest to the target.

1.4.2 The Homicidal Chauffeur Problem

The homicidal chauffeur problem is one of the best known and most discussed problems in pursuit-evasion games. Originally posed by Isaacs [6, p. 11], the problem was given a more thorough treatment by Merz [34, 9] and was revisited by other authors through the years [35]. The problem can be viewed as a simplified pursuit-evasion game between a fast pursuer with limited maneuverability and a slower but more agile evader.

The problem is posed such that a car and a pedestrian are located in an infinitely

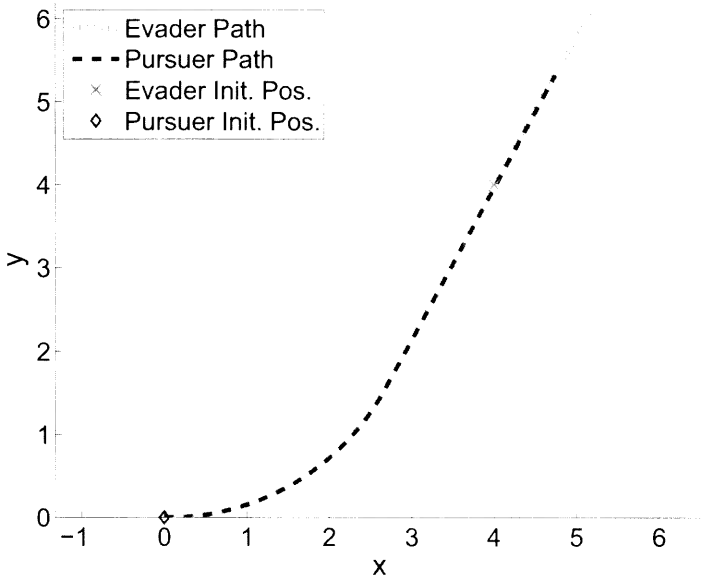

Figure 1-2: The Homicidal Chauffeur Problem

large parking lot. The objective of the game is for the car, P, to run over the pedestrian, E, whose goal it is to avoid capture. Both P and E have fixed speed where $v_P > v_E$. P is subject to a constraint on his turn rate whereas E has simple motion. Capture occurs when E gets within the capture radius of P. Figure 1-2 shows an example of a chase in which P begins at the origin with his velocity vector along the x-axis and turns left to intercept E.

1.5 Thesis Outline

This work provides a relevant summary of the application of differential game theory to problems that can be solved numerically using MATLAB. Ideally, the barrier of entry will be lowered for other students and researchers who would like to know what options exist for solving a differential or pursuit-evasion game problem using standard engineering software. The games of interest addressed in Section 1.4 will provide a baseline by which each of the techniques can be compared.

Chapter 2 examines the boundary value approach including how to create an initial guess that achieves convergence and what intuitions are necessary regarding the structure of the costate trajectory. In Chapter 3 the Decomposition method is implemented using MATLAB's `fmincon` function to iteratively solve a min and max problem. Finally, Chapter 4 surveys a new technique for solving differential game problems numerically using neural networks. The code for several of the solved problems is provided in the appendices accompanied by comments to guide the reader through the solution algorithms.

Chapter 2

Boundary Value Method

This chapter explains how to approach differential game problems using a traditional optimal control framework. Section 2.1 discusses the transformation required to solve a free end-time problem and explains what functions are used in the BVP solver. The target guarding and homicidal chauffeur problems are converted into two-point BVPs and solved by numerical methods using MATLAB in Sections 2.2 and 2.3.

2.1 Approach

The solution approach will follow the steps outlined in the problem formulation of Section 1.2. The primary steps are to take the system dynamics, form the Hamiltonian equation, analytically derive the necessary conditions, and enforce the boundary conditions at the endpoints. While any numerical boundary value problem solver can be used, this work will focus on MATLAB's `bvp4c` function. This built-in function is capable of solving two-point and multi-point boundary value problems for ordinary differential equations. Shampine and Kierzenka [23] provide a detailed description of how to set up and solve a variety of boundary value problems using `bvp4c`. Additional information can be found in the MATLAB Product Help.

The function `bvp4c` integrates a system of ordinary differential equations (ODEs) of the form

$$\frac{d\mathbf{y}}{dx} = \mathbf{f}(x, \mathbf{y}, \mathbf{p}), \tag{2.1}$$

subject to the general nonlinear, two-point boundary conditions

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b), \mathbf{p}) = 0$$

where x is the independent variable, $\mathbf{y}$ a vector containing the dependent variables, and $\mathbf{p}$ is a vector of unknown parameters.

The solver operates on the assumption that the time period in the problem is fixed, $t \in [a, b]$. In order to solve a free end-time problem time (the independent variable) must be rescaled so that $\tau = \frac{t}{t_f}$ and solve the problem for $\tau \in [0, 1]$. Because of this scaling all the ODEs describing the time derivatives of the state and costate must be

multiplied by the final time, because

$$d\tau = \frac{1}{t_f}\, dt \tag{2.2}$$

and

$$\frac{d\mathbf{y}}{d\tau} = t_f \frac{d\mathbf{y}}{dt} \tag{2.3}$$

An initial guess is required before `bvp4c` can evaluate a solution. This guess is created with the help of the `bvpinit` function that takes a vector of M mesh points and guesses at those points and turns them into a data structure that has the same format as the `bvp4c` function output. This feature is useful for occasions when `bvp4c` is placed in a loop where the solution of one iteration becomes the initial guess for the next. The guesses at the mesh points can be entered manually as a vector or by using a function handle that calculates values for $\mathbf{y}$ at each of the mesh-points using the commands

```
>> mesh = linspace(0,1,M)
>> solinit = bvpinit(mesh,@DG_init,alpha0)
```

An initial guess for the scalar α, which is the Lagrange multiplier for the capture condition, must also be provided. The `@DG_init` function handle returns a vector containing the values for the states, costate, and final time as a function of the vector of mesh-points. The initial guess is sent to `bvp4c` along with two function handles

```
>> sol = bvp4c(@DG_ode,@DG_bc, solinit)
```

The first function handle has the header

```
>> function dydtau = DG_ode(tau,y)
```

and evaluates the time derivative

$$\frac{d\mathbf{y}}{d\tau} = t_f \begin{bmatrix} \frac{d\mathbf{x}}{dt} \\ \frac{d\boldsymbol{\lambda}}{dt} \\ \frac{dt_f}{dt} \end{bmatrix} \quad \text{with} \quad \frac{dt_f}{dt} = 0 \tag{2.4}$$

at each mesh-point. The second function handle

```
>> function res = DG_bc(ya,yb)
```

computes the residual of the boundary conditions. The residual is composed of $2N+2$ boundaries which is equal to the number of unknowns. The unknowns in the problem are the N states, N costates, the capture condition Lagrange multiplier α, and the final time. The initial conditions, capture condition, costate terminal conditions, and Hamiltonian terminal condition must be expressed in a residual form. For example, if a system had the boundary conditions $y(0) = 0$ and $y(t_f) = 5$ then the first rows of the `res` vector would be

```
>> res = [ ya - 0
          yb - 5
           ...   ];
```

2.2 Target Guarding Problem

As explained in Section 1.4.1, the target guarding problem is a game between two simple motion bodies. The evader, E, attempts to minimize his final distance to the target C while the pursuer, P, attempts to intercept E at a distance as far as possible from the target.

2.2.1 Problem Dynamics

Begin with the equations of motion for each player[1]

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{x}}_P \\ \dot{\mathbf{x}}_E \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{x}}_1 \\ \dot{\mathbf{x}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix} = \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{x}_2 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} v_1 \cos u_1 \\ v_1 \sin u_1 \\ v_2 \cos u_2 \\ v_2 \sin u_2 \end{bmatrix} \tag{2.5}$$

with initial conditions

$$\mathbf{x}(t_0) = \begin{bmatrix} x_1(t_0) \\ y_1(t_0) \\ x_2(t_0) \\ y_2(t_0) \end{bmatrix} = \begin{bmatrix} x_{10} \\ y_{10} \\ x_{20} \\ y_{20} \end{bmatrix} \tag{2.6}$$

where x and y refer to the location of each player on the coordinate frame. Because both players have simple motion they are free to change their direction instantaneously.

In this problem the capture condition

$$\Psi(\mathbf{x}_f, t_f) \equiv l(\mathbf{x}(t_f), t_f) \equiv (x_1(t_f) - x_2(t_f))^2 + (y_1(t_f) - y_2(t_f))^2 - d^2 = 0 \tag{2.7}$$

is the only terminal constraint. The game concludes when the distance between the players is equal to the capture radius d. The objective of the game depends only on the final state of the players, thus $L(\mathbf{x}, u_1, u_2, t) = 0$ and

$$J = \phi(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\mathbf{x}, u_1, u_2, t)\, dt = \sqrt{(x_2(t_f) - x_C)^2 + (y_2(t_f) - y_C)^2} \tag{2.8}$$

where x_C and y_C are the coordinates for the target being guarded.

[1] To aid comparison with the MATLAB m-files the use of player *1* for P and player *2* for E will be adopted.

2.2.2 Necessary Conditions

The Hamiltonian equation is defined as

$$H \equiv \boldsymbol{\lambda}^T \mathbf{f} + L = \begin{bmatrix} \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 \end{bmatrix} \begin{bmatrix} v_1 \cos u_1 \\ v_1 \sin u_1 \\ v_2 \cos u_2 \\ v_2 \sin u_2 \end{bmatrix} \tag{2.9}$$

$$H = \lambda_1 v_1 \cos u_1 + \lambda_2 v_1 \sin u_1 + \lambda_3 v_2 \cos u_2 + \lambda_4 v_2 \sin u_2 \tag{2.10}$$

Because there is only one terminal constraint (applying to the capture condition l) the single multiplier α will be employed so that

$$\boldsymbol{\Phi} = \phi + \mathbf{v}^T \boldsymbol{\Psi} = \phi\left(\mathbf{x}\left(t_f\right), t_f\right) + \alpha l\left(\mathbf{x}\left(t_f\right), t_f\right) \tag{2.11}$$

The time derivative of the costate

$$\dot{\boldsymbol{\lambda}}(t) = -H_{\mathbf{x}} = -\begin{bmatrix} \frac{\partial H}{\partial x_1} \\ \frac{\partial H}{\partial y_1} \\ \frac{\partial H}{\partial x_2} \\ \frac{\partial H}{\partial y_2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.12}$$

is equal to the negative of the partial derivative of the Hamiltonian with respect to the state vector $\mathbf{x}$.

Because λ_1, λ_2, λ_3, and λ_4 have zero time derivatives, they will be constants throughout the game. The terminal conditions for the costate are

$$\boldsymbol{\lambda}^T(t_f) = \boldsymbol{\Phi}_{\mathbf{x}(t_f)} = \frac{\partial}{\partial \mathbf{x}} \phi\left(\mathbf{x}\left(t_f\right), t_f\right) + \alpha \frac{\partial}{\partial \mathbf{x}} l\left(\mathbf{x}\left(t_f\right), t_f\right) \tag{2.13}$$

$$\boldsymbol{\lambda}^T(t_f) = \begin{bmatrix} 0 \\ 0 \\ \frac{x_2(t_f) - x_C}{\sqrt{\left(x_C - x_2(t_f)\right)^2 + \left(y_C - y_2(t_f)\right)^2}} \\ \frac{y_2(t_f) - y_C}{\sqrt{\left(x_C - x_2(t_f)\right)^2 + \left(y_C - y_2(t_f)\right)^2}} \end{bmatrix} + \alpha \begin{bmatrix} 2\left(x_1\left(t_f\right) - x_2\left(t_f\right)\right) \\ 2\left(y_1\left(t_f\right) - y_2\left(t_f\right)\right) \\ 2\left(x_2\left(t_f\right) - x_1\left(t_f\right)\right) \\ 2\left(y_2\left(t_f\right) - y_1\left(t_f\right)\right) \end{bmatrix} \tag{2.14}$$

The transversality condition for this free final time problem is

$$\begin{aligned} H\left(\mathbf{x}^*\left(t_f\right), u_1^*, u_2^*, \boldsymbol{\lambda}\left(t_f\right), t_f\right) = -\frac{\partial}{\partial t_f} \boldsymbol{\Phi} = -\frac{\partial}{\partial t_f} \phi - \alpha \frac{\partial}{\partial t_f} l\left(\mathbf{x}\left(t_f\right), t_f\right) &= 0 \\ \lambda_1 v_1 \cos u_1 + \lambda_2 v_1 \sin u_1 + \lambda_3 v_2 \cos u_2 + \lambda_4 v_2 \sin u_2 &= 0 \end{aligned} \tag{2.15}$$

For the saddle point solution of this unconstrained problem,

$$u_1 = \arg\min_{u_1} H = H_{u_1} = 0 \tag{2.16}$$

$$u_2 = \arg\max_{u_2} H = H_{u_2} = 0 \tag{2.17}$$

The controls u_1 and u_2 can be written in terms of the costates for each player as

$$H_{u_1} = 0 = -\lambda_1 v_1 \sin u_1 + \lambda_2 v_1 \cos u_1 \tag{2.18}$$

$$u_1 = \tan^{-1} \frac{\lambda_2}{\lambda_1}$$

$$H_{u_2} = 0 = -\lambda_3 v_2 \sin u_2 + \lambda_4 v_2 \cos u_2 \tag{2.19}$$

$$u_2 = \tan^{-1} \frac{\lambda_4}{\lambda_3} \tag{2.20}$$

The dynamics of each player are reformulated in terms of the costate by a trigonometric identity, so that

$$\dot{\mathbf{x}}_1 = \begin{bmatrix} v_1 \cos u_1 \\ v_1 \sin u_1 \end{bmatrix} = \begin{bmatrix} v_1 \cos\left(\tan^{-1} \frac{\lambda_2}{\lambda_1}\right) \\ v_1 \sin\left(\tan^{-1} \frac{\lambda_2}{\lambda_1}\right) \end{bmatrix} = v_1 \begin{bmatrix} \frac{1}{\sqrt{1+\left(\frac{\lambda_2}{\lambda_1}\right)^2}} \\ \frac{\frac{\lambda_2}{\lambda_1}}{\sqrt{1+\left(\frac{\lambda_2}{\lambda_1}\right)^2}} \end{bmatrix} \tag{2.21}$$

$$\dot{\mathbf{x}}_2 = \begin{bmatrix} v_2 \cos u_2 \\ v_2 \sin u_2 \end{bmatrix} = \begin{bmatrix} v_2 \cos\left(\tan^{-1} \frac{\lambda_4}{\lambda_3}\right) \\ v_2 \sin\left(\tan^{-1} \frac{\lambda_4}{\lambda_3}\right) \end{bmatrix} = v_2 \begin{bmatrix} \frac{1}{\sqrt{1+\left(\frac{\lambda_4}{\lambda_3}\right)^2}} \\ \frac{\frac{\lambda_4}{\lambda_3}}{\sqrt{1+\left(\frac{\lambda_4}{\lambda_3}\right)^2}} \end{bmatrix} \tag{2.22}$$

Because they have zero time derivatives (Equation 2.12) the costates are constants during the entire game. Constant costates imply that the direction of travel for each player will also be constant throughout the game since the control angles are functions of the costates. So while simple motion allows for the freedom to change direction instantaneously, the best strategy for both players is to follow a straight line path.

2.2.3 Solution Implementation

The following initial conditions were used in this problem:

$$\begin{array}{ll} x_{10} = 10 \text{ m} & x_{20} = 0 \text{ m} \\ y_{10} = -50 \text{ m} & y_{20} = 0 \text{ m} \\ x_C = 50 \text{ m} & y_C = -40 \text{ m} \\ v_1 = 1 \text{ m/s} & v_2 = 1 \text{ m/s} \end{array}$$

Figure 2-1 shows the basic progression of the numerical solution. Given a set of initial conditions, the `bvpinit` function uses an embedded function called `TG_init` to create an initial guess for the state, costates, and terminal time. The initial guess is then sent to the `bvp4c` function that uses the `TG_ode` and `TG_bc` functions to find the solution to the boundary value problem. `TG_ode` describes the time derivatives of all states and costates. `TG_bc` is composed of a residual that includes terms for the

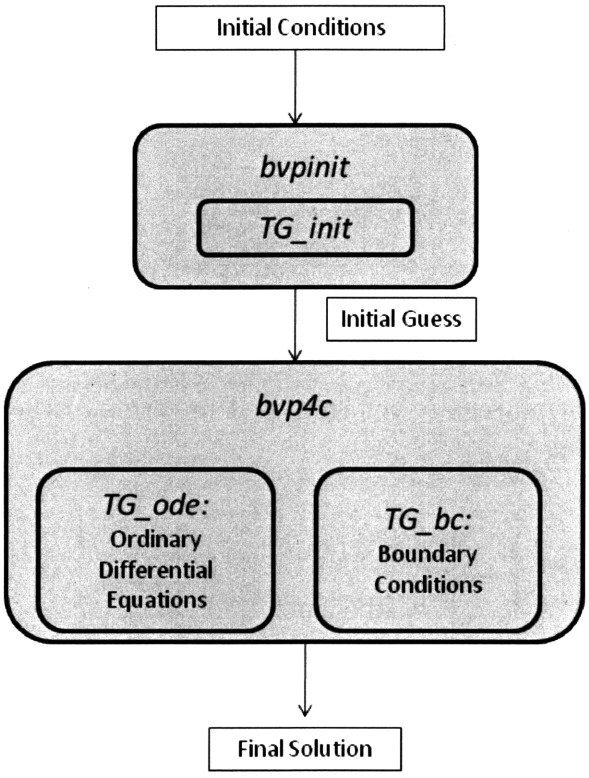

Figure 2-1: Target Guarding `bvp4c` Implementation

initial and terminal boundaries. The MATLAB code for this problem can be found in Appendix A.1.1.

2.2.4 Results

Figure 2-2 illustrates a successful target guarding interception found numerically using MATLAB. In order to get the solution to converge properly, a good initial guess was required. The guess incorporated the information that the pursuer and evader would have straight line paths to the capture location because of their constant costate terms. If an arbitrary initial guess is used, the solver fails to converge on the correct solution.

2.3 Homicidal Chauffeur Problem

The homicidal chauffeur problem pits a slow but maneuverable pedestrian against a faster, less agile car in an infinitely large parking lot. This problem adds a level of difficulty over the target guarding problem because there are two phases of pursuit for the car: the initial limited, maximum rate turn followed by a straight line pursuit and capture.

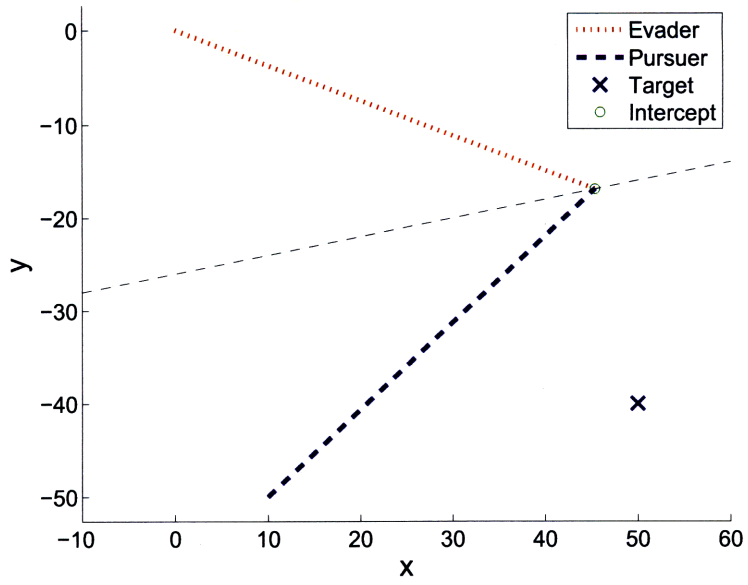

Figure 2-2: Target Guarding `bvp4c` Result

2.3.1 Problem Dynamics

The equations of motion for the pursuer (the car) and the evader (the pedestrian) are

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{x}}_P \\ \dot{\mathbf{x}}_E \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{x}}_1 \\ \dot{\mathbf{x}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix} = \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{\phi}_1 \\ \dot{x}_2 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} v_1 \cos\phi_1 \\ v_1 \sin\phi_1 \\ u_1 \\ v_2 \cos u_2 \\ v_2 \sin u_2 \end{bmatrix} \tag{2.23}$$

with initial conditions

$$\mathbf{x}(t_0) = \begin{bmatrix} x_1(t_0) \\ y_1(t_0) \\ \phi_1(t_0) \\ x_2(t_0) \\ y_2(t_0) \end{bmatrix} = \begin{bmatrix} x_{10} \\ y_{10} \\ \phi_{10} \\ x_{20} \\ y_{20} \end{bmatrix} \tag{2.24}$$

where x and y refer to the location of each player on the coordinate frame and ϕ_1 and u_2 are the player's heading angles relative to the x-axis. This game is a constant speed pursuit where $v_1 > v_2$.

While the pedestrian can change his direction instantaneously, the car has a limited turn rate. Thus, a constraint is placed on the absolute angular velocity of player 1 that

$$|u_1| \leq u_{max} \tag{2.25}$$

Again, the capture condition

$$\Psi\left(\mathbf{x}_f, t_f\right) = l(\mathbf{x}(t_f), t_f) = \left(x_1\left(t_f\right) - x_2\left(t_f\right)\right)^2 + \left(y_1\left(t_f\right) - y_2\left(t_f\right)\right)^2 - d^2 = 0 \quad (2.26)$$

is the only terminal constraint. The game concludes when the distance between the car and the pedestrian is equal to the capture radius d. The objective of the game depends only on the final time of the game, thus

$$J = \phi\left(\mathbf{x}\left(t_f\right), t_f\right) = t_f \quad (2.27)$$

2.3.2 Necessary Conditions

The necessary conditions are found through the manipulation of the Hamiltonian equation

$$H \equiv \boldsymbol{\lambda}^T \mathbf{f} + L = \begin{bmatrix} \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 \end{bmatrix} \begin{bmatrix} v_1 \cos\phi_1 \\ v_1 \sin\phi_1 \\ u_1 \\ v_2 \cos u_2 \\ v_2 \sin u_2 \end{bmatrix} \quad (2.28)$$

$$H = \lambda_1 v_1 \cos\phi_1 + \lambda_2 v_1 \sin\phi_1 + \lambda_3 u_1 + \lambda_4 v_2 \cos u_2 + \lambda_5 v_2 \sin u_2 \quad (2.29)$$

The single multiplier α is used for the terminal constraint on the capture condition l, so that

$$\boldsymbol{\Phi} = \phi + \mathbf{v}^T \boldsymbol{\Psi} = \phi\left(\mathbf{x}\left(t_f\right), t_f\right) + \alpha l\left(\mathbf{x}\left(t_f\right), t_f\right) \quad (2.30)$$

The time derivative of the costate is equal to the negative partial of the Hamiltonian with respect to the state vector $\mathbf{x}$

$$\dot{\boldsymbol{\lambda}}(t) = -H_{\mathbf{x}} = -\begin{bmatrix} \frac{\partial H}{\partial x_1} \\ \frac{\partial H}{\partial y 1} \\ \frac{\partial H}{\partial \phi_1} \\ \frac{\partial H}{\partial x_2} \\ \frac{\partial H}{\partial y_2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \lambda_1 v_1 \sin\phi_1(t) - \lambda_2 v_1 \cos\phi_1(t) \\ 0 \\ 0 \end{bmatrix} \quad (2.31)$$

Because λ_1, λ_2, λ_4, and λ_5 have zero time derivatives, they will be constant terms throughout the game. The only time-varying costate is λ_3, which corresponds to the car's turn rate.

The costate terminal condition is

$$\boldsymbol{\lambda}^T(t_f) = \boldsymbol{\Phi}_{\mathbf{x}(t_f)} = \frac{\partial}{\partial \mathbf{x}} \phi\left(\mathbf{x}\left(t_f\right), t_f\right) + \alpha \frac{\partial}{\partial \mathbf{x}} l\left(\mathbf{x}\left(t_f\right), t_f\right) \quad (2.32)$$

$$\boldsymbol{\lambda}^T(t_f) = \alpha \begin{bmatrix} 2\left(x_1\left(t_f\right) - x_2\left(t_f\right)\right) \\ 2\left(y_1\left(t_f\right) - y_2\left(t_f\right)\right) \\ 0 \\ 2\left(x_2\left(t_f\right) - x_1\left(t_f\right)\right) \\ 2\left(y_2\left(t_f\right) - y_1\left(t_f\right)\right) \end{bmatrix} \tag{2.33}$$

The transversality condition for this free final time problem is

$$H\left(\mathbf{x}^*\left(t_f\right), u_1^*, u_2^*, \boldsymbol{\lambda}\left(t_f\right), t_f\right) = -\frac{\partial}{\partial t_f}\boldsymbol{\Phi} = -\frac{\partial}{\partial t_f}\phi - \alpha\frac{\partial}{\partial t_f}l\left(\mathbf{x}\left(t_f\right), t_f\right) = -1 \tag{2.34}$$

so that at the final time

$$\lambda_1 v_1 \cos\phi_1 + \lambda_2 v_1 \sin\phi_1 + \lambda_3(t_f)u_1 + \lambda_4 v_2 \cos u_2 + \lambda_5 v_2 \sin u_2 + 1 = 0 \tag{2.35}$$

where $\lambda_3(t_f) = 0$. The transversality condition is placed in the residual of the **bvp4c** solver.

Because of the constraint on u_1, Pontryagin's Minimum Principle must be applied instead of simply solving $H_{u_1} = 0$. The following rule for u_1 will minimize H:

$$u_1 = \arg\min_{u_1} H = \begin{cases} -u_{1max} & \lambda_3 > 0 \\ 0 & \lambda_3 = 0 \\ u_{1,max} & \lambda_3 < 0 \end{cases} \tag{2.36}$$

Because there are no constraints on u_2,

$$u_2 = \arg\max_{u_2} H = H_{u_2} = 0$$

The control u_2 can be written in terms of the costate for player two as

$$H_{u_2} = 0 = -\lambda_4 v_2 \sin u_2 + \lambda_5 v_2 \cos u_2 \tag{2.37}$$

$$u_2 = \arctan\frac{\lambda_5}{\lambda_4} \tag{2.38}$$

The dynamics of player two are rewritten in terms of the costate as

$$\dot{\mathbf{x}}_2 = \begin{bmatrix} v_2\cos u_2 \\ v_2 \sin u_2 \end{bmatrix} = \begin{bmatrix} v_2 \cos\left(\arctan\frac{\lambda_5}{\lambda_4}\right) \\ v_2 \sin\left(\arctan\frac{\lambda_5}{\lambda_4}\right) \end{bmatrix} = v_2 \begin{bmatrix} \frac{1}{\sqrt{1+\left(\frac{\lambda_5}{\lambda_4}\right)^2}} \\ \frac{\frac{\lambda_5}{\lambda_4}}{\sqrt{1+\left(\frac{\lambda_5}{\lambda_4}\right)^2}} \end{bmatrix} \tag{2.39}$$

As previously mentioned, λ_4 and λ_5 are constants, which means that the direction of travel for the evader will be constant throughout the game.

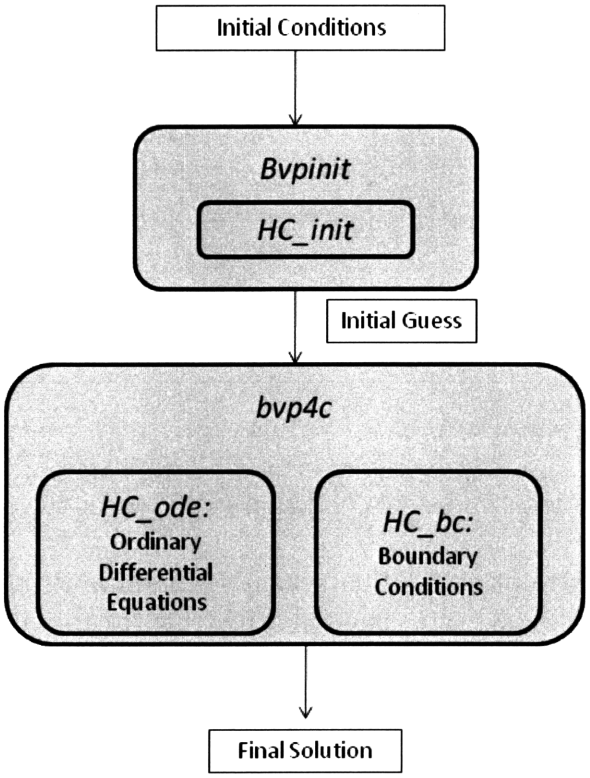

Figure 2-3: Homicidal Chauffeur `bvp4c` Implementation

2.3.3 Solution Implementation

The following initial conditions and parameters were used for the solution implementation:

$$\begin{array}{ll} x_{10} = 0 \text{ m} & y_{10} = 0 \text{ m} \\ x_{20} = 4 \text{ m} & y_{20} = 4 \text{ m} \\ \phi_{10} = 0 \text{ rad} & u_{1,max} = 1 \text{ rad/s} \\ v_1 = 3 \text{ m/s} & v_2 = 1 \text{ m/s} \\ d = 1 \text{ m} & \end{array}$$

Figure 2-3 shows the basic progression of the numerical solution for the homicidal chauffeur problem, which proceeds in the same manner as the target guarding problem in Section 2.2.3. The MATLAB code for this problem can be found in Appendix A.1.2.

2.3.4 Results

Figure 2-4 illustrates a successful implementation of the boundary value method in MATLAB for the homicidal chauffeur problem. Once again, in order to get the solution to converge properly, a good initial guess was required. The guess incorporated the information that the pursuer would initially be on a constrained arc, using the maximum turning rate possible until he was on a straight line trajectory with the pedestrian. If P had simple motion and could turn instantaneously, the pursuit would

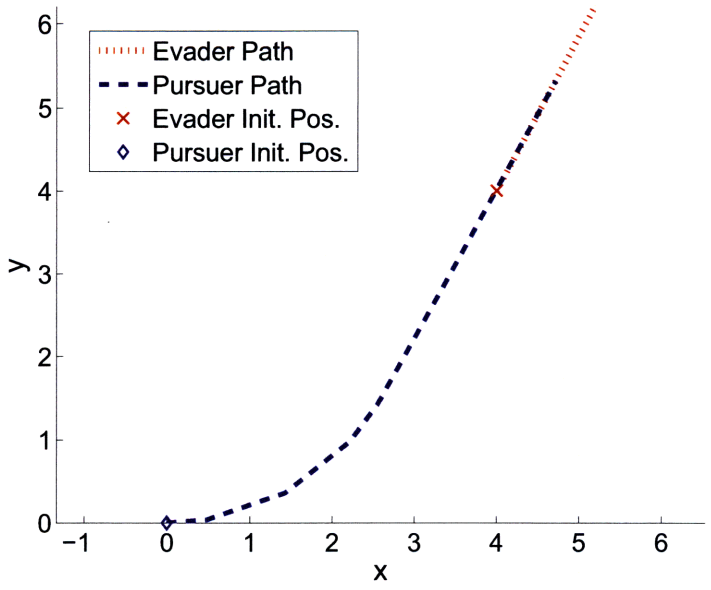

Figure 2-4: Homicidal Chauffeur Result

occur along a 45 degree angle line directly though the evader's starting position. As the car becomes less maneuverable, it benefits E to use steeper angles greater than 45 degrees.

2.4 Discussion

Solving differential game problems using boundary value methods presents a unique set of challenges. Indirect methods such as this one require the calculation of the analytical necessary conditions. However, as problems become dimensionally larger and include nonlinear terms, the calculation of the analytical necessary conditions becomes increasingly difficult. One means of making this process easier and reducing the number of errors would be to use an analytical differentiation tool that could take the derivatives of the Hamiltonian and capture conditions to form the necessary and boundary conditions.

Another primary difficulty of the boundary value method is providing the solver with an accurate initial guess. This difficultly primarily stems from the need for a guess of the costate λ and the Lagrange multiplier α. Some analysis is required to figure out what the sign of the costate should be and whether it should be constant, increasing, or decreasing. Choosing an α is even more difficult, as there is no obvious way to know what the value should be at the final time.

Continuation methods offer one way to aid the creation of the initial guess for more difficult initial conditions. The solution to the previous problem becomes the initial guess for the next problem with slightly different initial conditions. This technique allows the movement of the players' initial conditions to locations that could not pre-

viously be solved using an ad hoc initial guess method. Continuation could also allow the user to introduce increasingly tighter constraints on states and controls. Another possible method for initial guess creation is the use of a genetic algorithm to create the initial guess. This technique was explained by Conway [31] and implemented for his semi-DCNLP method for solving differential game problems.

Another issue with boundary value method is the determination of the control switching structure. In the target guarding problem, both players had constant costate terms so the game took place on a single, unconstrained arc that resulted in a straight line pursuit and capture. There were limits on the turning radius of the car in the homicidal chauffeur problem so the car began the game on a constrained arc, using the maximum turn rate possible, then switched off to zero turn rate for capture in a *bang-off* control scheme. In a realistic problem with multiple constrained control terms it would be difficult to know how the game should proceed and what form the control switching structure would ultimately have.

2.5 Conclusion

The boundary value method offers a straightforward way to solve differential game problems for those familiar with optimal control terminology and solution approaches. This methodical technique is a welcome departure from Isaacs' ad hoc approach for solving specific problems. However, even problems with easy to understand dynamics equations can become unmanageable during the calculation of the analytical necessary conditions. In addition, the boundary value approach requires an accurate guess for the costate trajectory which is often a non-intuitive quantity that requires knowledge of the control switching structure. Ultimately, the boundary value method is useful since it enables some actual differential game problems to be solved using built-in MATLAB functions. However, it is difficult to recommend this method for larger, non-linear problems without additional methods for developing initial guesses.

Chapter 3

Decomposition Method

This chapter explains how to approach differential game problems using a direct method that iteratively solves optimization problems for each player until convergence to a dual-optimal solution is achieved. Section 3.1 discusses the history of the decomposition method and provides a detailed description of the solution method. A MATLAB specific implementation of this method is provided in Section 3.2. Sections 3.3, 3.4, and 3.5 demonstrate the use of the decomposition method for the target guarding problem, the homicidal chauffeur problem, and the game of two cars.

3.1 Approach

Raivio proposed the decomposition method for solving differential game problems through a series of publications compiled into a doctoral dissertation [36]. This method has been applied to several different problems including the homicidal chauffeur problem [16], visual aircraft identification [29], and optimal missile guidance [28]. These works demonstrate how direct methods can solve differential game problems without the explicit computation of the necessary conditions.

The decomposition method for numerically solving differential game problems works by iteratively solving minimization and linearized maximization problems for the pursuer and the evader, respectively. If the iterations converge on a final solution, then this solution will satisfy the necessary conditions described in Equations 1.1-1.16 [16, p. 181]. References [16], [28], and [29] provide a detailed description of the solution process for the decomposition method.

Given the dynamics

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}, \mathbf{u}_1, \mathbf{u}_2, t\right) \tag{3.1}$$

for the game and the initial conditions $\mathbf{x}(0) = \mathbf{x}_0$, the controls $\mathbf{u}_1(t)$ and $\mathbf{u}_2(t)$ are found, which solve the minimax (or maximin) value problem

$$V = \min_{\mathbf{u}_1} \max_{\mathbf{u}_2} J = \max_{\mathbf{u}_2} \min_{\mathbf{u}_2} J \tag{3.2}$$

The cost functional is simply

$$J = \phi\left(\mathbf{x}\left(t_f\right), t_f\right) \tag{3.3}$$

because there is no integral cost in this formulation $\left(\int_{t_0}^{t_f} L(\mathbf{x}, \mathbf{u}_1, \mathbf{u}_2, t)\, dt = 0\right)$.

The first step of the iterative method is to find the trajectory of P that minimizes the cost functional

$$\min_{\mathbf{u}_1, t_f} \quad \phi\left(\mathbf{x}_1\left(t_f\right), \mathbf{x}_2^0\left(t_f\right), t_f\right) \tag{3.4}$$

given an arbitrary initial trajectory $\mathbf{x}_2^0(t)$ of E (the superscript describes the iteration count k beginning from $k = 0$). The terminal condition for this pursuit is

$$l\left(\mathbf{x}_1\left(t_f\right), \mathbf{x}_2\left(t_f\right), t_f\right) = 0 \tag{3.5}$$

The solution to the minimization results in the P trajectory $\mathbf{x}_1^0$ and the capture time t_f^0. Let the capture point be the evader's location at capture $\mathbf{e}^0 \equiv \mathbf{x}_2^0\left(t_f^0\right)$. The solution to the $\mathbf{x}_2^0$ trajectory, $\mathbf{x}_1^0$, can also be viewed as the solution to the problem given the fixed point $\mathbf{e}^0$ and fixed time t_f^0. The value function for P can be written as

$$\begin{aligned}\widetilde{V}\left(e, t_f\right) = \min_{u_1}\{\phi\left(\mathbf{x}_1\left(t_f\right), \mathbf{e}, t_f\right) \mid \dot{\mathbf{x}}_1\left(t\right) = f_1\left(\mathbf{x}_1(t), \mathbf{u}_1(t), t\right), t \in [0, t_f], \mathbf{x}_1(0) = \mathbf{x}_{10} \\ l\left(\mathbf{x}_1(t_f), \mathbf{e}^0, t_f\right) = 0\}\end{aligned} \tag{3.6}$$

such that

$$\widetilde{V}\left(\mathbf{e}^0, t_f^0\right) = \phi\left(\mathbf{x}_1^0\left(t_f^0\right), \mathbf{e}^0, t_f^0\right) \tag{3.7}$$

The objective for the evader is to maximize the pursuer's value function

$$\max_{u_2, t_f} \widetilde{V}\left(\mathbf{x}_2(t_f), t_f\right)$$

in the vicinity of $\left(\mathbf{e}^0, t_f^0\right)$ where

$$\dot{\mathbf{x}}_2(t) = f_2\left(\mathbf{x}_2(t), \mathbf{u}_2(t), t\right) \quad \text{and} \quad \mathbf{x}_2(0) = \mathbf{x}_{20} \tag{3.8}$$

Because there is no analytical expression for $\widetilde{V}$, the solution to the evader's problem above cannot be found directly. Instead, an approximate solution will be found by creating a linearized gradient for the value function in the neighborhood of $\left(\mathbf{e}^0, t_f^0\right)$. The solution for the maximization is then extended into the interval $\left[t_f^0, t_f^0 + \Delta t\right]$ where $\Delta t > 0$. The linear approximation for the value function about e^0 is

$$\widetilde{V}\left(\mathbf{e}^0, t_f\right) + \frac{\partial}{\partial \mathbf{e}} \widetilde{V}^T\left(\mathbf{e}^0, t_f^0\right)\left(\mathbf{e} - \mathbf{e}^0\right) \tag{3.9}$$

and the gradient of the value function is

$$\frac{\partial}{\partial \mathbf{e}} \widetilde{V}\left(\mathbf{e}^0, t_f\right) = \frac{\partial}{\partial \mathbf{e}} \phi\left(\mathbf{x}_1^0\left(t_f^0\right), \mathbf{e}^0, t_f\right) + \alpha^0 \frac{\partial}{\partial \mathbf{e}} l\left(\mathbf{x}_1^0\left(t_f\right), \mathbf{e}^0, t_f\right) \tag{3.10}$$

where α is the Lagrange multiplier that applies to the terminal constraint (the capture condition) $l(\mathbf{x}(t_f), t_f) = 0$. The gradient for the value function in Equation

3.10 can be found analytically thus avoiding numerical differentiation. Note that the gradient is composed of the partial derivatives of the cost function and the capture condition with respect to the capture point e^0.

Given the dynamics

$$\dot{\mathbf{x}}_2(t) = f_2\left(\mathbf{x}_2(t), \mathbf{u}_2(t), t\right) \quad \text{and} \quad \mathbf{x}_2(0) = \mathbf{x}_{20} \tag{3.11}$$

the maximization for E can be rewritten as

$$\max_{u_2, t_f} \; c^T\left(\mathbf{x}_2\left(t_f^0\right) - \mathbf{e}^0\right)$$

where

$$\mathbf{c} \equiv \frac{\partial}{\partial \mathbf{e}} \widetilde{V}\left(\mathbf{e}^0, t_f\right) = \frac{\partial}{\partial \mathbf{e}} \phi\left(\mathbf{x}_1^0\left(t_f^0\right), \mathbf{e}^0, t_f\right) + \alpha^0 \frac{\partial}{\partial e} l\left(\mathbf{x}_1^0\left(t_f\right), \mathbf{e}^0, t_f\right) \tag{3.12}$$

The solution $\mathbf{x}_2^1(t)$, $t \in \left[0, t_f^0\right]$ is extended onto the interval $\left[t_f^0, t_f^0 + \Delta t\right]$, $\Delta t > 0$ by the linear approximation

$$\mathbf{x}_2^1\left(t_f^0 + h\right) = \mathbf{x}_2^1\left(t_f\right) + \dot{\mathbf{x}}_2^1\left(t_f\right) h \quad , \quad h \in [0, \Delta t_f] \tag{3.13}$$

The extended solution x_2^1 becomes the trajectory that is used during the next minimization of the pursuer's trajectory. The minimization results in a new capture point e^1 that is used during the next linearized maximization step for E. The process continues until successive iterations converge no longer change the final value of the game within a user defined tolerance ϵ.

3.2 MATLAB Implementation

The solution procedure outlined below describes a possible implementation of the decomposition method using MATLAB:

- Begin with a set of initial conditions for P and E.
- Create an initial trajectory, $\mathbf{x}_2^0\left(\mathbf{u}_2^0(t), t\right)$, for E.
- Create an initial guess for P's control trajectory $\mathbf{u}_1^0(t)$ and a guess for the terminal time t_f which will be used by the `fmincon` solver in MATLAB.
- Let $k = 0$ and `tol` $> \epsilon$
- While `tol` $> \epsilon$
  - Solve P minimization using the `fmincon` function. The solver minimizes the cost function by manipulating the pursuer's controls and the intercept time found in the y-vector.

```
>> [y, ..., alpha] = fmincon(@cost_fun, y_guess, ..., @mycon)
```

* The solution is `y` = $\left[\ \mathbf{u}_1^{k+1}(\tau)\ \ \mathbf{u}_1^{k+1}(2\tau)\ \ \ldots\ \ \mathbf{u}_1^{k+1}(N\tau)\ \ t_f^{k+1}\ \right]^T$ where $\tau = \frac{t_f^{k+1}}{N-1}$
* `alpha` is the Lagrange multiplier found by the solver that applies to the terminal constraint $l\left(\mathbf{x}_1(t_f), \mathbf{x}_2(t_f), t_f\right) = 0$.
* `@cost_fun` is a function handle that returns the scalar terminal cost of the game
* `y_guess` = $\left[\ \mathbf{u}_1^k(\tau)\ \ \mathbf{u}_1^k(2\tau)\ \ \ldots\ \ \mathbf{u}_1^k(N\tau)\ \ t_f^k\ \right]^T$
* `@mycon` contains equality and inequality constraints including boundary conditions (initial and final) for $\mathbf{u}_1$ and the terminal constraint (capture condition) on $\mathbf{x}_1(t_f)$.

– Calculate $\mathbf{e}^{k+1}$, which is the location of the evader at the final time t_f^{k+1}

– Calculate $\mathbf{c} \equiv \frac{\partial \phi}{\partial \mathbf{e}} + \alpha^{k+1}\frac{\partial l}{\partial \mathbf{e}}$ using terminal state and time information

– Solve E maximization using `fmincon` function

```
>> u2 = fmincon(@val_fun,u2_p,...,@mycon2)
```

* `u2` = $\left[\ \mathbf{u}_2^{k+1}(\tau)\ \ \mathbf{u}_2^{k+1}(2\tau)\ \ \ldots\ \ \mathbf{u}_2^{k+1}(N\tau)\ \right]$
* `@val_fun` is the scalar function that represents that linearized gradient for the value of the game: $c^T\left(\mathbf{x}_2\left(t_f^{k+1}\right) - e^{k+1}\right)$
* `u2_p` = $\left[\ \mathbf{u}_2^k(\tau)\ \ \mathbf{u}_2^k(2\tau)\ \ \ldots\ \ \mathbf{u}_2^k(N\tau)\ \right]$
* `@mycon2` is contains the constraints and boundary conditions for the control

– Set the tolerance `tol` = $norm\left(\mathbf{u}_2^k - \mathbf{u}_2^{k+1}\right)$ and $k = k + 1$

– $\mathbf{u}_2^{k+1}$ is now the basis from which the next P minimization step is made.

The while loop continues until the control trajectory for the evader converges to a final solution within a user defined tolerance. The procedure above thus demonstrates how to turn a differential game problem into a MATLAB solvable iterative optimization problem for each player.

3.3 Target Guarding Problem

This section will examine the target guarding problem and how it can be solved using the decomposition method. As before, the problem dynamics can be written as

$$\dot{\mathbf{x}} = \begin{bmatrix} v_1 \cos u_1 \\ v_1 \sin u_1 \\ v_2 \cos u_2 \\ v_2 \sin u_2 \end{bmatrix} \tag{3.14}$$

with initial conditions

$$\mathbf{x}(t_0) = \begin{bmatrix} x_1(t_0) \\ y_1(t_0) \\ x_2(t_0) \\ x_2(t_0) \end{bmatrix} = \begin{bmatrix} x_{10} \\ y_{10} \\ x_{20} \\ y_{20} \end{bmatrix} \tag{3.15}$$

where x and y refer to the location of each player on the coordinate frame. Because both players are assumed to have simple motion, there are no constraints on the controls. In this problem the capture condition is the only terminal constraint, given by

$$\Psi(\mathbf{x}_f, t_f) = l(\mathbf{x}(t_f), t_f) = (x_1(t_f) - x_2(t_f))^2 + (y_1(t_f) - y_2(t_f))^2 - d^2 = 0 \tag{3.16}$$

The game concludes when the distance between the players is equal to the capture radius d. The objective of the game depends only on the final state of the players, thus $L(\mathbf{x}, u_1, u_2, t) = 0$, and

$$J = \phi(\mathbf{x}(t_f), t_f) = \sqrt{(x_2(t_f) - x_C)^2 + (y_2(t_f) - y_C)^2} \tag{3.17}$$

where x_C and y_C are the coordinates for the target being guarded. Each of the problems below had the following initial conditions:

$$\begin{array}{ll} x_{10} = 5 \text{ m} & x_{20} = 0 \text{ m} \\ y_{10} = 0 \text{ m} & y_{20} = -0.5 \text{ m} \\ v_1 = 1 \text{ m/s} & v_2 = 1 \text{ m/s} \end{array}$$

Figure 3-1 illustrates the progression of the solution. The `TG_init` function takes in an initial guess for where the optimal aim-point for the two players might be and then generates a control and state trajectory based on that aim-point. The $\mathbf{u}_1^0$ trajectory and t_f^0 are sent as the initial guess for the solution to the P minimization. The minimization takes place with respect to the $\mathbf{x}_2^0(t)$ trajectory. By convention, differential games are written such that P is minimizing the cost functional and E is maximizing it. Since P in this game wants the intercept to happen as far as possible away from the target, `fmincon` minimizes the *negative* distance of E from the target at capture.

Convergence was a significant difficulty with the target guarding problem. For a given set of initial conditions for P and E, convergence was heavily dependent upon the location of the target, C, represented in the figure as an X. Figure 3-2 shows how the iterations jumped back and forth across the actual solution denoted with a circle. The arrow at the intercept point shows the direction of the linearized value function $\mathbf{c} \equiv \frac{\partial \phi}{\partial \mathbf{e}} + \alpha \frac{\partial l}{\partial \mathbf{e}}$. With the target at position (10,10) even slight adjustments of the evader's control direction have a large impact on the position at the final time. However, when the target is at location (15,5) as shown in Figure 3-3 the optimal aim-point is closer to the evader's initial position and convergence is achieved. As the distance between the evader's initial position and the optimal aim-point increases, changes to the control angle have a greater effect on the final outcome. Table 3.1 shows the initial conditions for both the convergent and non-convergent cases.

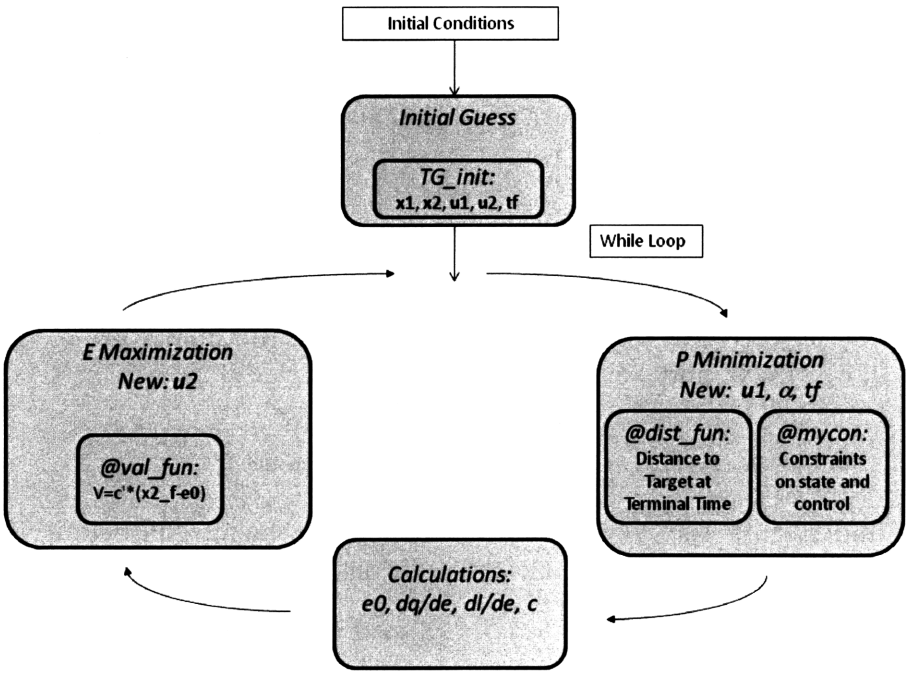

Figure 3-1: Decomposition Method for Target Guarding Problem

Table 3.1: Convergent and non-convergent cases for the target guarding problem

	Target position $(x,\ y)$	Optimal aim-point $(x,\ y)$	$\mathbf{x}_2(0)$ distance to target	$\mathbf{x}_2(0)$ distance to optimal aim-point
Non-convergent	(10,10)	(1.61,9.16)	14.5	9.79
Convergent	(15,5)	(2.15,3.72)	15.98	4.73

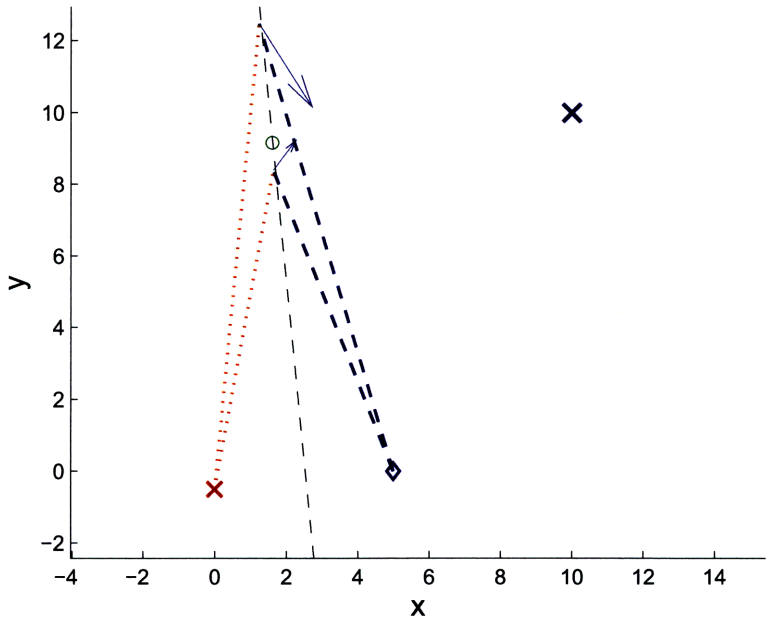

Figure 3-2: Non-Convergent Target Guarding Problem: Target Location (10,10)

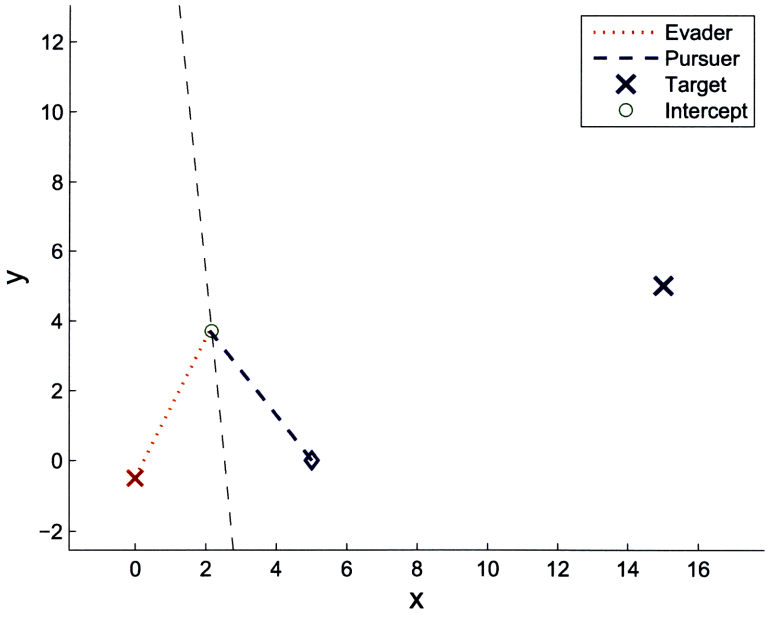

Figure 3-3: Convergent Target Guarding Problem: Target Location (15,5)

3.4 Homicidal Chauffeur Problem

As before, the equations of motion for P and E are

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{\phi}_1 \\ \dot{x}_2 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} v_1 \cos\phi_1 \\ v_1 \sin\phi_1 \\ u_1 \\ v_2 \cos u_2 \\ v_2 \sin u_2 \end{bmatrix} \tag{3.18}$$

with initial conditions

$$\mathbf{x}(t_0) = \begin{bmatrix} x_1(t_0) \\ y_1(t_0) \\ \phi_1(t_0) \\ x_2(t_0) \\ x_2(t_0) \end{bmatrix} = \begin{bmatrix} x_{10} \\ y_{10} \\ \phi_{10} \\ x_{20} \\ y_{20} \end{bmatrix} \tag{3.19}$$

where x and y refer to the location of each player on the coordinate frame and ϕ_1 is the pursuer's heading angle relative to the x-axis.

While the pedestrian E can change his direction instantaneously, the car P has a limited turn rate. Thus, a constraint is placed on the absolute angular velocity of player 1,

$$|u_1| \leq u_{max} \tag{3.20}$$

The capture condition

$$\Psi(\mathbf{x}_f, t_f) = l(\mathbf{x}(t_f), t_f) = (x_1(t_f) - x_2(t_f))^2 + (y_1(t_f) - y_2(t_f))^2 - d^2 = 0 \tag{3.21}$$

is the only terminal constraint. The game concludes when the distance between the car and the pedestrian is equal to the capture radius d. The objective of the game depends only on the final state of the players, thus $L(\mathbf{x},u_1,u_2,t) = 0$ and

$$J = \phi(\mathbf{x}(t_f), t_f) = t_f \tag{3.22}$$

The scenarios below each had the following initial conditions, which were also used in Raivio's work [16]. Using these parameters will provide a basis by which to validate the MATLAB approach for solving problems using the decomposition method.

$$\begin{array}{ll} x_{10} = 0 \text{ m} & y_{10} = 0 \text{ m} \\ \phi_{10} = 0 \text{ rad} & u_{1,max} = 1 \text{ rad/s} \\ v_1 = 3 \text{ m/s} & v_2 = 1 \text{ m/s} \\ d = 1 \text{ m} & \end{array}$$

Several different scenarios of the homicidal chauffeur problem were solved using the decomposition method. Each test case was chosen to demonstrate the method's ability to solve problems that interact with *dispersal surfaces*. The dispersal and barrier surfaces for the homicidal chauffeur game are described in [6]. Case I, Figure 3-4, is the simplest with E in front of P. In case II, Figure 3-5, E is behind P and must

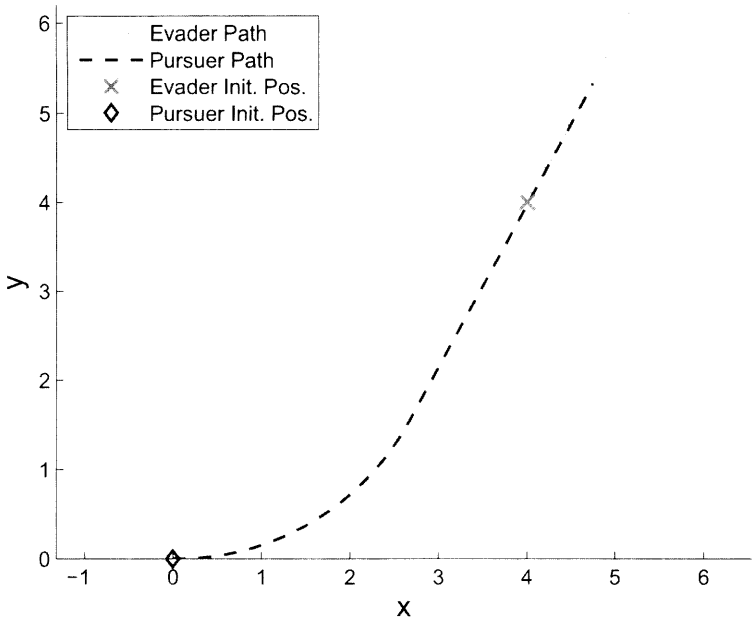

Figure 3-4: Homicidal Chauffeur with Decomposition Method: Case I

cross a dispersal surface during the pursuit. Case III, Figure 3-6, solves a problem where the initial conditions are near the barrier. In case IV, Figure 3-7, the game takes place on the other side of the barrier. The final results and comparisons to solutions found in [16] can be found in Table 3.2.

As expected, the solutions fall on a universal surface resulting in a tail-chase scenario. In case II, the mirror image solution would also be acceptable. The initial guess determines which of the mirror solutions is returned by the solver. It is an *instantaneous mixed* problem in which there is more than one optimal solution at the starting point of the game [6, p. 136]. These dilemmas can be solved by forcing one player to make an arbitrary selection if two or more equally good solutions present themselves.

One limitation of the decomposition method appears in case IV both in this MATLAB simulation and in [16]. It would actually behoove the pedestrian, E, to move towards the car and force the vehicle to drive away and turn around before completing the game in a tail chase scenario. As shown the game converges to a maxmin solution in which E does not play optimally. As currently formulated, the decomposition method will not solve for *equivocal surfaces* whose tributaries are not solutions of the maxmin problem [16, p. 184].

3.5 Game of Two Cars

The game of two cars [37] is similar to the homicidal chauffeur problem; however, the pedestrian is replaced with another car that is also restricted in turn rate. In

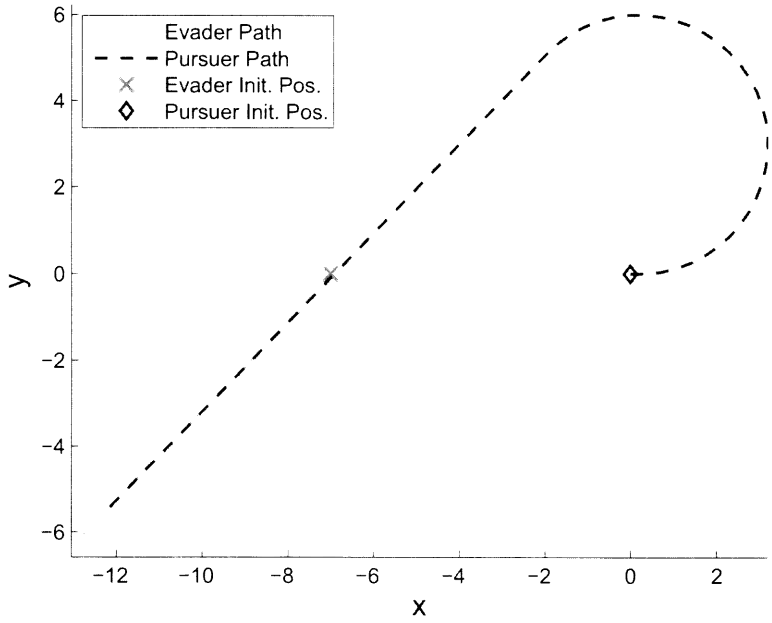

Figure 3-5: Homicidal Chauffeur with Decomposition Method: Case II

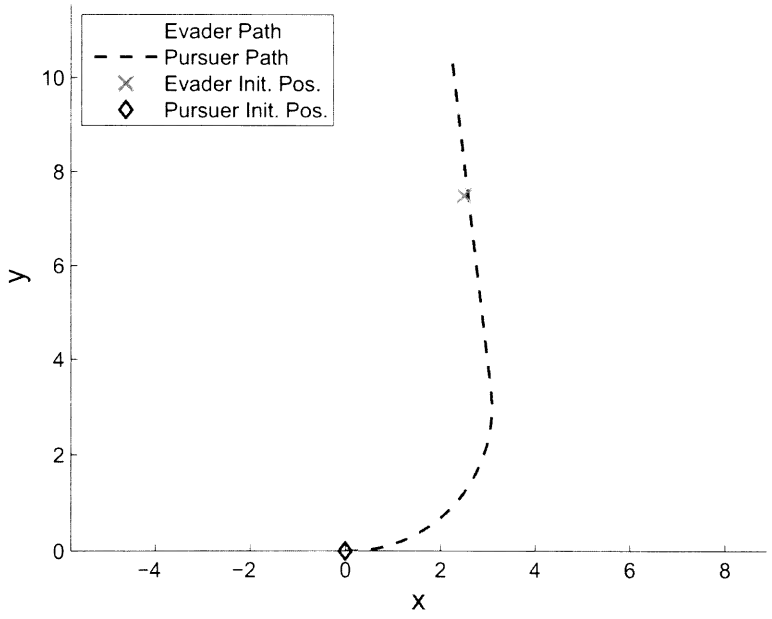

Figure 3-6: Homicidal Chauffeur with Decomposition Method: Case III

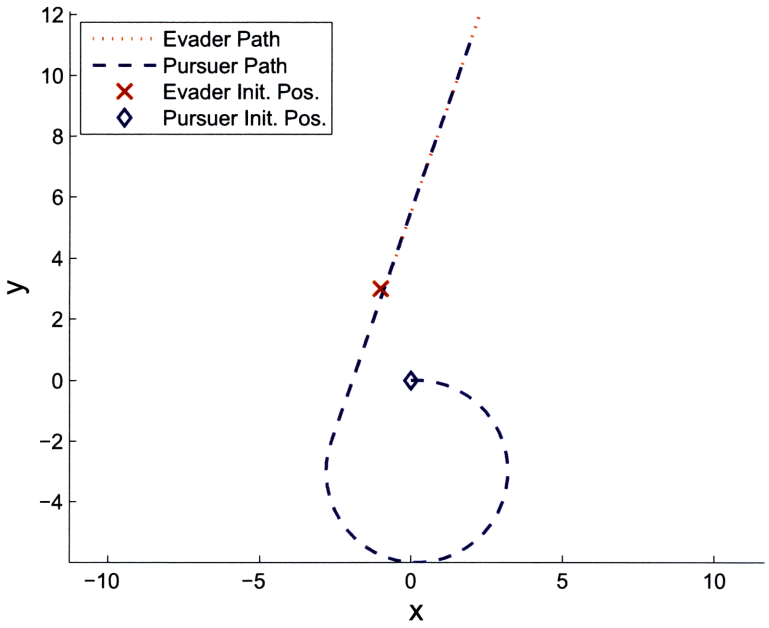

Figure 3-7: Homicidal Chauffeur with Decomposition Method: Case IV

Table 3.2: Homicidal chauffeur problem solved by the decomposition method

Case	I	II	III	IV
(x_{20}, y_{20})	(4,4)	(-7,0)	(2.5,7.5)	(-1,3)
Iterations	5	10	6	7
Terminal Time (s)	2.508	8.98	4.14	9.76

Isaacs' treatment of the problem [6, p. 237], the state space is reduced from six states to three by using a relative coordinate frame. The boundary of the usable part is defined after a series of analytical calculations. The decomposition method does not require the state reduction and can be used to find the actual solutions to problems without determining the boundaries and barriers that exist in the state space.

The equations of motion for the pursuer and the evader are

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{\phi}_1 \\ \dot{x}_2 \\ \dot{y}_2 \\ \dot{\phi}_2 \end{bmatrix} = \begin{bmatrix} v_1 \cos\phi_1 \\ v_1 \sin\phi_1 \\ u_1 \\ v_2 \cos\phi_2 \\ v_2 \sin\phi_2 \\ u_2 \end{bmatrix} \tag{3.23}$$

with initial conditions

$$\mathbf{x}(t_0) = \begin{bmatrix} x_1(t_0) \\ y_1(t_0) \\ \phi_1(t_0) \\ x_2(t_0) \\ x_2(t_0) \\ \phi_2(t_0) \end{bmatrix} = \begin{bmatrix} x_{10} \\ y_{10} \\ \phi_{10} \\ x_{20} \\ y_{20} \\ \phi_{20} \end{bmatrix} \tag{3.24}$$

where x and y refer to the location of each player on the coordinate frame and ϕ_1 is the pursuer's heading angle and ϕ_2 is the evader's heading angle relative to the x-axis. Both cars have a limited turn rate. Thus, a constraint is placed on the absolute angular velocity of each car

$$|u_1| \leq u_{1,max} \tag{3.25}$$

$$|u_2| \leq u_{2,max} \tag{3.26}$$

Like the homicidal chauffeur problem, the capture condition is the only terminal constraint

$$\Psi(\mathbf{x}_f, t_f) = l(\mathbf{x}(t_f), t_f) = (x_1(t_f) - x_2(t_f))^2 + (y_1(t_f) - y_2(t_f))^2 - d^2 = 0 \tag{3.27}$$

The game concludes when the distance between the cars is equal to the capture radius d. The objective of the game depends only on the final state of the players, thus $L(\mathbf{x}, u_1, u_2, t) = 0$ and

$$J = \phi(\mathbf{x}(t_f), t_f) = t_f \tag{3.28}$$

The following initial conditions and parameters were used in both cases and case specific conditions can be found in Table 3.3.

$$\begin{array}{ll} x_{10} = 0 \text{ m} & x_{20} = 4 \text{ m} \\ y_{10} = 0 \text{ m} & y_{20} = 4 \text{ m} \\ d = 1 \text{ m} & \end{array}$$

The solution was obtained in the same manner as previous problems. A limit on the turning rate of the evader was added to the constraint function of the E

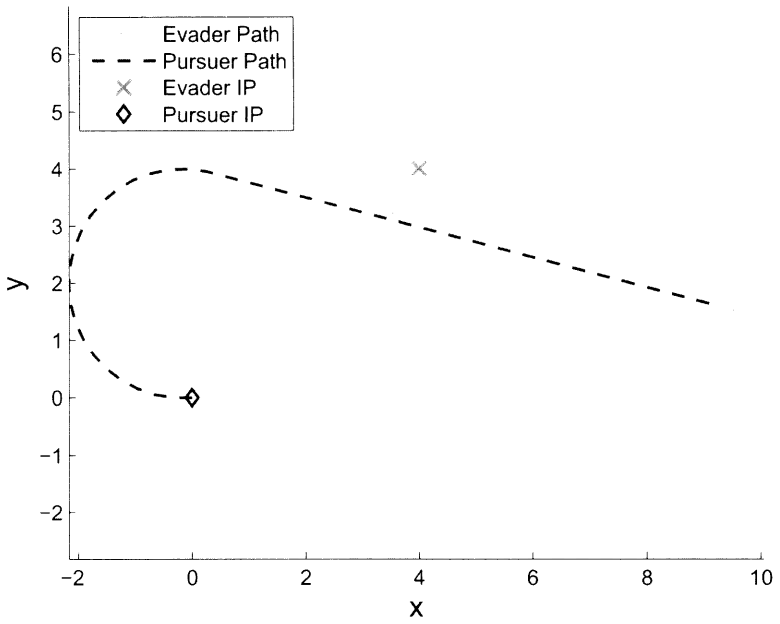

Figure 3-8: Game of Two Cars with Decomposition Method: Case I

maximization. Figures 3-8 and 3-9 show the solutions to two different two-car scenarios. In both cases, P and E use their maximum turning abilities until they reach their final heading direction. This control switching structure is the same *bang-off* pattern observed for P during the homicidal chauffeur scenario. The two car game demonstrates how easily the decomposition method can be adapted to solve problems without having to endure the rigorous mathematics required by Isaacs' approach.

Table 3.3: Two cars problem solved by the decomposition method

Case	I	II
ϕ_{10} (rad)	π	$-\frac{2\pi}{3}$
ϕ_{20} (rad)	$-\pi$	$-\frac{2\pi}{3}$
v_1 (m/s)	2	3
v_2 (m/s)	1	1
$u_{1,max}$ (rad/s)	1	0.66
$u_{2,max}$ (rad/s)	2	1.25
Iterations	15	13
Terminal Time (s)	8.00	9.848

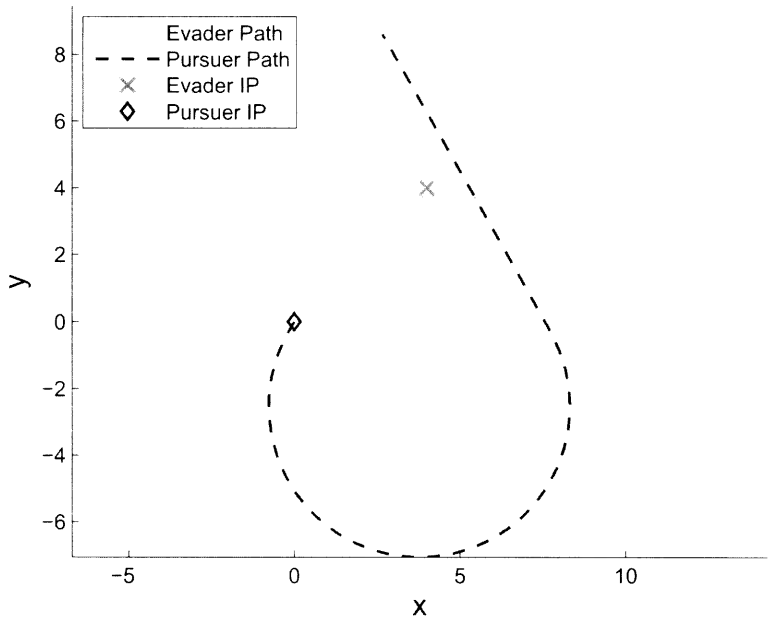

Figure 3-9: Game of Two Cars with Decomposition Method: Case II

3.6 Conclusion

While the decomposition method demonstrated an ability to solve a variety of problems, the issue of convergence in the target guarding problem is an area of concern. It may be possible that with some scaling and adjustment of the solver tolerances that the optimal solution could be found for any initial placement of the players and the target. The fact that the initial conditions had such an influence on convergence is a significant source of concern regarding the viability of the decomposition method for solving more difficult problems with realistic dynamics such as the submarine problem where variable speed and realistic thrust and drag profiles might be used.

Another issue this method shares with the boundary value method is that the final solution of the game does not result in a control policy. The assumption is that both players act optimally. Thus, the decomposition method is really an open loop solution that applies only to the specific initial conditions given for one problem. It would be useful to have the closed loop policy for each player that could then be applied to a variety of initial conditions in the state space. Though not a requirement for the stated goal of parametric assessment of game scenarios, understanding the control policy could aid the future development of real-time guidance laws for the security ship guarding the submarine that would have to handle incomplete, noisy information about the state of the game.

Chapter 4

Neural Networks for Differential Games

This chapter explains how neural networks can be used to govern the control policies for each player in a differential game. Section 4.1 provides insights into the solution approach including a basic overview of neural networks and an introduction to the TAPENADE Automatic Differentiation engine. In Section 4.2 the viability of the neural network method is illustrated through the solution of several optimal control problems. Finally, Section 4.3 demonstrates the use of neural networks to solve differential game problems using MATLAB.

4.1 Solution Approach

The control policy for each player is formed by manipulating the weights and biases of a neural network. The input to the network is the current state of the game, including exogenous inputs if they exist, and the output is a control signal. MATLAB's `fminunc` solver uses gradient information provided by an automatically generated adjoint code to adjust the weights and biases of the neural network to minimize the value or cost of the game. Once the first player's strategy is established, the second player's weights are optimized against that strategy. The first player then optimizes to the second player's new strategy, and the process continues until both players arrive at an optimal set of weights. The end result is the min-max or saddle-point solution in which one player minimizes the cost functional while the other player maximizes it.

4.1.1 Neural Networks

A basic overview of neural networks will be provided before describing the differential game solution method in detail. More comprehensive information can be found in [38] and [39]. The most basic unit of a feed-forward artificial neural network is the single input neuron as shown in Figure 4-1[1]. An input p is scaled by the neuron weight w and shifted by the neuron bias b. The net result $n = wp + b$ becomes the

[1]Neural network Figures 4-1 through 4-4 adapted from [39]

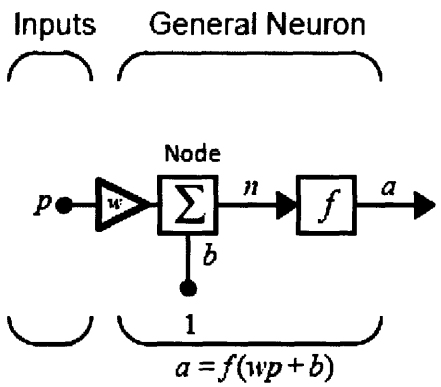

Figure 4-1: Single-Input Neuron

input to the transfer function f that has the scalar output a. The transfer function can be linear or non-linear. One popular choice is the log-sigmoid transfer function that scales the output between 0 and 1. The majority of the simulations conducted for this thesis use a scaled inverse tangent function $a = \frac{2}{\pi}\tan^{-1}(n)$ which results in an output range $-1 < a < 1$. One benefit provided by these functions is that they are differentiable. Transfer function differentiability enables the analytical calculation of gradient information, which will later be used during the solution of optimal control problems.

To form a one layer neural network, a set of R inputs connects to a layer of S neurons as shown in Figure 4-2 and in an abbreviated form in Figure 4-3. Each input connects to each node, and there is a corresponding weight associated with every connection. Most neural networks have more than one layer. With each added layer, the output of the previous layer acts as the input to the next layer as shown in the three-layer network in Figure 4-4. For control problems, the inputs to the first layer are the relevant states assuming there are no external inputs to the system. The hidden layers, the layers in between the input and output, have a user-selected number of nodes that vary according to the complexity of the problem. The output layer contains the control signal that is scaled according to the constraints on the controller and the type of node transfer function used.

4.1.2 Automatic Differentiation

Another feature of the approach outlined below is the use of an automatic differentiation (AD) engine. The TAPENADE AD software works line by line through FORTRAN code to create an adjoint code that contains analytical expressions for the desired derivatives or gradients. Adjoint methods are less computationally intensive and more accurate than finite difference methods, which are susceptible to round-off errors. This gradient information can be directly used by an optimization tool such as MATLAB's `fminunc` function. More information about AD can be found in [40, 41, 42].

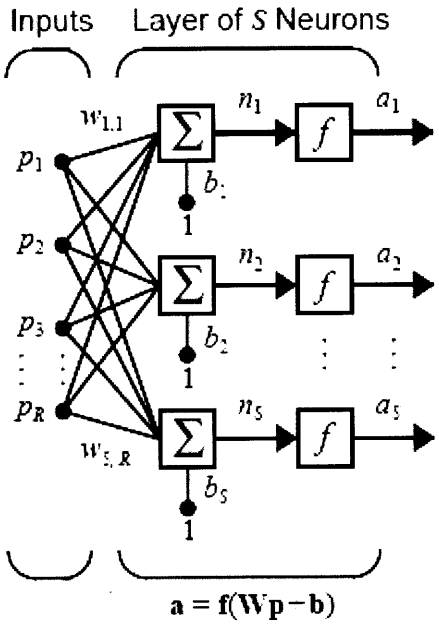

Figure 4-2: Layer of Neurons

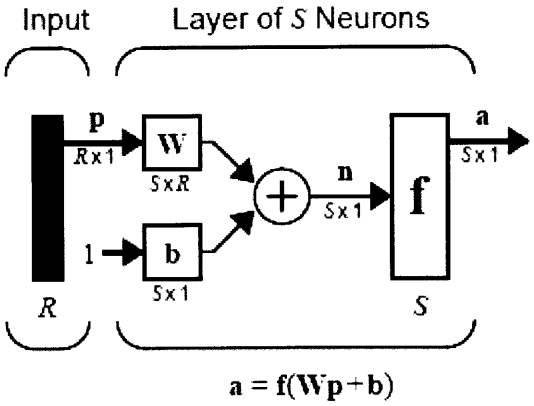

Figure 4-3: Layer of Neurons, Abbreviated Notation

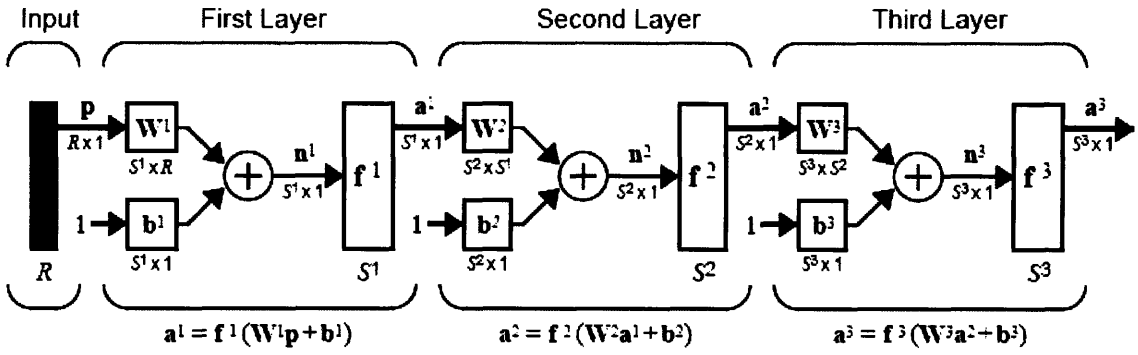

Figure 4-4: Three-Layer Neural Network

4.1.3 Method

While artificial neural networks have numerous uses, this thesis focuses on their ability to act as controllers. Typical control architectures for neural nets apply them as universal function or plant approximators that are inverted and used to develop a control solution [39]. Another technique trains neural networks on a set of known optimal trajectories in an attempt to forge a guidance law that applies to other locations within the state-space of the training set [43]. The approach outlined below assumes known plant dynamics and uses the neural net as the controller. The inputs to the neural net are the relevant plant states and the output is the control signal. The output of the network for a given input depends on the weights and biases established for each node.[2] The weights define the control policy that determines how the player will respond given the current state of the game.

The solution for differential game problems requires the computation of optimal control policies for each player. To find the optimal policies, an arbitrary control policy is first set for the evader, E. The pursuer, P, initializes the neural net control policy to a random, normal distribution of weights. The plant dynamics, neural net controller, cost functions, and Runge-Kutta integration algorithm are all combined into one FORTRAN main function as shown in Figure 4-5. The inputs to this function are the initial state conditions, the neural net weights, and a time vector. The outputs are the state trajectory, control trajectory, and final cost. Additionally, the FORTRAN main function is analytically differentiated line by line using the TAPENADE Automatic Differentiation Engine [41]. TAPENADE creates an adjoint main function whose output is the cost J and the cost gradient $\frac{\partial J}{\partial \mathbf{w}}$ also shown in Figure 4-5. The cost gradient output provides the sensitivity of the final cost to each weight. Both the main function and adjoint main function are compiled into MATLAB MEX-files, which enables them to be called from MATLAB as if they were native functions.

MATLAB's `fminunc` solver minimizes the cost output of the MEX-function by adjusting the weights as shown in Figure 4-6. The solver begins with a random, normal distribution of weights, determines the cost, changes the weights according to the direction indicated by the cost gradient, then runs the simulation again with the

[2]The generic term *weights* will be used to describe both node connection weights and node biases.

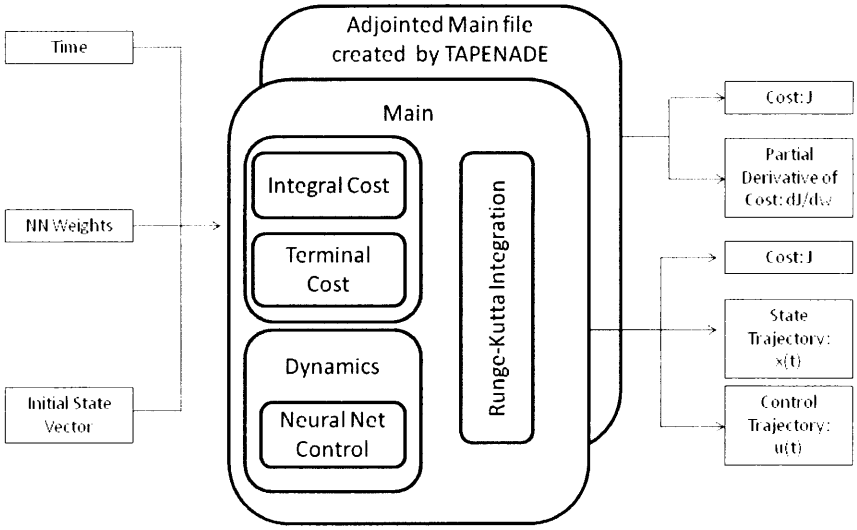

Figure 4-5: Main MEX-File

new weights. This process is repeated until adjustment of the weights can no longer reduce the final cost. With the pursuer's neural net weights established, the process to find the evader's optimal weights begins.

As before, the weights for the control policy of E are initialized with a random normal distribution. The weights for the control policy of P remain unchanged while `fminunc` adjusts the weights for E. Once `fminunc` produces a new set of weights, the cycle repeats. This time, however, `fminunc` uses the previous weights for P rather than a new normal distribution. The weight optimizations continue until both P and E have optimal policies.

Because the TAPENADE AD engine requires FORTRAN77, FORTRAN95, or C code, it was necessary to create an interface between MATLAB and the FORTRAN main files. Gnumex [44] facilitated the creation of the MEX-files from FORTRAN95 code compiled by GFortran [45]. Once created, the MEX-file operates like any other MATLAB function.

4.2 Optimal Control Problems

Before attempting a differential game problem, several one-player optimal control problems were examined in order to determine how the neural network control policies compare to the known optimal solutions.

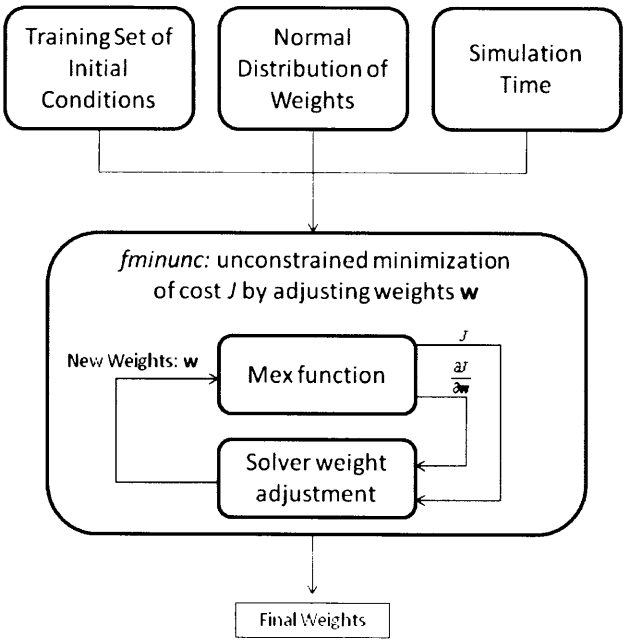

Figure 4-6: Neural Net Optimal Control Solution Approach

4.2.1 1D Optimal Problem

The first problem is a one-dimensional constrained optimal control problem [46]. The goal is to design the control inputs that minimize the following cost functional.

$$\min_u J = -x(t_f) + \int_0^{t_f} u^2(t)\, dt \tag{4.1}$$

where $\dot{x} = x + u$, $x(0) = 0$, $u(t) \leq 5$, and $t_f = 4$. Using a traditional optimal control approach of analytically deriving the necessary and boundary conditions. The optimal control strategy

$$u(t) = \begin{cases} 5 & t \leq t_c \\ \frac{1}{2}e^{4-t} & t \geq t_c \end{cases} \tag{4.2}$$

is found, which results in the state trajectory

$$x(t) = \begin{cases} 5e^t - 5 & t \leq t_c \\ -\frac{1}{4}e^{4-t} + e^t(5 - 25e^{-4}) & t \geq t_c \end{cases} \tag{4.3}$$

where $t_c = 4 - \ln(10)$.

The neural network was set up with a single input (the state x), two hidden layers, and a single output (the control u). The $\tan^{-1}$ function was used for all the neurons. The initial weights were a random, normal distribution scaled by 0.10. Figure 4-7 illustrates the agreement of the optimal control and neural network approaches.

While the neural network method usually resulted in the optimal solution, on

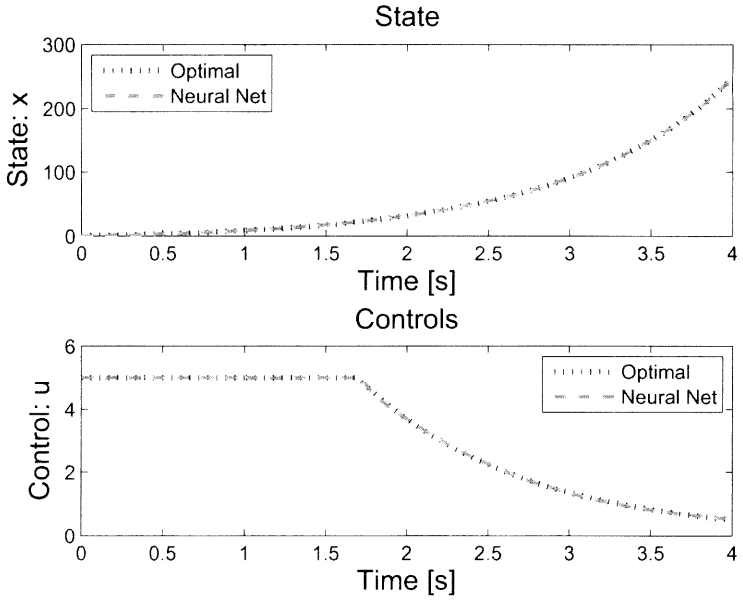

Figure 4-7: 1D Optimal Problem

occasion the solver converged to a sub-optimal solution as shown in Figure 4-8. In the sub-optimal cases the control command typically remained constant throughout the entire simulation. Convergence to the optimal solution was dependent upon the initial vector of random weights. Several initial weight vectors should be used before deeming any solution optimal, as it is possible that the solver did not converge to the global minimum cost for a particular problem.

4.2.2 Missile Intercept Problem

In his popular work on missile interceptor guidance, Zarchan discusses how a lag in the missile guidance system results in miss distance between the interceptor and the target at the final time when using proportional navigation and augmented proportional navigation [47, p. 155]. The interceptor guidance system dynamics are represented by a single lag

$$\frac{n_L}{n_c} = \frac{1}{1+sT} \tag{4.4}$$

where n_L is the achieved acceleration, n_c is the commanded acceleration, and T is the guidance system time constant. The dynamics can be written in state-space form as

$$\begin{bmatrix} \dot{y} \\ \ddot{y} \\ \dot{n}_T \\ \dot{n}_L \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{T} \end{bmatrix} \begin{bmatrix} y \\ \dot{y} \\ n_T \\ n_L \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{T} \end{bmatrix} \tag{4.5}$$

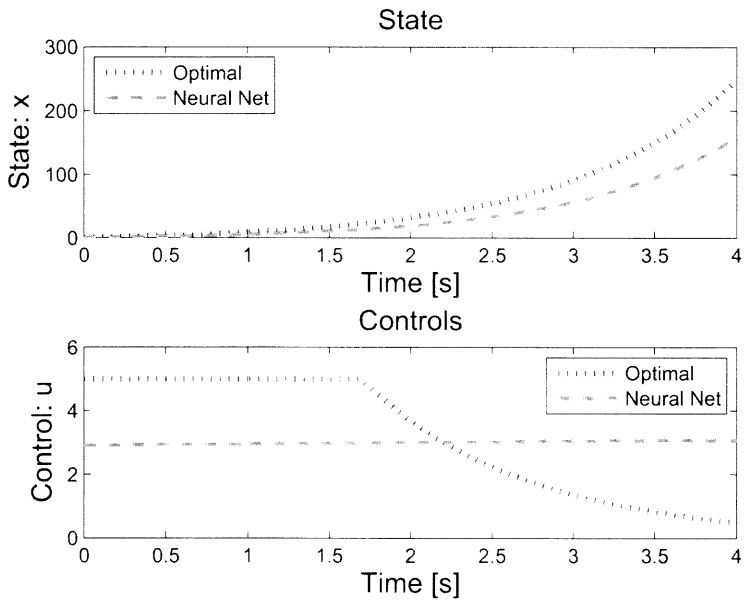

Figure 4-8: Sub-optimal Solution for 1D Optimal Problem

where n_T is the lateral target maneuver. The goal of the interceptor is to minimize the cost functional

$$J = y^2(t_f) + \gamma \int_0^{t_f} n_c^2 \, dt \tag{4.6}$$

where γ is a user-selected gain that weights the importance of the integral term. Smaller γ places more emphasis on miss distance at t_f.

Figure 4-9 shows how the performance of the neural net control policy was comparable to the optimal solution. However, this result was obtained after adding an additional penalty $\int_0^{t_f} y^2(t)\, dt$ for miss distance during the game. Without the extra penalty, the solver focused on zero miss at t_f with less consideration for miss distance through the rest of the game.

4.2.3 Orbit Raising Problem

Bryson and Ho present a problem regarding the maximum radius orbit transfer in a given time [13, p. 66][48]. In this problem thrust direction $\phi(t)$ is used to transfer a vehicle with a constant thrust rocket engine from an initial circular orbit to the largest possible circular orbit in a fixed amount of time. The cost functional is

$$J = r(t_f) \tag{4.7}$$

The following terms will be used:

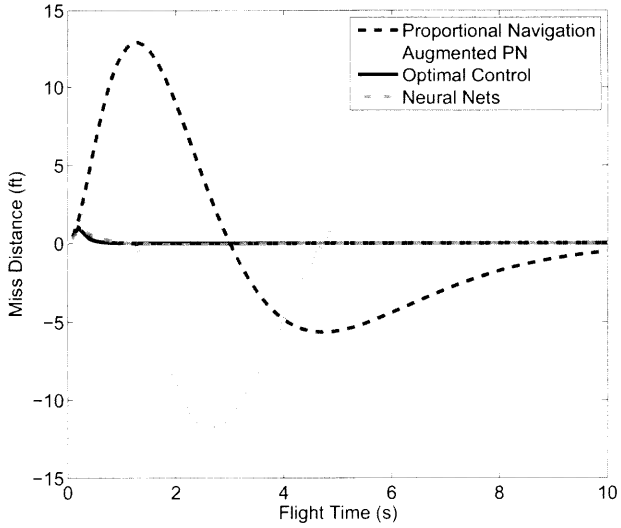

Figure 4-9: Missile Intercept Problem

r	=	radial distance of spacecraft from attracting center
u	=	radial component of velocity
v	=	tangential component of velocity
m	=	mass of spacecraft
$\dot{m}$	=	fuel consumption rate (constant)
ϕ	=	thrust direction angle
μ	=	gravitational constant of attracting center

The system has the dynamics

$$\dot{r} = u \tag{4.8}$$

$$\dot{u} = \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T \sin\phi}{m_0 - |\dot{m}|t} \tag{4.9}$$

$$\dot{v} = -\frac{uv}{r} + \frac{T \cos\phi}{m_0 - |\dot{m}|t} \tag{4.10}$$

with the initial conditions $r(0) = r_0$, $u(0) = 0$, and $v(0) = \sqrt{\frac{\mu}{r_0}}$ and the terminal conditions $u(t_f) = 0$ and $v(t_f) - \sqrt{\frac{\mu}{r(t_f)}} = 0$.

In order to enforce the terminal conditions, additional penalty terms were included in the terminal cost so that

$$J = r(t_f) - \alpha u^2(t_f) - \beta \left(v(t_f) - \sqrt{\frac{\mu}{r(t_f)}} \right) \tag{4.11}$$

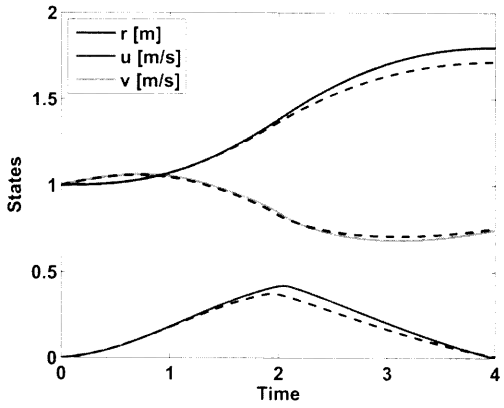

Figure 4-10: Orbit Raising States

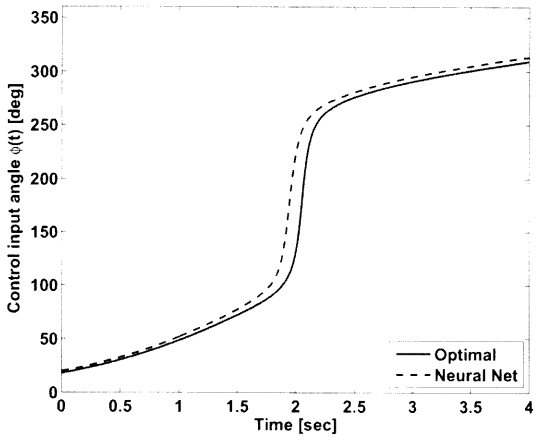

Figure 4-11: Orbit Raising Control

The terms α and β are user selected weights that scale the penalty for a violation of the terminal conditions during the maximization of the cost J.

Figures 4-10 and 4-11 demonstrate the ability of the neural network, represented by dashed lines, to provide a solution that is close to the optimal solution provided by more traditional optimal control methods. The final solution varied according to the number of nodes and layers, the size of the normal distribution of weights, and the magnitude of the terminal condition penalties. Adjusting these term or adding additional initial conditions to the training set could help the neural network controller achieve a closer correlation with the optimal solution.

4.3 Differential Game Problems

4.3.1 Homicidal Chauffeur

Once again the equations of motion for P and E are

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{\phi}_1 \\ \dot{x}_2 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} v_1 \cos\phi_1 \\ v_1 \sin\phi_1 \\ u_1 \\ v_2 \cos u_2 \\ v_2 \sin u_2 \end{bmatrix} \tag{4.12}$$

with initial conditions

$$\mathbf{x}(t_0) = \begin{bmatrix} x_1(t_0) \\ y_1(t_0) \\ \phi_1(t_0) \\ x_2(t_0) \\ x_2(t_0) \end{bmatrix} = \begin{bmatrix} x_{10} \\ y_{10} \\ \phi_{10} \\ x_{20} \\ y_{20} \end{bmatrix} \tag{4.13}$$

where x and y refer to the location of each player on the coordinate frame and ϕ_1 is the pursuer's heading angle relative to the x-axis.

While the pedestrian, E, can change direction instantaneously, the car P has a limited turn rate. Thus, a constraint

$$|u_1| \leq u_{max} \tag{4.14}$$

is placed on the absolute angular velocity of P. The capture condition

$$\boldsymbol{\Psi}(\mathbf{x}_f, t_f) = l(\mathbf{x}(t_f), t_f) = (x_1(t_f) - x_2(t_f))^2 + (y_1(t_f) - y_2(t_f))^2 - d^2 = 0 \tag{4.15}$$

is the only terminal constraint. The game concludes when the distance between the car and the pedestrian is equal to the capture radius, d. This game begins in the capture zone so P is guaranteed to catch E. The objective of the game of degree depends on the time of the game at capture thus,

$$J = \phi(\mathbf{x}(t_f), t_f) = t_f \tag{4.16}$$

As explained in Section 4.1.3, MATLAB's `fminunc` solver is used to find the optimal weights for each neural network control scheme. E is initially assigned a strategy to move away from P. The solver finds the optimal weights for P against this assigned strategy. Once the weights for P are established, another simulation series is run in which the pursuer's weights are held constant while the solver adjusts the evader's weights. The optimizations for each player continue until the solver converges to optimal strategies for both players. For the homicidal chauffeur scenario shown in Figure 4-12, the solution was found within five iterations.

A set of ten initial conditions was used to train the neural network for this problem. When evaluating a training set, MATLAB runs each initial condition through the

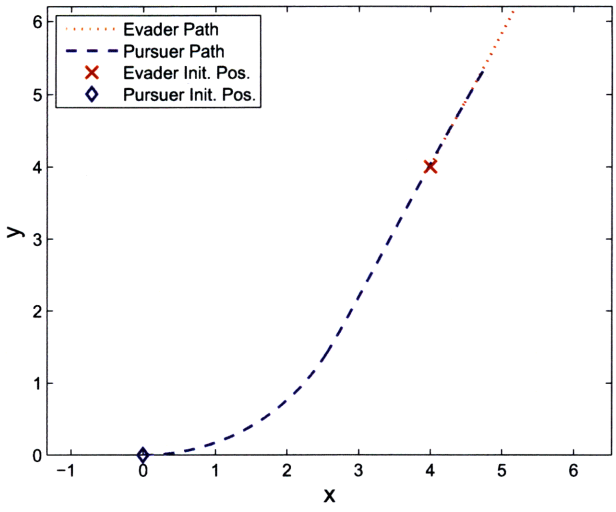

Figure 4-12: Homicidal Chauffeur Problem by Neural Network Method

FORTRAN MEX-file and sums the resulting cost and gradient outputs. The final summation of both those figures is sent to the `fminunc` solver to provide the search direction for the next weight adjustments. Figure 4-13 shows how the solver trained the weights in the first step for P to chase E. In order to achieve this result for the large training set, however, the cost function was changed to

$$J = \sqrt{\left(x_1\left(t_f\right) - x_2\left(t_f\right)\right)^2 + \left(y_1\left(t_f\right) - y_2\left(t_f\right)\right)^2} \tag{4.17}$$

in order to account for the fact that the capture time would not be the same for each scenario. This cost function substitution works because the trajectory for a minimum-time capture is the same as the trajectory that minimizes the distance between the players at the final time. Once the neural networks are trained for both players using the training set of initial conditions, the cost function can be changed back to $J = t_f$ and MATLAB's constrained minimization solver `fmincon` can find the capture time subject to the terminal constraint of Equation 4.15.

4.3.2 Football Problem

The football problem examines the interaction between two players during a hypothetical kick-off return. This problem is particularly useful as a framework from which to eventually build a combat scenario with realistic dynamics. As proposed by Breakwell and Merz [12], both players have fixed speed and simple motion yielding the dynamic equations

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix} = \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{x}_2 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} v_1 \cos u_1 \\ v_1 \sin u_1 \\ v_2 \cos u_2 \\ v_2 \sin u_2 \end{bmatrix} \tag{4.18}$$

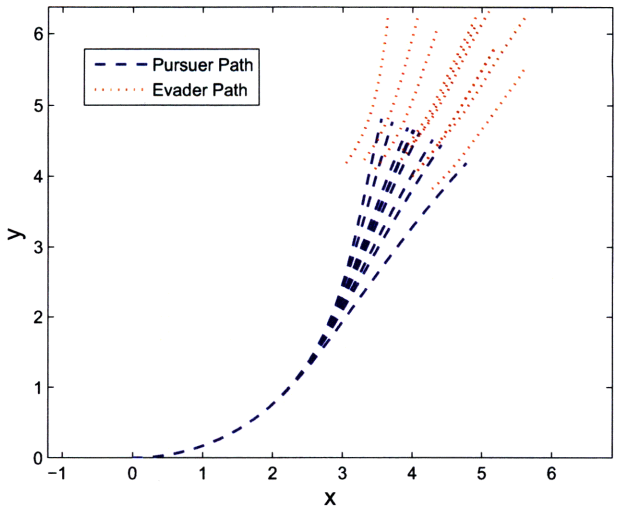

Figure 4-13: Training Set for Homicidal Chauffeur Problem

with initial conditions

$$\mathbf{x}(t_0) = \begin{bmatrix} x_1(t_0) \\ y_1(t_0) \\ x_2(t_0) \\ y_2(t_0) \end{bmatrix} = \begin{bmatrix} x_{10} \\ y_{10} \\ x_{20} \\ y_{20} \end{bmatrix} \tag{4.19}$$

The offensive player, E, begins near his own goal line ($y_{20} = 0$) and attempts to run up-field along the positive y-axis as far as possible. The resulting cost functional for the game is

$$J = y_2(t_f) \tag{4.20}$$

The pursuer, P, seeks to capture E or push him out of bounds as far down-field as possible. Capture takes place when E enters the capture radius of P according to the condition

$$l(\mathbf{x}(t_f), t_f) = (x_1(t_f) - x_2(t_f))^2 + (y_1(t_f) - y_2(t_f))^2 - d^2 = 0 \tag{4.21}$$

In the case where E has a slight speed advantage and both players begin in the lateral center of the field, the optimal strategy for E is to run directly towards P then arc towards either sideline while running just outside of the capture radius. In the free time problem, a wide field or a large speed advantage will allow E to move off this arc and run directly up-field avoiding both capture and the sideline. The neural net implementation below uses a fixed final time and does not enforce sideline boundaries.

In order to provide a valid gradient search direction to the `fminunc` solver, the cost function must be written so that adjustments to the weights do not cause discontinuous jumps in the cost. For example, if the current policy allowed E to skirt around the capture radius of P and make it up-field without getting caught, a slight change to the policy might cause E to get captured at half the distance as the previ-

ous policy. The jump in final cost would cause a discontinuity in the gradient search. To mitigate this problem, the cost calculation is posed so that the transition between a game without capture and a game with capture is continuous. The following code demonstrates how the final cost for each game is calculated:

```
EYdist = x(4,nt);
do i=nt-1,1,-1
    beta = 0
    distPE = sqrt((x(4,i)-x(2,i))**2+(x(3,i)-x(1,i))**2)
    if (distPE < CR) then
        beta = exp(-alpha*(distPE-CR))-1+alpha*(distPE-CR)
    end if
    EYdist = (beta*x(4,i)+EYdist)/(beta+1)
end do
C = -EYdist
```

`EYdist` begins as the up-field position $y_2(t_f)$ of E at the final time of the simulation. A search along the state history is conducted to see if E enters the capture radius (`CR`) of P at any time during the game. The `beta` term

$$\beta = e^{(-\alpha(\text{distPE}-\text{CR}))} - 1 + \alpha(\text{distPE} - \text{CR}) \tag{4.22}$$

is plotted for several values of `alpha` in Figure 4-14 with `CR` of 0.5. This term is used to continuously transition the cost of the game from the up-field position at the final time to the up-field location where capture actually occurs. The `beta` function has a zero derivative when `distPE = CR`. Another `beta` calculation can be conducted to penalize movement out of the side boundaries as well.

P is initially given the strategy to move towards E as shown in Figure 4-15. Once the weights for E are established, P is then allowed to find his optimal strategy. Rather than minimize the `EYdist` from the normal distribution of weights, a more general cost function

$$J = \int_0^{t_f} \sqrt{\left(x_1(t) - x_2(t)\right)^2 + \left(y_1(t) - y_2(t)\right)^2}\, dt \tag{4.23}$$

is assigned. This cost function directs the weights to minimize the distance between P and E during the entire game. The result of this training is shown in Figure 4-16. These weights are used as the starting point for the weight adjustments of the real cost function to minimize `EYdist`. After several iterations the result shown in Figure 4-17 is achieved where the bold lines represent the neural net method solution and the faint lines are the optimal solution.

The x and y displacements between the players serve as the input to the neural network during this fixed time solution where $t_f = 7$ s. There are three hidden layers with 5, 7, and 7 nodes, respectively. Each network has a single control output indicating the heading angle for the player. Seven initial conditions make up the initial training set. The evader begins at the origin (0, 0) in each scenario and the pursuer

begins at scattered locations up-field. The primary initial condition of concern is the one in which P is directly in front of E at location (0, 5). The capture radius for P is 0.5. E has a speed advantage over P so that $\frac{v_2}{v_1} = 1.10$. After the first two rounds of iterations, the training set is reduced to three points in the vicinity of (0, 5). Reduction of the training set often improves the solution for the next optimization, but this result comes at the risk of over-training. For example, if P over-trains to a capture of E on the left side of the field, during the next optimization E might break right and then move straight up-field while P continues to move towards the left side of the field completely disregarding the evader's change of strategy.

Based on the Breakwell's solution description, the result from the neural net policies is sub-optimal. The evader in Figure 4-17 moves right too early in the game and gives up substantial up-field distance compared to the optimal solution. One issue is that this problem is what Isaacs calls an *instantaneous mixed* problem. In the optimal solution, at the location just outside the pursuer's capture radius, the evader can either go left or right on the arc just outside the capture radius. In the neural net solution, however, E tends to pick one direction and then optimize strictly to that option. Whether the left or right arc is chosen is a function of the initial random weights used to establish the network. Once the strategy for E is established in one direction, P can take advantage of that tendency and move out in front E in order to decrease the pursuer's down-field distance at final time. The optimal strategy for E would direct him to the left if P was using the strategy shown in the figure.

Some deviation from the optimal solution also may have resulted from the use of the `beta` function to keep the cost continuous. In the optimal solution, the movement from the head-on collision course to the arc towards the sideline occurs instantaneously just outside the capture radius of P. The ramp up of the beta function may cause this instantaneous direction change to smooth out. As with previous uses of the neural net method, this problem is affected by the size of the neural network used as well as the breadth of the training set. More trials need to be conducted in order to determine what training sets produce the best results.

4.4 Discussion

The neural network method provides control policies for optimal control and differential game problems without knowing the structure of the solution beforehand. This benefit is a welcome departure from the boundary value method where plausible guesses for the state, costate, and control trajectories are required in order to achieve convergence. In addition, the solution comes in the form of policies that can apply to any games within the state-space of the training set.

There are some difficulties that still remain to be explored before the viability of this method for solving optimal control and differential game problems can be confirmed. One of the primary challenges is how to deal with boundary conditions. For the optimal control problems such as the orbit raising problem, boundary conditions were enforced by adding penalties to the final cost of the game. In the homicidal chauffeur differential game problem, the capture condition required a different simu-

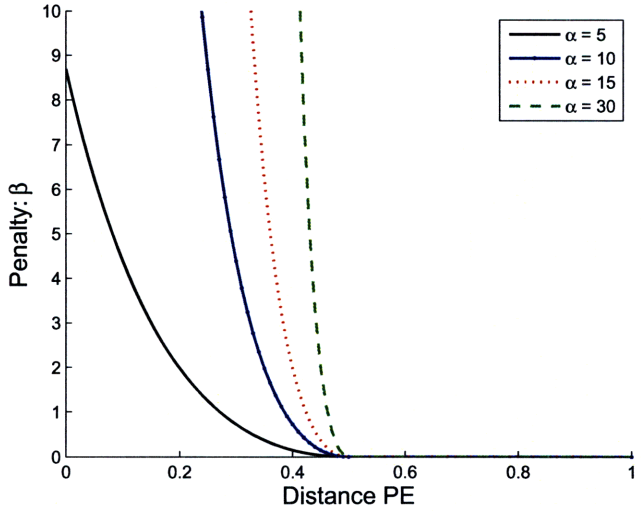

Figure 4-14: Penalty Function

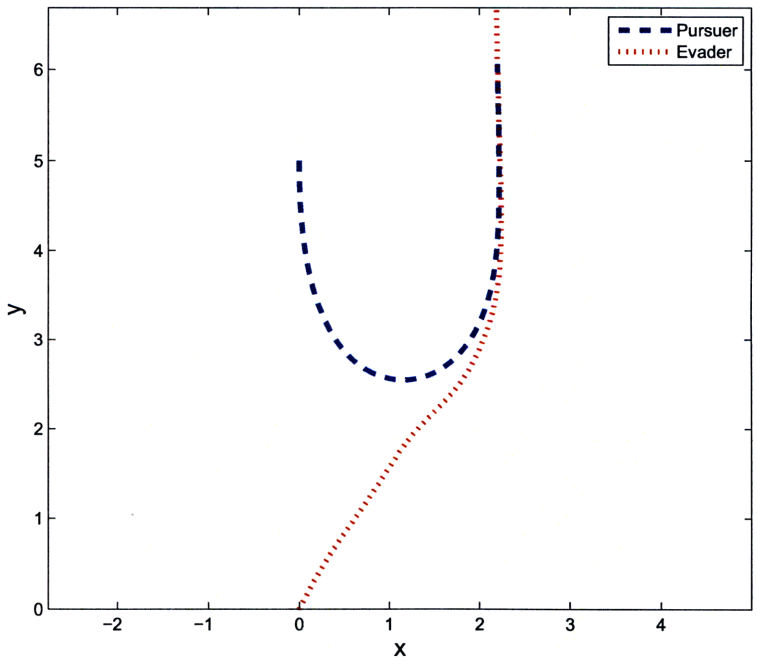

Figure 4-15: Football Problem, Initial Evader Strategy

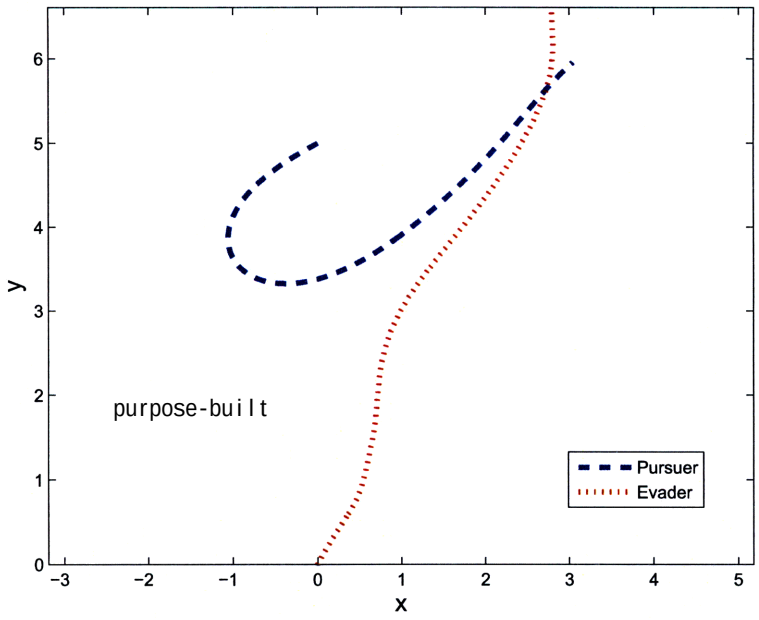

Figure 4-16: Football Problem, Pursuer Training

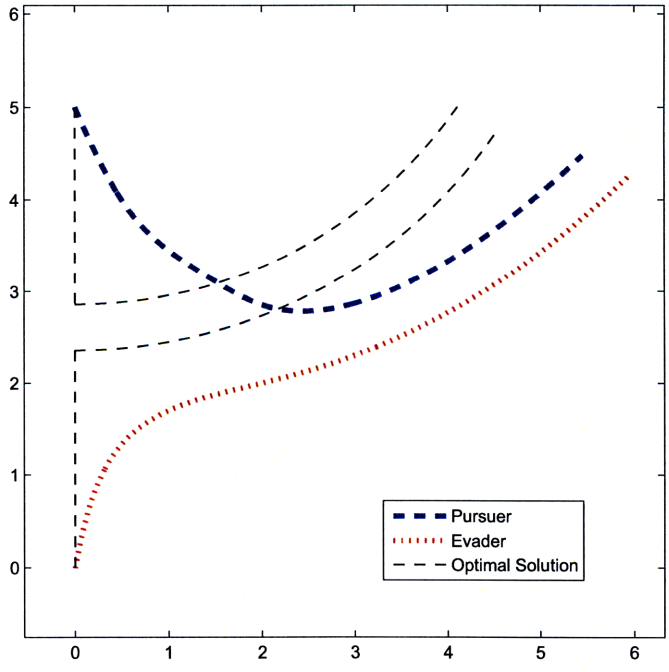

Figure 4-17: Football Problem, Sub-Optimal Policies

lation end time for each initial condition set. To get around this limitation the game was solved with a modified cost function. In the football problem E operated on an arc just outside the capture radius of P. A slight weight change could discontinuously drop the final down-field distance from the end of the arc to the beginning. The cost function had to be carefully reformulated so that there were no discontinuities. Each optimal and differential game problem examined required some ingenuity to achieve desirable results.

Another difficulty with this method is the sheer number of parameters that can be adjusted before solving a problem. There are no set rules for determining an adequate number of nodes, layers, or what type of transfer function to use. The scaling of the initial set of weights was also a result of trial and error. In addition, there is no ideal size for a training set in terms of the number of initial conditions or the spread of the conditions throughout the state-space. Future development of software routines that could do batch processing could aid the convergence to optimal solutions.

The issue of mixed situations, where two optimal solutions exist, presents a significant barrier to the viability of the neural network method for differential game problems. The network often optimizes to one direction and allows the other player to take advantage of that self-imposed limitation. If the neural network were capturing the optimal behavior, it would switch to the other optimal strategy if the opposing player tried to take advantage of a particular trajectory choice. Finally, the related issue of over training against the opposing player's control policy can also have disastrous results during the next weight optimization. A player that over trains to a particular trajectory will not be able to adapt during the next player's optimization process.

4.5 Conclusion

The use of neural networks to form control policies for optimal control and differential game problems is still in its infancy. Of all the techniques examined in this thesis, however, this method provides the most potential for the solution of large problems where the switching structure may not be readily apparent. With more study regarding how to solve free end-time problems and enforce boundary conditions this method could provide a sufficient means to do the type of parametric analysis that originally motivated this work.

Chapter 5

Conclusions

Three methods for solving differential game problems numerically have been examined. Each method offers advantages and disadvantages when compared with the other techniques. Sections 5.1 through 5.3 provide an overview of the benefits and shortcomings of each method discussed in this thesis. In section 5.4 recommendations are made regarding what solution method to use depending on the type of differential game problem encountered. Finally, Section 5.5 discusses what future work could aid the solution of differential game problems using numerical methods.

5.1 Boundary Value Method

The boundary value method is a mathematically rigorous approach that has many similarities to the classical method for solving optimal control problems. The basic procedure is to take a set of dynamics governing the motion of both players, form the Hamiltonian, analytically derive the necessary conditions, and enforce the conditions at the boundaries thus creating a boundary value problem. The final solution includes state, control, and costate information that can then be used to verify the optimality of the solution. Numerous software packages have been developed for a variety of programming languages that can solve these two-point and multi-point boundary value problems. In particular, the MATLAB `bvp4c` function provides a means for rapid construction and solution of boundary value problems.

While it may be appropriate for simple problems, the boundary value approach becomes much more difficult as the complexity and nonlinearity of the dynamics increases. The approach solves both the homicidal chauffeur and target guarding problems but requires significant help by way of good initial guesses for the state and costate trajectories in addition to the terminal time for the problem. The initial guess must incorporate knowledge of the switching structure that arises as the players move on and off the state and/or control constraints. Also, coming up with an initial guess for the costate can be non-intuitive yet have a significant impact on the final convergence of the problem. The costate requirement is a primary drawback of all indirect solution methods.

The boundary value method also requires analytical derivation of the necessary

conditions that can be cumbersome for higher order problems. In addition, the boundary constraints are highly dependent on the nature of the problem such as whether there is free or fixed terminal state or end time. An additional difficulty arises when solving differential game problems because knowledge of whether the game begins in the capture zone or the evasion zone is required. The zone will determine how the value function is defined as either a game of *kind* (does capture occur?) or a game of *degree* (how well does the player perform during the capture?). Finally, when the boundary value method does converge to a solution it only applies to the specific problem at hand where both players are acting optimally and does not reveal the policy of each player. Thus, a new simulation must be conducted for every situation being examined.

5.2 Decomposition Method

The decomposition method provides a linearized search method that can be used to solve differential game problems. In this approach, an initial arbitrary trajectory is provided for the evader, and the pursuer's course is optimized to minimize the value (cost) function. At the capture location an approximate linearized gradient of the value function is evaluated in the neighborhood of the capture point. The direction of the gradient indicates where the evader's final aim-point should be moved in order to maximize the value of the game. The pursuer's trajectory is optimized against the evader's new trajectory and the cycle repeats until changes to the evader's trajectory no longer increase the payoff.

As a direct method, one of the benefits of the decomposition approach is that it does not require any information about the costate. While the lack of costate information increases the likelihood of achieving convergence it remains more difficult to verify the optimality of the solution after the fact. Another benefit of the decomposition method is that it can be implemented using a variety of readily accessible solvers including MATLAB's `fmincon` function. This implementation enables easier handling of the boundary conditions and constraints compared with other more cumbersome, customized solvers. Another shortcoming of this approach is that the initial conditions often affect the final convergence of the problem. Certain target locations in the target guarding problem cause the evader to jump back and forth across the optimal solution without ever converging to the final solution. There is no way to know what initial conditions will result in convergence.

5.3 Neural Network Method

The method of neural nets provides a way to directly solve for the control policies for optimal control and differential game problems. In this method a set of weights is given to a neural network that determines the output of the network for a given input. The relevant system states are the inputs to the neural network and the output is the control signal. The input to the simulation is a set of weights $\mathbf{w}$, the initial conditions

(or training set of initial conditions), and the time period, and the output is the final cost J and a cost gradient $\frac{\partial J}{\partial \mathbf{w}}$. The MATLAB solver adjusts the weights based on the direction indicated by the gradient in order to further reduce the cost. The simulation is then repeated with the new weights and the solver once again adjusts the weights to further reduce the cost. The solver continues to adjust the weights until the minimum cost is found. The final weights define the control policy. Once the policy for one player is found, the other player's weights are optimized against the previous player's policy. The weights which govern the control policies are optimized iteratively until a dual-optimal solution is found.

This method requires no knowledge of the costate or of the control switching structure. The initial guess for the set of weights is a random, normal distribution. Once the weights for the control policy are established, the policy can be used for any initial condition within the range of the training set. With an adequately large training set, the weights will be able to provide a closed loop policy for a variety of different initial starting conditions with no need for further analysis. There are no guarantees of good player behavior when a situation arises that is outside the range of the neural network training set. There is also no certainty that the solution that the solver converges to is globally optimal as the set of random weights can often cause convergence to local minima.

Several optimal control problems were solved using the neural network method. While the method produced agreeable solutions that matched or closely followed the optimal solutions, some additions had to be made to the cost functions in order to achieve the desired results. Specifically, dealing with terminal conditions presented a challenge. Currently, the only way to enforce them is to add a penalty to the terminal cost for any violation of a terminal condition.

The neural network method was also applied to two differential game problems. The homicidal chauffeur problem demonstrated the difficulty of using neural networks when dealing with free end-time problems that are typical of pursuit-evasion scenarios. The football problem revealed how careful formulation of the cost function is required so that no discontinuous jumps occur while the solver adjusts the weights of the control policies. This problem also demonstrated how the networks can often over-train against the opposing players policy. Over-training reduces the ability of the policy to adapt during the next player's control policy optimization. Additional difficulties are encountered when *instantaneous mixed* arise and a player has two equally good trajectory options.

5.4 Recommendations

For smaller problems, the boundary value or decomposition methods may require the least amount of preparation before actual problems can be solved in MATLAB. The boundary value method provides the most mathematically rigorous option and has very high accuracy. It can also provide costate information for the verification of the optimality of the solution. The decomposition method is not as difficult to set up but is limited in that there is no clear way to know what problems will converge to a

final solution. As demonstrated by the target guarding problem, a simple movement of the initial conditions can make the difference between a simulation that converges and one that does not. If the goal is to solve more realistic, nonlinear problems, then the neural network method may be the best option, though the optimality of the solution is not guaranteed. Further improvement of the solution procedure regarding how to choose the number of nodes, layers, training set, and initial distribution of weights will help establish the method as a viable option for the parametric analysis that motivated this work.

5.5 Future Work

Because the boundary value and decomposition methods have received thorough treatments in the literature, the primary area for future work is with the neural network method. One specific area for research concerns how the neural networks are established and trained. Currently, the number of layers and nodes, the size of the initial random distributions, and the number of initial conditions in the training set are all established through trial and error. More research could aid the determination of general guidelines for network creation based on the complexity of the problem.

Another area worthy of future work regards the convergence rate for the neural network method. While this thesis demonstrated that neural networks can be used to emulate the optimal solutions of some small problems, a better understanding of how fast and how well the method converges to the optimal solution is necessary before its viability can be confirmed. In principle, highly dimensional nonlinear problems are difficult to solve (Bellman's "curse of dimensionality" [26]), but in some cases the optimal control solutions can be represented in simple terms. As the complexity of the solution increases the more neurons will be required to accurately capture the solution behavior and the rate of convergence may slow and fail to reach the optimal solution.

Assuming that neural network methods can find solutions to optimal control problems, there is still no guarantee that they will be able to solve differential game problems. As demonstrated during the football problem, over-training can become a significant issue when dealing with a differential game. The other primary issue is dealing with what Isaacs refers to as *instantaneous mixed* locations where two or more equally good solutions exist. Further study needs to be done to determine if it is possible that the correct switching behaviors will emerge or if it possible to introduce logic to the neural networks in order to capture these behaviors. Lastly, if the issues of over training and mixed problems can be solved for small problems such as the football problem, work must be completed to determine how the networks and training sets can be altered in order to obtain the optimal solutions for highly dimensional problems with realistic, nonlinear dynamics.

Appendix A

Differential Game Code

A.1 Boundary Value Problems

A.1.1 Target Guarding Problem

```
function BVP_TG
global c0 x10 x20 v w l
%%% Initial Conditions
c0  = [50; -40];    % Target position
x10 = [10; -50];    % P initial position
x20 = [0; 0];       % E initial position
v   = 1;            % P speed
w   = 1;            % E speed
l   = 0.03;         % "Armlength" distance of P
alpha0 = .7;
Tg     = 47;
params = [alpha0, Tg];
% Initial Solution
solinit = bvpinit(linspace(0,1),@TG_init,params);
% Run Solver
options = bvpset('stats','on');
sol = bvp4c(@TG_ode,@TG_bc,solinit,options);
alpha = sol.parameters(1)
max = size(sol.y,2);
dist = norm(sol.y(1:2,max)-sol.y(3:4,max))
p = sol.y(5:8,max)
tf = sol.parameters(2)
% Final Solution Plot
    hold on
    state = sol.y(1:4,:);
    plot(state(3,:),state(4,:),'r:','LineWidth',1.5)
    plot(state(1,:),state(2,:),'b--','LineWidth',1.5)
    plot(c0(1),c0(2),'x','LineWidth',2,'MarkerSize',12)
    grid off
    xlabel('x','FontSize',14);ylabel('y','FontSize',14);

    % Plot perpendicular bisector
    xm = ( x10(1)-x20(1) )/2; ym = ( x10(2)-x20(2) )/2;
    m  = -( (x10(2)-x20(2))/(x10(1)-x20(1)) )^-1;
```

```
    b1  = ym - m*xm;
    xb1 = -10; xb2 = 60; yb1 = m*xb1+b1; yb2 = m*xb2+b1;
    hold on
    b2 = c0(2)+1/m*c0(1);
    x_cpa  = (b2 - b1)/(m+1/m); y_cpa  = m*x_cpa + b1;
    scatter(x_cpa,y_cpa);
    plot([xb1,xb2],[yb1,yb2],'k--');
    axis equal
end
function dydt = TG_ode(t,y,params)
global c0 x10 x20 v w l
T = params(2);
p1 = y(5); p2 = y(6); p3 = y(7); p4 = y(8);
cosu1   = 1/sqrt(1+(p2/p1)^2);
sinu1   = (p2/p1)/sqrt(1+(p2/p1)^2);
cosu2   = 1/sqrt(1+(p4/p3)^2);
sinu2   = (p4/p3)/sqrt(1+(p4/p3)^2);
dydt=T*[v*cosu1;v*sinu1;w*cosu2;w*sinu2;0;0;0;0];
end
function res = TG_bc(ya,yb,params)
global c0 x10 x20 v w l
alpha = params(1);
T     = params(2);
Tmin = 20;
if T < Tmin
    W = Tmin;
else
    W = T;
end
p1 = yb(5); p2 = yb(6); p3 = yb(7); p4 = yb(8);
cosu1b   = 1/sqrt(1+(p2/p1)^2);
sinu1b   = (p2/p1)/sqrt(1+(p2/p1)^2);
cosu2b   = 1/sqrt(1+(p4/p3)^2);
sinu2b   = (p4/p3)/sqrt(1+(p4/p3)^2);
res = [ ya(1)-x10(1);ya(2)-x10(2);
        ya(3)-x20(1);ya(4)-x20(2);
        norm(yb(1:2)-yb(3:4))-l;
        yb(5) - alpha*(yb(1)-yb(3))/norm(yb(1:2)-yb(3:4));
        yb(6) - alpha*(yb(2)-yb(4))/norm(yb(1:2)-yb(3:4));
        yb(7) - (yb(3)-c0(1))/norm(c0-yb(3:4)) - ...
          alpha*(yb(3)-yb(1))/norm(yb(1:2)-yb(3:4));
        yb(8) - (yb(4)-c0(2))/norm(c0-yb(3:4)) - ...
          alpha*(yb(4)-yb(2))/norm(yb(1:2)-yb(3:4));
        yb(5)*v*cosu1b+yb(6)*v*sinu1b+yb(7)*w*cosu2b+yb(8)*w*sinu2b];
%        T-W];
end
function g = TG_init(tau)
global c0 x10 x20 v w l
% Guess End Location: True x ~ 43 y ~ -17
xg  = 43;
yg  = -30;
% Calculate End Time
%Tg  = norm(x10-[xg;yg])/v;
%Tg  = norm(x10-c0)/v;
```

```
Tg = 1*47;
u1g = atan2(yg-x10(2),xg-x10(1));
u2g = atan2(yg-x20(2),xg-x20(1));
t = tau*Tg;
xp = [v*t*cos(u1g)+x10(1);v*t*sin(u1g)+x10(2)];
xe = [w*t*cos(u2g)+x20(1);w*t*sin(u2g)+x20(2)];
p1 = cos(u1g); p2 = sin(u1g); p3 = cos(u2g); p4 = sin(u2g);
g = [xp;xe;p1;p2;p3;p4];
end
```

A.1.2 Homicidal Chauffeur Problem

```
function BVP_HC
clc
close all
clear all
%%% Initial Conditions
% Pursuer
x10     = [0;0];
phi10   = 0;
% Evader
x20     = [4;4];
% Parameters
v1      = 3;
v2      = 1;
u1max   = 1;
l       = 1;
% Initializing variables for the initial guess
t_p     = 0;            phi1_p  = phi10;
x1_p    = x10(1);       y1_p    = x10(2);
x2_p    = x20(1);       y2_p    = x20(2);
% Initial guess for lagrange multiplier of terminal cond.
alpha0  = .6;
% Threshold for p3 below which p3 = 0
thresh  = 0.02;
N       = 100;
mesh    = linspace(0,1,N);
% Create initial guess for state and costate
solinit = bvpinit(mesh,@HC_init,alpha0);
% Run BVP Solver
options = bvpset('stats','on');
sol = bvp4c(@HC_ode,@HC_bc,solinit,options);
alpha = sol.parameters
max = size(sol.y,2);
tf = sol.y(11,max)
% Plot Results
    if 1
        figure
        hold on
        Pstate = sol.y(1:2,:);
        Estate = sol.y(4:5,:);
        plot(Estate(1,:),Estate(2,:),'r:','linewidth',1.5)
        plot(Pstate(1,:),Pstate(2,:),'b--','linewidth',1.5)
```

```
    plot(x20(1),x20(2),'rx','LineWidth',1.5,'MarkerSize',10);
    plot(x10(1),x10(2),'bd','LineWidth',1.5,'MarkerSize',6);
    legend('Evader Path','Pursuer Path','Evader Init. Pos.', ...
        'Pursuer Init. Pos.','location','NorthWest');
    axis equal
    xlabel('x','FontSize',14);ylabel('y','FontSize',14)
    shg
    hold off
    figure
    P3 = sol.y(8,:);
    tout = tf*sol.x;
    plot(tout,P3);
end
function dydt = HC_ode(t,y,alpha)
    % States
    x1  = y(1);     x2  = y(4);
    y1  = y(2);     y2  = y(5);
    phi1= y(3);
    % Costates
    p1  = y(6);     p4  = y(9);
    p2  = y(7);     p5  = y(10);
    p3  = y(8);
    % Final Time
    T   = y(11);

    u1 = -p3;
    if u1 > thresh
        u1 = u1max;
    elseif u1 < -thresh
        u1 = -u1max;
    end
    cosu2   = 1/sqrt(1+(p5/p4)^2);
    sinu2   = (p5/p4)/sqrt(1+(p5/p4)^2);
    x1_d    = v1*cos(phi1);
    y1_d    = v1*sin(phi1);
    phi1_d  = u1;
    x2_d    = v2*cosu2;
    y2_d    = v2*sinu2;
    p1_d    = 0;
    p2_d    = 0;
    p3_d    = p1*v1*sin(phi1) - p2*v1*cos(phi1);
    p4_d    = 0;
    p5_d    = 0;
    dt      = 0;
    dydt = T*[x1_d; y1_d; phi1_d; x2_d; y2_d; ...
                  p1_d; p2_d; p3_d; p4_d; p5_d; dt];
end

function res = HC_bc(ya,yb,alpha)
    p3T  = yb(8);
    u1T   = -p3T;
    if u1T > thresh
        u1T = u1max;
    elseif u1T < -thresh
```

```
        u1T = -u1max;
    end
    cosu2   = 1/sqrt(1+(yb(10)/yb(9))^2);
    sinu2   = (yb(10)/yb(9))/sqrt(1+(yb(10)/yb(9))^2);

    res = [ya(1)-x10(1);ya(2)-x10(2);
           ya(3)-phi10;
           ya(4)-x20(1);ya(5)-x20(2);
           norm(yb(1:2)-yb(4:5))-l;
           yb(6) - alpha*(yb(1)-yb(4))/norm(yb(1:2)-yb(4:5));
           yb(7) - alpha*(yb(2)-yb(5))/norm(yb(1:2)-yb(4:5));
           yb(8) - 0;
           yb(9) - alpha*(yb(4)-yb(1))/norm(yb(1:2)-yb(4:5));
           yb(10)- alpha*(yb(5)-yb(2))/norm(yb(1:2)-yb(4:5));
           yb(6)*v1*cos(yb(3))+yb(7)*v1*sin(yb(3))+yb(8)*u1T+...
           yb(9)*v2*cosu2+yb(10)*v2*sinu2+1];
end
function v = HC_init(tau)
    %u2g     = 58*pi/180;
    u2g     = 45*pi/180;
    phi_Tg  = u2g;
    %Tg      = 2.5;
    Tg      = 2.3;

    if  phi1_p < phi_Tg
        p3 = -1;
    else
        p3 = 0;
    end
    u1 = -p3;
    phi1_d = u1;
    t       = tau*Tg;
    dt      = t - t_p;
    phi1    = phi1_p + dt*phi1_d;
    p1 = -1;
    p2 = 2*p1;
    p4 = -p1;
    p5 = -p2;
    cosu2   = 1/sqrt(1+(p5/p4)^2);
    sinu2   = (p5/p4)/sqrt(1+(p5/p4)^2);
    x1_d    = v1*cos(phi1);
    y1_d    = v1*sin(phi1);
    x2_d    = v2*cosu2;
    y2_d    = v2*sinu2;
    x1      = x1_p + dt*x1_d;
    y1      = y1_p + dt*y1_d;
    x2      = x2_p + dt*x2_d;
    y2      = y2_p + dt*y2_d;
    v = [x1;y1;phi1;x2;y2;p1;p2;p3;p4;p5;Tg];
    phi1_p  = phi1;
    x1_p    = x1;
    y1_p    = y1;
    x2_p    = x2;
    y2_p    = y2;
```

```
        t_p     = t;
    end
end
```

A.2 Decomposition Method

A.2.1 Target Guarding Problem

```
function Decomp_TG
% Other Initial Conditions
N = 50;             % Number collocation points
v   = 1;            % P speed
w   = 1;            % E speed
d   = 0.001;        % "Armlength" distance of P
if 1 % Target Guarding
%%% Initial conditions of Pursuer and Evader
c0  = [15; 5];      % Target position
x10 = [5; 0];       % P initial position
x20 = [0; -.5];     % E initial position
end
%%% Initial conditions for P:min %%%
A = [];             % Linear inequality constraints
b = [];             %   A*X  <= b
Aeq = [];           % Linear equality constraints
beq = [];           %   Aeq*X  = Beq
lb = [];          % Bounds
ub = [];     %   lb <= X <= ub
% Initial Guess
% xg = 2.1;
% yg = 3.7;
xg = 5;
yg = 5;
[x1,x2,u1,u2,T1,T2] = TG_init(xg,yg);
T2_step = linspace(0,T2,N);
% Loop parameters
tol   = 1;
count = 0;
options = optimset('Display','final');
while tol > 0.0050

    u2_p = u2;
    count = count + 1

    %%% P: min
    [x,fval,exitflag,output,lambda] = fmincon(@dist_fun, ...
        [u1;T1],A,b,Aeq,beq,lb,ub,@mycon,options);

    % Store lagrange multiplier associated with capture condition 1 = 0
    lagrange = lambda.ineqnonlin(1);
    T1  = x(N+1); % Final time at intercept
    u1 = x(1:N);  % P control angle to achieve optimal intercept
```

```
    e0 = x2_f;

    %Partial derivative of payoff, q, WRT changes in e0
    dqde = -[(e0(1)-c0(1))/norm(c0-e0);
             (e0(2)-c0(2))/norm(c0-e0)];
    % Partial derivative of capture condition, l, WRT changes in e0
    dlde = [(e0(1)-x1_f(1))/norm(x1_f-e0);
            (e0(2)-x1_f(2))/norm(x1_f-e0)];

    % Gradient of the value function
    c = dqde+lagrange*dlde;

    %%% E: max
    %u2 = fmincon(@val_fun,u2_p,A,b,Aeq,beq,lb,ub,@mycon2,options);
    %u2 = fminunc(@val_fun,u2_p);
    bound = 10*pi/180;
    u2 = fmincon(@val_fun,u2_p,A,b,Aeq,beq,u2_p-bound,u2_p+bound);

    tol = norm(u2-u2_p);

    T2_step = linspace(0,T1,N);
end
% Plot Commands
    if 1
        hold on;
        plot( x2(1,:), x2(2,:),'r:','LineWidth',1.5)
        plot( x1(1,:), x1(2,:),'b--','LineWidth',1.5)
        plot(c0(1),c0(2),'x','LineWidth',2,'MarkerSize',12)

        xm = ( x10(1)-x20(1) )/2; ym = ( x10(2)-x20(2) )/2;
        m  = -( (x10(2)-x20(2))/(x10(1)-x20(1)) )^-1;
        b1  = ym - m*xm;
        xb1 = 1; xb2 = 3; yb1 = m*xb1+b1; yb2 = m*xb2+b1;
        b2 = c0(2)+1/m*c0(1);
        x_cpa  = (b2 - b1)/(m+1/m); y_cpa  = m*x_cpa + b1;
        scatter(x_cpa,y_cpa);

        plot(x20(1),x20(2),'rx','LineWidth',1.5,'MarkerSize',10);
        plot(x10(1),x10(2),'bd','LineWidth',1.5,'MarkerSize',6);

        legend('Evader','Pursuer','Target','Intercept');
        axis equal
        xlabel('x','FontSize',14);ylabel('y','FontSize',14);
        %title({'Decomposition Method','Homicidal Chauffeur Problem'})
        disp(['Final Time = ', num2str(T1)]);

        plot([xb1,xb2],[yb1,yb2],'k--')

    end

function q = dist_fun(x)
    T = x(N+1,1); % Final Time

    % Final positions of E
```

```
    x2_f = [interp1(T2_step',x2(1,:),T,'linear','extrap'); ...
            interp1(T2_step',x2(2,:),T,'linear','extrap')];

    q = -norm(c0-x2_f);
    end
function [ciq, ceq] = mycon(x)

    % Values being optimized
    u1_fn   =  x(1:N,1);    % Control u1 at discrete points
    T      =   x(N+1);     % Final time
    % Final positions of E
    x2_f = [interp1(T2_step',x2(1,:),T,'linear','extrap'); ...
            interp1(T2_step',x2(2,:),T,'linear','extrap')];
    % Position of P at each time step
    dt = T/(N-1);
    x1(:,1) = x10(1:2,1);
    for i = 1:N-1
        x1(:,i+1) = [ v*dt*cos(u1_fn(i)) + x1(1,i);
                      v*dt*sin(u1_fn(i)) + x1(2,i)];
    end
    x1_f = x1(:,N); % Final Position
    % Inequality constraints
    ciq = [norm(x1_f-x2_f)-d];    % Capture condition
    % Equality constraints
    ceq = [0];

end
function v = val_fun(x)
    % Value being optimized
    u2 = x;
    % New E path final after changing u2
    x2(:,1) = x20(1:2,1); % [x, y, t]
    dt = T1/(N-1);
    for i = 1:N-1
        x2(:,i+1) = [w*dt*cos(u2(i))+ x2(1,i);
                     w*dt*sin(u2(i))+ x2(2,i)];
    end
    x2_f = x2(:,N);
    % Value function (linearized)
    v = -( c'*(x2_f-e0) );
end
    function [x1_g, x2_g, u1_g, u2_g, T_g1, T_g2] = TG_init(x_g,y_g)
    % Initial Angle
    u1_G = atan2((y_g-x10(2)),(x_g-x10(1)));
    u2_G = atan2((y_g-x20(2)),(x_g-x20(1)));
    T_g1 = norm([x_g;y_g]-x10(1:2))/v;
    T_g2 = norm([x_g;y_g]-x20(1:2))/w;

    u1_g = u1_G*ones(N,1);
    u2_g = u2_G*ones(N,1);
    dt = T_g1/(N-1);
    x1_g(:,1) = x10(1:2);
    for i = 1:N-1
        x1_g(:,i+1) = [ v*dt*cos(u1_g(i)) + x1_g(1,i);
```

```
                        v*dt*sin(u1_g(i)) + x1_g(2,i)];
    end
    dt = T_g2/(N-1);
    x2_g(:,1) = x20(1:2);
    for i = 1:N-1
        x2_g(:,i+1) = [ w*dt*cos(u2_g(i))+ x2_g(1,i);
                        w*dt*sin(u2_g(i))+ x2_g(2,i)];
    end
end
end
```

A.2.2 Homicidal Chauffeur

```
function Decomp_HC
% Other Initial Conditions
N = 75;             % Number collocation points
v   = 3;            % P speed
w   = 1;            % E speed
d   = 1;            % "Armlength" distance of P
u1max = 1;          % Limitation of P angular rate
case2 = 0;
case4 = 0;
if 1 % Homicidal Chauffeur 1
%%% Initial conditions of Pursuer and Evader
x10 = [0;0;0];      % P initial state [x10,y10,phi10]
x20 = [4;4;1.0783]; % E initial state [x20,y20,phi20]
end
if 0 % Homicidal Chauffeur 2
x10 = [0;0;0];
x20 = [-7;0;0];
case2 = 1;
end
if 0 % Homicidal Chauffeur 3
x10 = [0;0;0];
x20 = [2.5;7.5;0];
end
if 0 % Homicidal Chauffeur 4
x10 = [0;0;0];
x20 = [-1;3;0];
case4 = 1;
end
%%% Initial conditions for P:min %%%
A = [];             % Linear inequality constraints
b = [];             %   A*X  <= b
Aeq = [];           % Linear equality constraints
beq = [];           %   Aeq*X  = Beq
lb = [];            % Bounds
ub = [];            %   lb <= X <= ub
%%% E initial trajectory (arbitrary, away from P)
u2g   = atan2( (x20(2) - x10(2)),(x20(1)-x10(2)) );
% Game time guess
T = norm(x10(1:2)-x20(1:2))/abs(v-w); % Vrelative/InitDistance
if case2
```

```
    T = 9; % for case 2
end
% Form initial guess for E (moving away from P in time)
u2 = u2g*ones(N,1);
% u1 Condition for case 4
if case4
    T  = 8;
    ang = 74*pi/180;
    u2 = ang*ones(N,1);
end
% The E Trajectory in x,y coordinates
x2(:,1) = x20(1:2);
dt = T/(N-1);
for i = 1:N-1
    x2(:,i+1) = [w*dt*cos(u2(i))+ x2(1,i);
                  w*dt*sin(u2(i))+ x2(2,i)];
end
x2_f = x2(:,N);
T_step = linspace(0,T,N);
%%% Form initial guess for P
u1 = u2g*ones(N,1);    % Assume u1 = u2
% u1 Condition for case 4
if case4
    T  = 8;
    ang = 74*pi/180;
    u1  = [linspace(0,-2*pi+ang,4*N/9)';(-2*pi+ang)*ones(5*N/9+1,1)];
end
% Loop parameters
tol   = 1;
count = 0;
options = optimset('Display','off');
while tol > 0.0050

    u2_p = u2;
    count = count + 1

    %%% P: min
    [x,fval,exitflag,output,lambda] = fmincon(@time_fun, ...
        [u1;T],A,b,Aeq,beq,lb,ub,@mycon,options);

    % Store lagrange multiplier associated with capture condition l = 0
    lagrange = lambda.ineqnonlin(1);
    T  = x(N+1);  % Final time at intercept
    u1 = x(1:N);  % P control angle to achieve optimal intercept

    e0 = x2_f;

    %Partial derivative of payoff, q, WRT changes in e0
    dqde = -[0;0];
    % Partial derivative of capture condition, l, WRT changes in e0
    dlde = [(e0(1)-x1_f(1))/norm(x1_f-e0);
            (e0(2)-x1_f(2))/norm(x1_f-e0)];

    % Gradient of the value function
```

```
    c = dqde+lagrange*dlde;

    %%% E: max
    u2 = fmincon(@val_fun,u2_p,A,b,Aeq,beq,lb,ub,@mycon2,options);

    tol = norm(u2-u2_p);

    T_step = linspace(0,T,N);

end
% Plot Commands
    if 1
        figure
        hold on;
        %grid on
        plot( x2(1,:), x2(2,:),'r:','LineWidth',1.5)
        plot( x1(1,:), x1(2,:),'b--','LineWidth',1.5)
        plot(x20(1),x20(2),'rx','LineWidth',1.5,'MarkerSize',10);
        plot(x10(1),x10(2),'bd','LineWidth',1.5,'MarkerSize',6);
        legend('Evader Path','Pursuer Path','Evader Init. Pos.', ...
            'Pursuer Init. Pos.','location','NorthWest');
        axis equal
        xlabel('x','FontSize',14);ylabel('y','FontSize',14);
        %title({'Decomposition Method','Homicidal Chauffeur Problem'})
        disp(['Final Time = ', num2str(T)]);
    end

function q = time_fun(x)
    % Value function being optimized (Min Time)
    q = x(N+1,1);
    end
function [c, ceq] = mycon(x)

    % Values being optimized
    u1_f   =   x(1:N,1);   % Control u1 at discrete points
    T      =   x(N+1);     % Final time
    % Final positions of E
    x2_f = [interp1(T_step',x2(1,:),T,'linear','extrap'); ...
            interp1(T_step',x2(2,:),T,'linear','extrap')];
    % Position of P at each time step
    dt = T/(N-1);
    x1(:,1) = x10(1:2,1);
    for i = 1:N-1
        x1(:,i+1) = [ v*dt*cos(u1_f(i)) + x1(1,i);
                      v*dt*sin(u1_f(i)) + x1(2,i)];
    end
    x1_f = x1(:,N); % Final Position
    % Calculate (N-1) derivatives of the path
    for i = 1:N-1
        u1_d(i,1) = abs(u1_f(i+1)-u1_f(i))/dt;
    end
    % Inequality constraints
    c = [norm(x1_f-x2_f)-d    % Capture condition
         u1_d - u1max];       % Turn rate constraint
```

```
    % Equality constraints
    ceq = [x10(3) - x(1)];    % Enforce initial condition of phi1

end
function v = val_fun(x)
    % Value being optimized
    u2 = x;
    % New E path final after changing phi2
    x2(:,1) = x20(1:2,1); % [x, y, t]
    dt = T/(N-1);
    for i = 1:N-1
        x2(:,i+1) = [w*dt*cos(u2(i))+ x2(1,i);
                     w*dt*sin(u2(i))+ x2(2,i)];
    end
    x2_f = x2(:,N);
    % Value function (linearized)
    v = -( c'*(x2_f-e0) );
end
function [c, ceq] = mycon2(x)
    % Values being optimized
    u2   =   x;   % Control u1 at discrete points
    % Position of E at each time step
    x2(:,1) = x20(1:2,1); % [x, y, t]
    dt = T/(N-1);
    for i = 1:N-1
        x2(:,i+1) = [w*dt*cos(u2(i))+ x2(1,i);
                     w*dt*sin(u2(i))+ x2(2,i)];
    end
    x2_f = x2(:,N);
    % Inequality constraints
    c = [];        % No constraint on E turn rate
    % Equality constraints
    ceq = [x20(3) - x(1)];      % Enforce initial condition of phi2
end
end
```

A.2.3 Game of Two Cars

```
function Decomp_2C
close all
clear
clc
N = 50;             % Number collocation points
if 0 % Two Cars 1
%%% Initial conditions of Pursuer and Evader
x10 = [0;0;pi];     % P initial state [x10,y10,phi10]
x20 = [4;4;-pi];    % E initial state [x20,y20,phi20]
v   = 2;            % P speed
w   = 1;            % E speed
d   = 1;            % "Armlength" distance of P
u1max = 1;          % Limitation of P angular rate
u2max = 2;          % Limitation of E angular rate
end
```

```
if 1 % Two Cars 2
%%% Initial conditions of Pursuer and Evader
x10 = [0;0;-2*pi/3];    % P initial state [x10,y10,phi10]
x20 = [4;4;-2*pi/3];    % E initial state [x20,y20,phi20]
v   = 3;                % P speed
w   = 1;                % E speed
d   = 1;                % "Armlength" distance of P
u1max = .66;            % Limitation of P angular rate
u2max = 1.25;           % Limitation of E angular rate
end
% E initial trajectory (arbitrary, away from P)
u2g   = atan2( (x20(2) - x10(2)),(x20(1)-x10(2)) );
% Game time guess
T = norm(x10(1:2)-x20(1:2))/abs(v-w); % Vrelative/InitDistance

%%% Initial conditions for P:min %%%
A = [];             % Linear inequality constraints
b = [];             %   A*X  <= b
Aeq = [];           % Linear equality constraints
beq = [];           %   Aeq*X  = Beq
lb = [];            % Bounds
ub = [];            %   lb <= X <= ub
%%% Form initial guess for P
u1 = u2g*ones(N,1);    % Assume u1 = u2
dt = T/(N-1);
x1(:,1) = x10(1:2);
for i = 1:N-1
    x1(:,i+1) = [ v*dt*cos(u1(i)) + x1(1,i);
                  v*dt*sin(u1(i)) + x1(2,i)];
end
x1_f = x1(:,N);
%%% Form initial guess for E (moving away from P in time)
u2 = u2g*ones(N,1);
x2(:,1) = x20(1:2);
for i = 1:N-1
    x2(:,i+1) = [w*dt*cos(u2(i))+ x2(1,i);
                 w*dt*sin(u2(i))+ x2(2,i)];
end
x2_f = x2(:,N);
T_step = linspace(0,T,N);
% Loop parameters
tol   = 1;
count = 0;
options = optimset('Display','off');
while tol > 0.0050

    u2_p = u2;
    count = count + 1

    %%% P: min
    [x,fval,exitflag,output,lambda] = fmincon(@time_fun, ...
        [u1;T],A,b,Aeq,beq,lb,ub,@mycon,options);

    % Store lagrange multiplier associated with capture condition l = 0
```

```
    lagrange = lambda.ineqnonlin(1);
    T  = x(N+1);  % Final time at intercept
    u1 = x(1:N);  % P control angle to achieve optimal intercept

    e0 = x2_f;

    %Partial derivative of payoff, q, WRT changes in e0
    dqde = -[0;0];
    % Partial derivative of capture condition, l, WRT changes in e0
    dlde = [(e0(1)-x1_f(1))/norm(x1_f-e0);
            (e0(2)-x1_f(2))/norm(x1_f-e0)];

    % Gradient of the value function
    c = dqde+lagrange*dlde;

    %%% E: max
    u2 = fmincon(@val_fun,u2_p,A,b,Aeq,beq,lb,ub,@mycon2,options);

    tol = norm(u2-u2_p);

    T_step = linspace(0,T,N);

end
% Plot Commands
    if 1
        figure
        hold on;
        %grid on
        plot( x2(1,:), x2(2,:),'r:','LineWidth',1.5)
        plot( x1(1,:), x1(2,:),'b--','LineWidth',1.5)
        plot(x20(1),x20(2),'rx','LineWidth',1.5,'MarkerSize',10);
        plot(x10(1),x10(2),'bd','LineWidth',1.5,'MarkerSize',6);
        legend('Evader Path','Pursuer Path','Evader IP', ...
            'Pursuer IP','location','NorthWest');
        axis equal
        xlabel('x','FontSize',14);ylabel('y','FontSize',14);
        %title({'Decomposition Method','Two Cars Problem'})
        disp(['Final Time = ', num2str(T)]);
    end

function q = time_fun(x)
    % Value function being optimized (Min Time)
    q = x(N+1,1);
    end
function [c, ceq] = mycon(x)

    % Values being optimized
    u1_f   =   x(1:N,1);   % Control u1 at discrete points
    T      =   x(N+1);     % Final time
    % Final positions of E
    x2_f = [interp1(T_step',x2(1,:),T,'linear','extrap'); ...
            interp1(T_step',x2(2,:),T,'linear','extrap')];
    % Position of P at each time step
    dt = T/(N-1);
```

```
    x1(:,1) = x10(1:2,1);
    for i = 1:N-1
        x1(:,i+1) = [ v*dt*cos(u1_f(i)) + x1(1,i);
                      v*dt*sin(u1_f(i)) + x1(2,i)];
    end
    x1_f = x1(:,N); % Final Position
    % Calculate (N-1) derivatives of the path
    for i = 1:N-1
        u1_d(i,1) = abs(u1_f(i+1)-u1_f(i))/dt;
    end
    % Inequality constraints
    c = [norm(x1_f-x2_f)-d    % Capture condition
         u1_d - u1max];       % Turn rate constraint
    % Equality constraints
    ceq = [x10(3) - x(1)];    % Enforce initial condition of phi1

end
function v = val_fun(x)
    % Value being optimized
    u2 = x;
    % New E path final after changing phi2
    x2(:,1) = x20(1:2,1); % [x, y, t]
    dt = T/(N-1);
    for i = 1:N-1
        x2(:,i+1) = [w*dt*cos(u2(i))+ x2(1,i);
                     w*dt*sin(u2(i))+ x2(2,i)];
    end
    x2_f = x2(:,N);
    % Value function (linearized)
    v = -( c'*(x2_f-e0) );
end
function [c, ceq] = mycon2(x)
    % Values being optimized
    u2   =   x;   % Control u1 at discrete points
    % Position of E at each time step
    x2(:,1) = x20(1:2,1); % [x, y, t]
    dt = T/(N-1);
    for i = 1:N-1
        x2(:,i+1) = [w*dt*cos(u2(i))+ x2(1,i);
                     w*dt*sin(u2(i))+ x2(2,i)];
    end
    x2_f = x2(:,N);
    % Calculate (N-1) derivatives of the path
    for i = 1:N-1
        u2_d(i,1) = abs(u2(i+1)-u2(i))/dt;
    end
    % Inequality constraints
    c = [u2_d - u2max];        % Constraint on E turn rate
    % Equality constraints
    ceq = [x20(3) - x(1)];      % Enforce initial condition of phi2
end
end
```

A.3 Neural Nets

A.3.1 Optimization Framework

opt_football.m

```
global w1 w2 w3 w4 w5
close all
X = [  0     5 0 0
      .5  4.75 0 0
     -.5  4.75 0 0
       1    2 0 0
      -1    2 0 0
      -2    4 0 0
       2    4 0 0]';
t = 0:0.01:7;
options = optimset('GradObj','on','Display','on','LevenbergMarquardt', ...
    'on','DerivativeCheck','off','LargeScale','off','Diagnostics',...
    'on','HessUpdate','bfgs','TolFun',1e-9,'TolX',1e-9,'MaxIter',250);
if 1
    nwe1 = MakeMexFunctionsWin3('footballE1')
    s = 0.05;
    w0 = randn(nwe1,1)*s;

    [We, fval] = fminunc(@(w)myfunTGE1b(w,X,t,'footballE1'),w0,options);

    We =  w1;
    x = X(:,1);
    [J,xt,u] = footballE1(x,We,t);
    figure
    clf
    plot(xt(1,:),xt(2,:),'b--','LineWidth',1.5);
    hold on
    plot(xt(3,:),xt(4,:),'r:','LineWidth',1.5);
    axis equal
    hold off
end
if 0
    nwp = MakeMexFunctionsWin4('footballP')
    s = 0.05;
    w0p = randn(nwp/2,1)*s;
    w0 = [w0p ; We];
    %w0 = w1;
    [W, fval] = fminunc(@(w)myfunTGP(w,X,t,'footballP'),w0,options);
    Wp = w1(1:nwp/2);
end
if 0
    nwe2 = MakeMexFunctionsWin4('footballE2')
    w0  = [Wp; We];
    %w0 = w1;
    [W, fval] = fminunc(@(w)myfunTGE2(w,X,t,'footballE2'),w0,options);
    We2 = w1(nwe2/2+1:end);
end
if 0
```

```
    w = w1;
    x = X(:,1);
    [J,xt,u] = footballE2(x,w,t);
    plot(xt(1,:),xt(2,:),'b--','LineWidth',1.5);
    hold on
    plot(xt(3,:),xt(4,:),'r:','LineWidth',1.5);
    axis equal
    hold off
end
```

A.3.2 Mex function build

```
function y = MakeMexFunctionsWin3(name,varargin)
%Y = MakeMexFunctionsWin3(NAME);
%Y = MakeMexFunctionsWin3(NAME,APP);
%Creates MEX functions based on files stored in the directory referred to
%by NAME.  In general, this funcion will look for a file named
%'dynamics.F90' (and a few other files as well), if a second input is
%given, the file ['dynamics' APP '.F90'] will be used.
%Examples:
%name = 'pendulum';  pendulum.F
%name = 'glider1'; app = '_con'; glider1_con.F90
%
if nargin == 1;
    app = '';
elseif nargin ==2;
    app = varargin{1};
end;
%function y = MakeMexFunctionsWin(name)
%
%   Based on Prof. Steven Hall's MakeMexFunctions.m
%
%   Function that makes the appropriate mex functions in the directory
%   'dir' to allow optimization of a neural net control strategy
%
%   The following files must be in the directory 'name':
%
%       parameters.m      parameters for neural net controller, dynamic system
%       dynamics.F90
%
cr = [' \' char(10)];
sp = [' '];
%...Set up the directory names used throughout
%basedir     = [dirname('MakeMexFunctionsWin') '\']; %apparently DIRNAME no
%longer works in Matlab newer than 7.4
basedir     = [fileparts(which('MakeMexFunctionsWin3')) ];
dynamicsdir = [basedir     '\..\' name ];
builddir    = [dynamicsdir '\build'];
tapdir      = ['C:\tapenade3.1'];
fortrandir  = [basedir      '\..\FortranTemplates'];
stackdir    = [fortrandir   '\ADFirstAidKit'];
%Clean up directories
curdir = pwd;
```

```
cd(tapdir);
fclose all;
cd(builddir);
fclose all;
cd(curdir);
%Read data from the parameter file
%Change from MakeMexFunctions.m:
%Instead of copying parameter.m to basedir, just run it from the
%dynamicsdir
disp('Reading from parameter file ...')
cd(dynamicsdir);
parameters;  %Runs parameter.m to load needed parameters into workspace
cd(curdir);
%Using data from the parameter file, set up the neural net controller
disp('Generating neural net program ...');
[s,nw] = makenn1(nInputs,nNeurons,neuronType);
file1 = [builddir '\nn.F90'];
fclose all;
fid = fopen(file1,'w');
if fid == 0
    disp(['Couldn''t open file ' file1])
end
n = fwrite(fid,s);
if n ~= length(s)
    disp(['Couldn''t write to file ' file1])
end;
%-------------------------------------------------------------------------
% We need to find the adjoint code, using TAPENADE. To do this, we
% first preprocess the fortran code, since TAPENADE doesn't recognize
% compiler directives. So first compile all the files with the -E compiler
% directive, to replace all the symbolic values with numeric values.
disp('Stripping compiler directives from code ...')
file2 = [fortrandir     '\main.F90'];
file3 = [dynamicsdir    '\dynamics' app '.F90'];
file4 = [builddir       '\nn.F90'];
file5 = [builddir       '\' name app '.f90'];
%Combine RK integration routine (main.F90), equation of motion for the
%problem (dynamics.F90), and the neural net (nn.F90) into one file.
command = ['gfortran -E -DNX=' num2str(nx) ' -DNU=' num2str(nu) ' -DNW=' ...
    num2str(nw) ' -DNY=' num2str(ny) sp '"' file2 '"' sp '"' file3 '"' sp '"' file4 '"' sp ..
    ' > ' '"' file5 '"'];
disp(' ');
disp(command);
[status,result] = system(command);
if status
    disp(result);
    return
end
% Now we must comment out remaining compiler directives
fclose all;
fid = fopen(file5,'r');
s = fread(fid,'*char')';
fclose all;
s = strrep(s,'#','!#');
```

```
fid = fopen(file5,'w');
n = fwrite(fid,s);
if n ~= length(s)
    disp(['Failed to write to file ' file5])
end
%-------------------------------------------------------------
% Now we are ready to call TAPENADE
command = ['"' tapdir '\bin\tapenade' '"'...
    ' -inputlanguage fortran95 ' ...
    '-outvars "j" -vars "alpha w" -head main4 -reverse -o ' name app ...
    ' -outputlanguage fortran90 -O ' '"' builddir '"' ' -diffvarname '...
    '_b -i4 -dr8 -r4 ' '"' file5 '"'];
% Took out -parserfileseparator "\"  from command
disp(' ');
disp(command);
[status,result]=system(command);
disp(' ')
disp(result)
if status, return, end
%file1 = [fortrandir 'testmain.F90'];
file3 = [builddir    '\' name app '.F90'];
file4 = [builddir    '\' name app '_b.F90'];
%file5 = [builddir    name];
file6 = [stackdir    '\' 'adBuffer.f'];
file7 = [stackdir    '\' 'adStackNoUnderscore.c'];
% Do the MEX command for the forward case.
file1 = [fortrandir  '\' 'gateway1win1.F90'];
%Also need mexinterface.mod compiled from mexinterface_c.f90 ...
%Make sure mexinterface.mod is accessible by MEX, and is up-to-date
command = ['mex '...
    '-DNX=' num2str(nx) ' -DNU=' num2str(nu) ' -DNW=' num2str(nw) ...
    ' -DNY=' num2str(ny) ' -output ' name app ...
    ' ' '"' file1 '"' ' ' '"' file3 '"' ];
disp(command);
eval(command);
% Do the MEX command for the reverse case.
file1 = [fortrandir  '\' 'gateway2win1.F90'];
command = ['mex '...
    '-DNX=' num2str(nx) ' -DNU=' num2str(nu) ' -DNW=' num2str(nw) ...
    ' -DNY=' num2str(ny) ' -output ' name app '_b' ...
    ' ' '"' file1 '"' ' ' '"' file3 '"' ' ' '"' file4 '"' ' ' '"' ...
file6 '"' ' ' '"' file7 '"'];
disp(command);
eval(command);
y=nw;
fclose('all');
return
```

A.3.3 Minimization function

```
function [F,G] = myfunTGP1(w,X,t,func)
global w1 w2 w3 w4 w5
w5 = w4;
```

```
w4 = w3;
w3 = w2;
w2 = w1;
w1 = w;
persistent N
if length(N) == 0, N = 0; end
N = N+1;
W = w;
F = 0;
G = 0;
if nargout ==1
    for i=1:size(X,2);
        x = X(:,i);
        [f] = eval([func '_b(x,w,t)']);
        F = F + f;
    end
else
    for i=1:size(X,2);
        x = X(:,i);
        [f,g] = eval([func '_b(x,w,t)']);
        F = F + f;
        G = G + g;
    end
end
if mod(N,10) == 0
    fprintf('%i,  %f\n',nargout,F)
    clf
    for i=1:size(X,2)
        x = X(:,i);
        [J,xt,u] = eval([func '(x,w,t)']);
        plot(xt(1,:),xt(2,:),'b--','LineWidth',1.5);
        hold on
        plot(xt(3,:),xt(4,:),'r:','LineWidth',1.5);
        axis equal
    end
    hold off
    drawnow
end
return
```

A.3.4 Dynamics

```
# if NX-4
error: Should have NX = 4
# endif
# if NU-1
error: Should have NU = 1
# endif
# if NY-2
error: Should have NY = 2
# endif
!This file contains the following:
!  cost
```

```
!  f dynamics of the pursuer
subroutine cost(x,u,L)
implicit none
real(8) :: x(NX), u, L
    intrinsic sqrt
    L = 0
return
end subroutine
subroutine general(x,u,nt,t,C)
    implicit none
!   !Input variables
    integer :: nt, i
    real(8) :: x(NX,nt), u(nt), t(nt)
!   !Output variables
    real(8) :: C
!   !Working variables
    real(8) :: EYdist, distPE, distEB, beta
    real(8), parameter :: alpha1 = 50, CR = .5
    real(8), parameter :: alpha2 = 5, OB = 1.0, width = 20
    intrinsic sqrt, exp, abs

    EYdist = x(4,nt);
    do i=nt-1,1,-1
        beta = 0
        distPE = sqrt((x(4,i)-x(2,i))**2+(x(3,i)-x(1,i))**2)
        distEB = width/2-abs(x(3,i))
        if (distEB < 0) then
            distEB = 0
        end if
        if (distPE < CR) then
            beta = exp(-alpha1*(distPE-CR))-1+alpha1*(distPE-CR)
        end if
        if (distEB < OB) then
            beta = exp(-alpha2*(distEB-OB))-1+alpha2*(distEB-OB)
        end if
        EYdist = (beta*x(4,i)+EYdist)/(beta+1)
    end do
    C = -EYdist
    return
end subroutine
subroutine terminal(x,u,t,dJ)
implicit none
real*8 :: x(NX), u, t, dJ

    dJ = 0
return
end subroutine
!*******************************************************************************
subroutine f(x,w,x_dot,u2)
implicit none
real(8) :: x(NX), w(NW), x_dot(NX), y(NY), T
real(8), parameter :: v1 = 1.0D0, v2 = 1.1D0, PI = 3.14159265
    real(8), parameter :: u2max = 4, width = 12.0
real(8) :: u2, cosu1, sinu1, quo
```

```
    real(8) :: xdisp, ydisp, distPE
intrinsic sin, cos, atan2, sqrt, abs, acos
    xdisp = x(3)-x(1)
    ydisp = x(4)-x(2)
    distPE = sqrt(xdisp**2+ydisp**2)
! NN control input
    y(1) = xdisp
    y(2) = ydisp
call nn(y,u2,w)
    u2 = u2*2/PI*u2max
! Assigned Control input for Evader
    quo = -u2-(atan2(ydisp,-xdisp)+PI/2)

    cosu1 = cos(quo)
    sinu1 = sin(quo)
! Dynamics
    x_dot(1) = v1*cosu1
    x_dot(2) = v1*sinu1
    x_dot(3) = v2*cos(u2)
    x_dot(4) = v2*sin(u2)
return
end subroutine
```

A.3.5 Parameters

```
% parameters.m
global nx nu ny nInputs nNeurons neuronType
%
% Parameters of dynamic system and controller
%
nx = 4;
nu = 1;
ny = 2;
%
% Parameters of neural net controller
%
nInputs = ny;
nNeurons = [5 7 7 1];
neuronType = {'atan';'atan';'atan';'atan'};
```

A.3.6 Main file

```
subroutine main1(x0,w,t,nt,J)
     implicit none
!  !Input variables
integer :: nt
real*8  :: x0(NX), w(NW), t(nt)
!  !Output variables
real*8 :: J
!  !Working variables
integer  :: i
real*8   :: dt, x1(NX), xf(NX), u1(NU), dJ, x(NX,nt), u(NU,nt), C
!  !Initialize cost, state vector
```

```
J = 0.
!  x1 = x0
    x(:,1) = x0
!  !Integrate the dynamics and cost forward in time
do i=1,nt-1
!  !Find the time increment to pass to Runge-Kutta routine
dt = t(i+1)-t(i)
!  !Do the Runge-Kutta step
x1 = x(:,i)
call rk(x1,w,dt,xf,dJ,u1)
x(:,i+1) = xf
        u(:,i) = u1
!  x1 = xf
J = J + dJ
end do
    u(:,nt) = u(:,nt-1) !**** FIX! *****
    call terminal(x(:,nt),u(:,nt),t(nt),dJ)
    J = J + dJ
    call general(x,u,nt,t,C)
    J = J + C
return
end subroutine
!=====================================================================
subroutine main2(x0,w,t,nt,J,x)
implicit none
!  !Input variables
integer :: nt
real*8  :: x0(NX), w(NW), t(nt)
!  !Output variables
real*8 :: J, x(NX,nt)
!  !Working variables
integer  :: i
real*8   :: dt, x1(NX), xf(NX), u1(NU), dJ, u(NU,nt), C
!  !Initialize cost, state vector
J = 0.
x(:,1) = x0
!  !Integrate the dynamics and cost forward in time
do i=1,nt-1
!  !Find the time increment to pass to Runge-Kutta routine
dt = t(i+1)-t(i)
!Do the Runge-Kutta step
x1 = x(:,i)
call rk(x1,w,dt,xf,dJ,u1)
x(:,i+1) = xf
u(:,i) = u1
        J = J + dJ
end do
    u(:,nt) = u(:,nt-1) !**** FIX! *****
    call terminal(x(:,nt),u(:,nt),t(nt),dJ)
    J = J + dJ
    call general(x,u,nt,t,C)
    J = J + C
return
end subroutine
```

```
!=============================================================
subroutine main3(x0,w,t,nt,J,x,u)
implicit none
!  !Input variables
integer :: nt
real*8  :: x0(NX), w(NW), t(nt)
!  !Output variables
real*8 :: J, x(NX,nt), u(NU,nt)
!  !Working variables
integer  :: i
real*8   :: dt, x1(NX), xf(NX), u1(NU), dJ, C
!  !Initialize cost, state vector
J = 0.
x(:,1) = x0
!  !Integrate the dynamics and cost forward in time
do i=1,nt-1
!  !Find the time increment to pass to Runge-Kutta routine
dt = t(i+1)-t(i)
!  !Do the Runge-Kutta step
x1 = x(:,i)
call rk(x1,w,dt,xf,dJ,u1)
x(:,i+1) = xf
u(:,i) = u1
J = J + dJ
end do
u(:,nt) = u(:,nt-1) !**** FIX! *****
    call terminal(x(:,nt),u(:,nt),t(nt),dJ)
    J = J + dJ
    call general(x,u,nt,t,C)
    J = J + C
return
end subroutine
!=============================================================
subroutine main4(x0,w,t,nt,J,alpha)
     implicit none
!  !Input variables
integer :: nt
real*8  :: x0(NX), w(NW), t(nt), alpha
!  !Output variables
real*8 :: J
!  !Working variables
integer  :: i
real*8   :: dt, x1(NX), xf(NX), u1(NU), dJ, x(NX,nt), u(NU,nt), C
!  !Initialize cost, state vector
J = 0.
x(:,1) = x0
!  !Integrate the dynamics and cost forward in time
do i=1,nt-1
!  !Find the time increment to pass to Runge-Kutta routine
dt = t(i+1)-t(i)
!  !Do the Runge-Kutta step
  x1 = x(:,i)
call rk(x1,w,dt,xf,dJ,u1)
  x(:,i+1) = xf
```

```
        u(:,i) = u1
J = J + dJ
end do
    u(:,nt) = u(:,nt-1)
    call terminal(x(:,nt),u(:,nt),t(nt),dJ)
    J = J + dJ
call general(x,u,nt,t,C)
    J = J + C
    J = J*alpha

    return
end subroutine
!**************************************************************************
subroutine rk(x1,w,dt,xf,dJ,u1)
implicit none
!  !Input variables
real*8  :: x1(NX), w(NW), dt
!  !Output variables
real*8  :: xf(NX), dJ, u1(NU)
!  !Working variables
real*8 :: x2(NX), x3(NX), x4(NX)
real*8 :: xdot1(NX), xdot2(NX), xdot3(NX), xdot4(NX)
real*8 :: J1, J2, J3, J4, u2(NU), u3(NU), u4(NU), L1, L2, L3, L4
!  !Find xdot and L (Jdot) at each sample point
call f(x1,w,xdot1,u1)
call cost(x1,u1,L1)
x2 = x1 + xdot1 * (dt/2.)
call f(x2,w,xdot2,u2)
call cost(x2,u2,L2)
x3 = x1 + xdot2 * (dt/2.)
call f(x3,w,xdot3,u3)
call cost(x3,u3,L3)
x4 = x1 + xdot3 * dt
call f(x4,w,xdot4,u4)
call cost(x4,u4,L4)
!  !Find the final point, and increment in cost
xf = x1 + (xdot1 + 2.*xdot2 + 2.*xdot3 + xdot4) * (dt/6.)
dJ = (L1 + 2.*L2 + 2.*L3 + L4) * (dt/6.)
return
end subroutine
```

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] R. Isaacs, "Differential games, I: Introduction," Research Memorandum RM-1391, RAND Corporation, Santa Monica, CA, 1954.

[2] R. Isaacs, "Differential games, II: The definition and formulation," Research Memorandum RM-1399, RAND Corporation, Santa Monica, CA, 1954.

[3] R. Isaacs, "Differential games, III: The basic principles of the solution process," Research Memorandum RM-1411, RAND Corporation, Santa Monica, CA, 1954.

[4] R. Isaacs, "Differential games, IV: Mainly examples," Research Memorandum RM-1486, RAND Corporation, Santa Monica, CA, 1955.

[5] R. Isaacs, "Games of pursuit," Paper P-257, RAND Corporation, Santa Monica, CA, 1951.

[6] R. Isaacs, *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization.* New York: John Wiley and Sons, Inc., 1965.

[7] Y. Ho, "Differential games," *IEEE Transactions on Automatic Control*, vol. 10, no. 4, pp. 501–503, 1965.

[8] A. Merz, "The game of two identical cars," *Journal of Optimization Theory and Applications*, vol. 9, no. 5, pp. 324–343, 1972.

[9] A. Merz, "The homicidal chauffeur," *AIAA Journal*, vol. 12, no. 3, pp. 259–260, 1974.

[10] A. Merz and D. Hague, "Coplanar tail-chase aerial combat as a differential game," *AIAA Journal*, vol. 15, no. 10, pp. 1419–1423, 1977.

[11] G. Olsder and J. Breakwell, "Role determination in an aerial dogfight," *International Journal of Game Theory*, vol. 3, no. 1, pp. 47–66, 1974.

[12] J. Breakwell and A. Merz, "Football as a differential game," *Journal of Guidance, Control, and Dynamics*, vol. 15, no. 5, pp. 1292–1294, 1991.

[13] A. E. Bryson and Y.-C. Ho, *Applied Optimal Control: Optimization, Estimation, and Control.* Washington: Hemisphere Pub. Corp., 1975.

[14] Y. Ho, A. Bryson, and S. Baron, "Differential games and optimal pursuit-evasion strategies," *IEEE Transactions on Automatic Control*, vol. 10, no. 4, pp. 385–389, 1965.

[15] T. Başar and G. J. Olsder, *Dynamic Noncooperative Game Theory*. San Diego, CA: Academic Press, second ed., 1995.

[16] T. Raivio and H. Ehtamo, *On the numerical solution of a class of pursuit-evasion games*, vol. 5 of *Annals of the International Society of Dynamic Games*. 2000.

[17] J. V. Breakwell and A. W. Merz, "Minimum required capture radius in a coplanar model of the aerial combat problem," *AIAA Journal*, vol. 15, no. 8, pp. 1089–1094, 1977.

[18] N. Farber and J. Shinar, "An approximate feedback solution of a variable speed non-linear pursuit-evasion game between two airplanes in a horizontal plane," in *Atmosphereic Flight Mechanics Conference*, (Danvers, MA), pp. 337–347, AIAA, August 1980.

[19] N. Rajan, U. Prasad, and N. Rao, "Pursuit-evasion of two aircraft in a horizontal plane," *Journal of Guidance, Control, and Dynamics*, vol. 3, no. 3, pp. 261–267, 1980.

[20] M. Guelman, J. Shinar, and A. Green, "Qualitative study of a planar pursuit evasion game in the atmosphere," in *AIAA Guidance, Navigation, and Control Conference*, (Minneapolis, MN), pp. 874–882, AIAA, August 1988.

[21] J. P. How, "16.323 lecture 17: Dido, a pseudospectral method for solving optimal control problems." MIT Open Courseware, 2006. http://ocw.mit.edu/.

[22] M. Breitner, H. Pesch, and W. Grimm, "Complex differential games of pursuit-evasion type with state constraints, part 1: Necessary conditions for open-loop strategies," *Journal of Optimization Theory and Applications*, vol. 78, no. 3, pp. 419–441, 1993.

[23] L. F. Shampine and J. Kierzenka, "Solving boundary value problems for ordinary differential equations in MATLAB with bvp4c." MATLAB Central File Exchange, October 2000. http://www.mathworks.com/matlabcentral/fileexchange/3819.

[24] R. G. Melton, "Direct methods of optimal control for space trajectory optimization." Penn State University Computational Science Invited Lectures, February 2007. http://www.csci.psu.edu/seminars/springnotes/2007/melton2007.pdf.

[25] B. Järmark, A. Merz, and J. Breakwell, "The variable speed tail-chase aerial combat problem," *Journal of Guidance, Control, and Dynamics*, vol. 4, no. 3, pp. 323–328, 1981.

[26] R. E. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[27] M. Breitner, H. Pesch, and W. Grimm, "Complex differential games of pursuit-evasion type with state constraints, part 2: Numerical computation of open-loop strategies," *Journal of Optimization Theory and Applications*, vol. 78, no. 3, pp. 443–463, 1993.

[28] T. Raivio, "Capture set computation of an optimally guided missile," *Journal of Guidance, Control, and Dynamics*, vol. 24, no. 6, pp. 1167–1175, 2001.

[29] T. Raivio and H. Ehtamo, "Visual aircraft identification as a pursuit-evasion game," *Journal of Guidance, Control, and Dynamics*, vol. 23, no. 4, pp. 701–708, 2000.

[30] K. Horie, *Collocation with Nonlinear Programming for Two-Sided Flight Path Optimization*. PhD thesis, Dept. of Aeronautical and Astronautical Engineering, Univ. of Illinois, Urbana, IL, February 2002.

[31] K. Horie and B. A. Conway, "Genetic algorithm preprocessing for numerical solution of differential games problems," *Journal of Guidance, Control, and Dynamics*, vol. 27, no. 6, pp. 1075–1078, 2004.

[32] K. Horie and B. A. Conway, "Optimal fighter pursuit-evasion maneuvers found via two-sided optimization," *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 1, pp. 105–112, 2006.

[33] MauroPontani and B. A. Conway, "Optimal interception of evasive missile warheads: Numerical solution of the differential game," *Journal of Guidance, Control, and Dynamics*, vol. 31, no. 4, pp. 1111–1122, 2008.

[34] A. W. Merz, *The Homicidal Chauffeur: A Differential Game*. PhD thesis, Stanford University, California, 1971.

[35] V. Patsko and V. Turova, "Numerical study of the homicidal chauffeur game," in *Proceedings of the Ninth International Colloquium on Differential Equations* (D. Bainov, ed.), (Plovdiv, Bulgaria), 18-23 August 1998, 1999.

[36] T. Raivio, *Computational Methods for Dynamic Optimization and Pursuit-Evasion Games*. PhD thesis, Helsinki University of Technology, Espoo, Finland, March 2000.

[37] Y. Yavin and R. D. Villiers, "Game of two cars: Case of variable speed," *Journal of Optimization Theory and Applications*, vol. 60, no. 2, pp. 327–339, 1989.

[38] R. S. S. W. Thomas Miller III and P. J. Werbos, eds., *Neural Networks for Control*. London: MIT Press, 1990.

[39] M. T. Hagan and H. B. Demuth, "Neural networks for control," in *1999 American Control Conference*, (San Diego, CA), pp. 1642–1656, June 1999.

[40] L. Hascoët, "Tapenade: a tool for automatic differentiation of programs," in *Proceedings of 4^{th} European Congress on Computational Methods, ECCOMAS'2004, Jyvaskyla, Finland*, 2004.

[41] L. Hascoët and V. Pascual, "Tapenade 2.1 user's guide," Technical Report 0300, INRIA, 2004.

[42] L. Hascoët, "Automatic differentiation by program transformation," April 2007. http://www-sop.inria.fr/tropics/tropics/supportCoursDA.pdf.

[43] B. Järmark and H. Bengtsson, *Computational Optimal Control*, ch. "Near-Optimal Flight Trajectories Generated by Neural Networks", pp. 319–328. Birkhäuser, 1994.

[44] M. Brett, *Compiling Matlab mex files with gcc for Windows*. http://gnumex.sourceforge.net/, 2005.

[45] The gfortran team, *Using GNU Fortran*. Free Software Foundation, Boston, 2008.

[46] J. P. How, "16.323 lecture 9: Constrained optimal control." MIT OpenCourseWare, 2008. http://ocw.mit.edu/.

[47] P. Zarchan, *Tactical and Strategic Missile Guidance*, vol. 219 of *Progress in Astronautics and Aeronautics: A Volume in the AIAA Tactical Missile Series*. Reston, VA: AIAA, fifth ed., 2007.

[48] J. P. How, "16.323 lecture 7: Numerical solution in matlab." MIT OpenCourseWare, 2008. http://ocw.mit.edu/.

[49] T. Shima and J. Shinar, "Time-varying linear pursuit-evasion game," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 3, pp. 425–432, 2002.

[50] D. A. Roberts and R. C. Montgomery, "Development and application of a gradient method for solving differential games," NASA Technical Note D-6502, Langley Research Center, Hampton, VA, November 1971.

[51] C. Hillberg and B. Järmark, "Pursuit-evasion between two realistic aircraft," *Journal of Guidance, Control, and Dynamics*, vol. 7, no. 6, pp. 690–694, 1983.

[52] J. Przemieniecki, *Mathematical Methods in Defense Anayses*. AIAA Education Series, Reston, VA: AIAA Inc., third ed., 2000.

[53] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[54] R. Giering and T. Kaminski, "Recipes for adjoint code construction," *ACM Transactions on Mathematical Software*, vol. 24, no. 4, pp. 437–474, 1998.