Simultaneous Localization and Planning of Cooperative Air Munitions via Dynamic Programming

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Emily A. Doucette

Certificate of Approval:

John E. Cochran
Department Head and Professor
Aerospace Engineering

Andrew J. Sinclair, Chair
Assistant Professor
Aerospace Engineering

David A. Cicci
Professor
Aerospace Engineering

Joe F. Pittman
Interim Dean
Graduate School

Simultaneous Localization and Planning of Cooperative Air Munitions via Dynamic Programming

Emily A. Doucette

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
May 10, 2008

Simultaneous Localization and Planning of Cooperative Air Munitions via Dynamic Programming

Emily A. Doucette

_____

Signature of Author

_____

Date of Graduation

Vita

Emily Ann Doucette, daughter of Roland Jr. and Judith (Cusimano) Doucette, was born September 30, 1984, in Metairie, Louisiana. She graduated from Pope John Paul II Catholic High School as Valedictorian in 2002. She attended Auburn University in Auburn, Alabama, where she was chapter president of her sorority, Alpha Xi Delta, and graduated magna cum laude with a Bachelor of Aerospace Engineering degree in May 2006. She entered Graduate School at Auburn University, in August 2006.

Thesis Abstract

Simultaneous Localization and Planning of Cooperative Air Munitions via Dynamic Programming

Emily A. Doucette

Master of Science, May 10, 2008
(B.A.E., Auburn University, 2006)

80 Typed Pages

Directed by Andrew J. Sinclair

This work centers on the real-time trajectory planning for the cooperative control of two aerial munitions in a planar setting. One munition strikes the stationary ground target and the other will serve as an observer. Sensor information from each munition is assumed available, and the individual target-location estimates are fused in a weighted least squares solution. The variance of this combined estimate is used to define a cost function. The problem is posed to design munition trajectories that minimize this cost function. This work describes the solution of the problem by a dynamic-programming method. The dynamic-programming method establishes a set of grid points for each munition to traverse based on the initial position of the munition relative to the target. The method determines the optimal path along those points to minimize the value of the cost function and consequently decrease the value of uncertainty in the estimate of the target location. The method is validated by comparison to known solutions computed by a variational method for sample solutions.

Numerical solutions are presented along with computational run times to indicate that this method is effective in trajectory design and target location estimation.

Acknowledgments

The author would like to acknowledge Dr. Andrew Sinclair for his guidance and support throughout this work. The author would also like to thank Dr. David Jeffcoat and the Munitions Directorate of Air Force Research Laboratory for the opportunity to investigate this problem, and the Aerospace Engineering Department at Auburn University for its support and financial assistance. Finally, the author would like to bestow many thanks to her family and friends for their constant encouragement and unwavering support.

Style manual or journal used Journal of Approximation Theory (together with the style known as "aums"). Bibliograpy follows van Leunen's *A Handbook for Scholars.*

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `aums.sty`.

Table of Contents

List of Figures x

List of Tables xi

1 Introduction 1
1.1 Literature Review 2
1.2 Optimal Control Theory 5
1.2.1 Convexity 7
1.3 Dynamic-Programming and Graph Theory 9

2 Development and Validation of Dynamic-Programming Routine 12
2.1 Method Development 12
2.2 Validation Process 12

3 Problem Definition 17

4 Application of Dynamic-Programming Approach 25

5 Estimation Performance 30
5.1 Estimation Results 31
5.1.1 Problem 1 31
5.1.2 Problem 2 32
5.2 Discussion of Results 33

6 Conclusions 34

Bibliography 36

Appendices 38

A FORTRAN Code Used for Dynamic-Programming 39

B MATLAB Code Used for Plots and Error Estimation 65

List of Figures

1.1 (a) Convex set (b) Nonconvex set 8

1.2 Convex function f 9

1.3 Results from Textbook Example Validation 11

2.1 Results from Single Degree of Freedom Validation 16

3.1 Measurement of the target by the ith munition and the associated error probability ellipse. 19

3.2 Investigation of cost function convexity with constant t_F 24

3.3 Investigation of cost function convexity with constant heading, ψ_1 and ψ_2 24

4.1 Example of (a) path grids and (b) DAG where $n = 64$ and $m = 168$. 26

4.2 DAG where $n = 64$ and $m = 168$ 27

4.3 Problem 1 SLAP trajectories from (a) variational and (b) DP methods. 28

4.4 Problem 2 SLAP trajectories from (a) variational and (b) DP methods. 28

5.1 Estimation errors using (a) Variational Method and (b) DP trajectories with $x_2(0) = 100\,\text{ft}$, and $y_2(0) = -2000\,\text{ft}$. 32

5.2 Estimation errors using (a) Variational Method and (b) DP trajectories with $x_2(0) = 0\,\text{ft}$, and $y_2(0) = 2000\,\text{ft}$. 33

List of Tables

3.1 Cost of Trial Solutions. 23

4.1 Cost for sample SLAP trajectories. 29

5.1 Area of one-sigma uncertainty ellipse. 32

Chapter 1

Introduction

Research is in progress on the cooperative control of air armament designed to detect, identify, and attack ground targets with minimal operator oversight. One class of this type of armament is wide-area search munitions, which can be deployed in an area of unknown targets. Current development is focused on possibilities of enhancing munition capabilities through cooperative control. This problem of designing an attack trajectory that enhances the ability to estimate the target location will be referred to as simultaneous localization and planning (SLAP). Previous work by Sinclair et al. solved the SLAP problem by variational methods [1]. This work presents a method to drastically reduce the computational expense incurred in variational approaches to the SLAP problem by use of the dynamic-programming (DP) method. The methods presented in this work will be illustrated for a planar problem with two munitions and one stationary target.

In Chapter 2, the steps taken during the development of the dynamic-programming routine will be discussed and the results of the validation process will be analyzed. Chapter 3 presents models for the munition motion and sensor performance. Next in Chapter 4, the SLAP trajectory design is posed as a DP problem. The performance cost sensitivity to grid resolution is then investigated. Finally, the performance of a target-location estimation algorithm is evaluated along the SLAP trajectories and compared to alternative trajectories. This chapter presents past investigations of cooperative control problems that played a role in the motivation for the DP approach,

an overview of general optimal control problems, the dynamic-programming method, and graph theory used in this work.

1.1 Literature Review

Several aspects of the cooperative control of unmanned air vehicles have been investigated. Important work exists in the literature on the problem of cooperative search. In 2002, Chandler et al. presented the cooperative use of eight vehicles to maximize the probability of correct target classification through the development of templates and combination of views over various aspect angles. In this work, a hierarchical distributed decision system is used to sequentially perform task assignments through a market based bidding scheme, coordinate cooperative tasks, and plan the optimal paths to view the targets. This method demonstrated significant improvement in classification performance than that achieved when the vehicles operate independently [2]. The effect of cueing on the probability of target detection was investigated by use of two searchers through the derivation from first principles using a Markov chain analysis [3]. Additionally in 2005, Frew et al. developed guidance laws for tracking a ground vehicle by an autonomous team of two air vehicles by the construction of Lyapunov vector fields. This method used a second Lyaponuv vector field to produce advantageous phasing of aircraft to follow an agile target [4].

The problem of cooperative search is related to that of the often investigated problem of designing optimal trajectories for single observers. Fawcett used the Cramer-Rao lower bound to investigate the effect of course maneuvers on bearings-only range estimation. Although optimal observer trajectories were not designed, it was shown by Monte Carlo simulations that for small amounts of noise, the variance

of the maximum likelihood estimator range estimate follows the Cramer-Rao lower bound [5]. Hammel et al. derived observer paths in the context of continuous-time bearing measurements with a performance index based on the determinant of the Fisher information matrix (FIM). Because that formulation resulted in an optimal control problem not suited for standard solution methods based on the minimum principle, an approximate numerical solution based on the direct maximization of the FIM was used [6]. Oshman and Davidson investigated Hammel's problem by use of direct optimal control numerical schemes, including two gradient-based numerical procedures and differential inclusion. In addition to improving results, state constraints were also imposed on the observer trajectory to simulate the defense system of a target [7].

Logothetis et al. compared two groups of suboptimal optimization techniques for computing observer trajectories in bearings-only tracking. It was shown that the approximate DP strategy yielded smaller range errors than one-step-ahead suboptimal strategies, the smallest range error being produced by the DP method that minimized the trace of the error covariance matrix at each time instant [8]. Passerieux and Van Cappel generalized the constant speed observer trajectories to nonrectilinear segments and computed the true optimal observer maneuvers for bearings-only tracking. The Euler equations were established and resolved by a combination of analytical and numerical methods and the performance index was based on minimizing an accuracy criterion from the FIM [9].

Additional single observer trajectory optimization problems involve vision based guidance. Frew and Rock recognized the strong relationship between the performance of monocular vision based target tracking and camera motion. Camera paths were

designed in real time based on the predicted target state error covariance using a pyramid, breadth-first search algorithm [10]. Wantanabe et al. studied vehicle guidance design that utilized sensor trajectory optimization for monocular vision based guidance. This method improved mission performance by minimizing a weighted sum of guidance performance cost, control cost, and estimation cost [11].

The problem of planning optimal trajectories for cooperative observers has been studied using collocation. Ousingsawat and Campbell developed a receding horizon optimal control formulation to solve for trajectories that maximize information gathered by the sensing vehicles. Important notes in this work were that nonsymmetric sensors tend to triangulate as they approach the target and that the addition of a third vehicle does not yield large performance improvement but does add redundancy [12]. Grocholsky also defined information gathering as an optimal control problem to quantify his results that explain the coupling and coordination of decentralized decision makers. It was also shown that the use of static information structures lead to sub-optimal yet efficient control strategies for the team [13].

The problem of cooperative attack was previously investigated using variational methods [1]. This method yielded encouraging results, but was too computationally expensive for online implementation. Coupled with the need for reduced computational expense and the results from Logothetis, it was determined that the effectiveness of the dynamic-programming method on the SLAP problem should be investigated.

1.2 Optimal Control Theory

The primary focus of Applied Optimal Control Theory is the analysis and design of complicated dynamic systems and the determination of effective ways to control such systems. The mathematical statement of the optimal control problem consists of descriptions of the system to be controlled, system constraints, the task to be accomplished, and the criterion against which the performance is to be judged and determined optimal [14].

The system to be controlled is expressed in a set of differential equations known as the state equations. The state equation is typically a function of the state, x, the control variable, u, and time, t, as shown in Eq. (1.1). Constraints often exist on allowable values of state and control variables.

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) \tag{1.1}$$

The task is often dictated by a boundary condition on Eq. (1.1), such as a state transition from a known initial state to a specified final state. The task is frequently specified implicitly by the performance criterion, also known as the cost function. The general form of a continuous cost function is given by Eq. (1.2).

$$J = \phi(\boldsymbol{x}(t_F), t_F) + \int_0^{t_F} L(\boldsymbol{x}, \boldsymbol{u}, t)\mathrm{d}t \tag{1.2}$$

In Eq. (1.2), ϕ is a scalar-valued function of the cost associated with the terminal state at time t_F. The cost function L is also a scalar-valued function and depends on the transient states and control effort. These functions are selected to emphasize

the importance of terminal condition, transient behavior, and the expended control effort in the total cost function, J.

The next step to optimization is the definition of the Hamiltonian. This scalar-valued function is important due to Pontryagin's minimum principle that states the optimal control u^*(t) is a member of the admissible control set U which minimizes H at every time. The Lagrange multipliers λ are referred to as costates and measure the sensitivity of the cost to the current value of the states.

$$H = L(\boldsymbol{x}, \boldsymbol{u}, t) + \boldsymbol{\lambda}^{\mathsf{T}} \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) \tag{1.3}$$

After defining the Hamiltonian, Eq. (1.3) is substituted into Eq. (1.2) to form the augmented cost function J_a.

$$J_a = \phi(\boldsymbol{x}(t_F)) + \int_0^{t_F} \left(H(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}, t) - \boldsymbol{\lambda}^{\mathsf{T}} \dot{\boldsymbol{x}} \right) \mathrm{d}t \tag{1.4}$$

Expanding the augmented cost function by taking the variation due to the states $\delta\boldsymbol{x}$ and control $\delta\boldsymbol{u}$ yields δJ_a.

$$\begin{aligned} \delta J_a = & \left(\frac{\partial \phi}{\partial \boldsymbol{x}(t_F)} \right)^{\mathsf{T}} \delta \boldsymbol{x}(t_F) \\ & + \int_0^{t_F} \left[\frac{\partial H}{\partial \boldsymbol{x}}^{\mathsf{T}} \delta \boldsymbol{x} + \frac{\partial H}{\partial \boldsymbol{u}}^{\mathsf{T}} \delta \boldsymbol{u} + \frac{\partial H}{\partial \boldsymbol{\lambda}}^{\mathsf{T}} \delta \boldsymbol{\lambda} - \dot{\boldsymbol{x}}^{\mathsf{T}} \delta \boldsymbol{\lambda} - \boldsymbol{\lambda}^{\mathsf{T}} \delta \dot{\boldsymbol{x}} \right] \mathrm{d}t \end{aligned}$$

This result can be integrated by parts.

$$\delta J_a = \left(\frac{\partial \phi}{\partial \boldsymbol{x}(t_F)}\right)^{\mathsf{T}} \delta \boldsymbol{x}(t_F) - \boldsymbol{\lambda}^{\mathsf{T}} \delta \boldsymbol{x}\Big|_0^{t_F} + \int_0^{t_F} \left[\left(\frac{\partial H}{\partial \boldsymbol{x}} + \dot{\boldsymbol{\lambda}}\right)^{\mathsf{T}} \delta \boldsymbol{x} + \frac{\partial H}{\partial \boldsymbol{u}}^{\mathsf{T}} \delta \boldsymbol{u} + \left(\frac{\partial H}{\partial \boldsymbol{\lambda}} - \dot{\boldsymbol{x}}\right)^{\mathsf{T}} \delta \boldsymbol{\lambda}\right] \mathrm{d}t$$

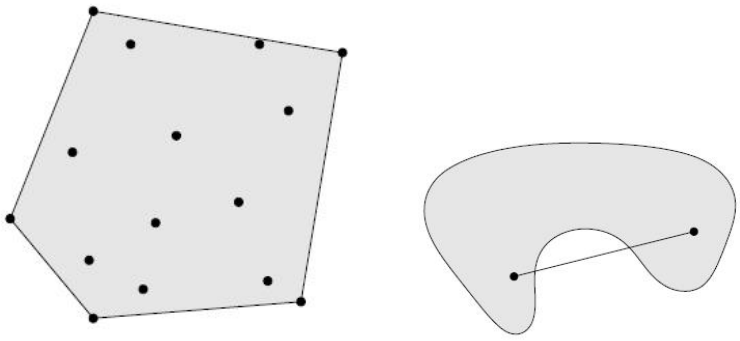
By requiring the expressions in the integrand to go to zero for arbitrary variations $\delta \boldsymbol{x}$ and $\delta \boldsymbol{u}$, necessary conditions known as the Euler-Lagrange equations are formed.

$$\begin{aligned} \dot{\lambda} &= -\frac{\partial H}{\partial x} \\ \frac{\partial H}{\partial u} &= 0 \\ \dot{x} &= \frac{\partial H}{\partial \lambda} \end{aligned} \tag{1.5}$$

The terms outside the integral yield the boundary conditions on the problem. Solution of these necessary and boundary conditions is referred to as solving the optimal control problem by indirect or variational methods.

1.2.1 Convexity

It is important to note the role of convexity in optimal control problems. If an optimal control problem is convex, an optimal solution should yield a global minimum, whereas the solution of a nonconvex optimal control problem may yield only a local minimum. Convex sets and convex functions must be defined to determine the convexity of an optimal control problem.



(a) (b)

Figure 1.1: (a) Convex set (b) Nonconvex set

A set C is convex if a line segment between any two points in the set lies in the set, *i.e.* for any $x_1, x_2 \in C$ and any θ where $0 \leq \theta \leq 1$, convexity requires $\theta x_1 + (1-\theta)x_2 \in C$, as illustrated by Fig. 1.1.

A function f, shown in Fig. 1.2, is convex if and only if:

1. the domain of f is a convex set,
2. and the function evaluation of a linear interpolation of x_1 and x_2 must be less than or equal to the linear interpolation of $f(x_1)$ and $f(x_2)$.

Given an optimization problem of minimizing the objective function $f_0(x)$ subject to inequality constraints $f_i(x) \leq 0$ and equality constraints $a_i^{\mathsf{T}} x = b_i$, the problem is convex if and only if:

1. f_0 and f_i are convex,
2. and the equality constraints are linear.

Given these definitions of convex sets, functions, and optimization problems, the convexity of an optimal control problem may be determined. The general form of

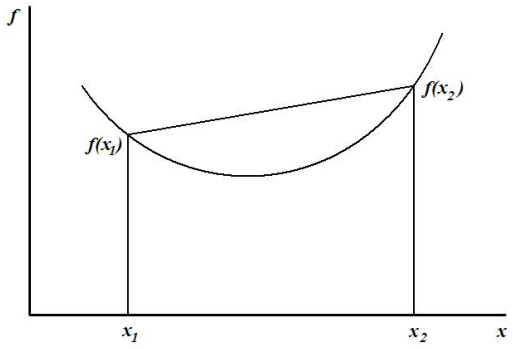


Figure 1.2: Convex function f

an optimal control problem is to minimize the cost function $J(\boldsymbol{x}, \boldsymbol{u}, t)$ subject to state equations $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t)$, equality constraints $\boldsymbol{C_e}(\boldsymbol{x}, \boldsymbol{u}, t) = 0$, and inequality constraints $\boldsymbol{C_i}(\boldsymbol{x}, \boldsymbol{u}, t) \leq 0$. The control variable $\boldsymbol{u}(t)$ is continuous in time, therefore making this an infinite-dimensional optimization problem. If the states are written as a function of the controls, the optimal control problem more closely resembles the previous optimization problem. Therefore, the optimal control problem is convex if $J(u)$ and $C_i(u)$ are convex and $C_e(u)$ is linear [15].

1.3 Dynamic-Programming and Graph Theory

The dynamic-programming method was developed by Richard Bellman and others in the 1950's as a mathematical theory of multi-stage decision processes and is used in economic, industrial, engineering, and military domains. This approach was based on using functional equations and the principle of optimality during the onset of the ever-growing field of digital computing. It provided versatility and numerical solutions where the classical technique of the calculus of variations was lacking. The

calculus of variations and DP are complementary theories. The DP theory regards the extremal curve as an envelope of tangents and attempts to determine the optimal direction at each point on the extremal, while the calculus of variations considers the extremal curve to be a locus of points and attempts to determine this curve by means of a differential equation [16].

At the time of the inception of DP theory, there were two classes of methods to solve nonlinear differential equations. The first class depended on a discrete approximation to the exact solution, while the second depended on deriving an exact equation for a discrete approximation to the original continuous process. The latter class of methods calls for choosing the values of a function $y(x)$ at specific points over the desired interval as opposed to choosing the function $y(x)$ over a given interval. This process is used in the DP method for trajectory optimization [16].

The dynamic-programming method discussed herein is based on a simple trajectory optimization problem discussed by Bryson and Ho [17]. This problem, illustrated by the grid in Fig. 1.3, requires a path from A to G that yields the minimal cost. The grid is divided into vertical groupings of nodes referred to as subsets, here labeled by the letters A-G. The optimal path will stem from A through subsets B-F, moving always to the right, to G, the terminal point.

Each possible path segment has an associated cost, indicated by the number along each edge of the grid. The algorithm uses a recursive summation of the costs to determine the optimal path from the initial to the final position. The recursive summation technique begins at G and marches backward to the next subset of nodes, F, storing the optimal cost from each node in F to G. In this first case, these are trivially computed: traveling from the upper point in F to G has a cost of 10 while

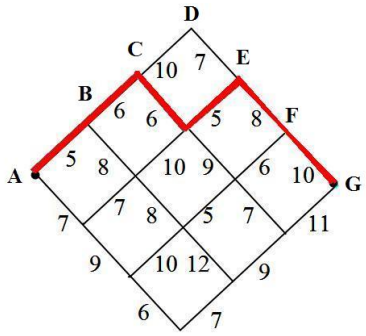


Figure 1.3: Results from Textbook Example Validation

the cost of traveling from the lower node in F to G yields a cost of 11. This process is repeated at subset E. The upper, middle, and lower nodes of E have costs of 18, 16, and 20 respectively. For each node, the optimal path through F is also stored. For example, for the middle node of E the optimal path passes through the upper node of F. This process of determining the optimal path to the terminal point continues backward through each subset until the initial point A is reached, and therefore the optimal path and cost are determined. In this algorithm, each path segment is only evaluated once.

The graph formed by the n vertices and m edges of Fig. 1.3 is known as a directed acyclic graph. A directed graph is obtained if a vertex p is connected to another vertex q by edge e and the order of the vertices is prescribed, meaning the vertex p is the tail of e and q is the head of e. Directed graphs containing no directed circuits, or non-isolated vertices, are called acyclic graphs. The directed acyclic graphs (DAG) resulting from DP trajectory optimization problems have solutions that can be computed in $\mathcal{O}(m)$ time. In the next chapter, the concept of DP is compared to the variational method in application to a single degree of freedom dynamic system.

Chapter 2

Development and Validation of Dynamic-Programming Routine

2.1 Method Development

Throughout the development of this routine, it was assumed that a fixed target and variable initial conditions for each munition were to be considered. It should be noted that in this method, a summation from a given node to the end point was used. However, a summation from the initial point to a given node could have been implemented, as both approaches yield the same path from A to B. Given the duality of this problem, the backward-marching method was chosen based on the problem to which it was to be applied, in which the target was fixed.

2.2 Validation Process

The dynamic programming algorithm was subjected to a dual validation process. First, it was shown to reconstruct the correct optimal path and final cost from an example grid with known costs. The solution is illustrated by the bold solid line in Fig. 1.3.

The next step in the validation process was to apply the algorithm to a dynamic system. A single degree of freedom system was chosen with a linear state equation,

quadratic cost function, and given initial and final states and times.

$$\dot{x} = ax + bu \tag{2.1}$$

$$J = \int_0^{t_F} \left(qx^2 + ru^2\right) \mathrm{d}t \tag{2.2}$$

For this problem, the optimal solution was computed analytically by the variational method. By substituting the cost function defined in Eq. (2.11) into Eq. (1.3), the Hamiltonian was determined.

$$H = \frac{1}{2}(qx^2 + ru^2) + \lambda(ax + bu) \tag{2.3}$$

Using the optimal control principles in Eq. (1.6), the necessary conditions for the single degree of freedom problem were determined.

$$\dot{\lambda} = -(qx + \lambda a) \tag{2.4}$$

$$0 = ru + \lambda b \tag{2.5}$$

$$\dot{x} = ax + bu \tag{2.6}$$

Equation (2.5) was solved for the control u and the solution was substituted into Eq. (2.6). This resulted in differential equations for the state, x, and costate, λ, in

terms of only x, λ, and the scalars a, b, q, and r.

$$\begin{bmatrix} \dot{x} \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} a & -\frac{b^2}{r} \\ -q & -a \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} \tag{2.7}$$

The solution of this linear differential equation was used for comparison with the DP solution. For implementation in the DP algorithm, constant control was assumed along each edge of the grid. Using the given initial and final conditions on the state and time, the state equation Eq. (2.1) was solved in terms of a constant k and the control u.

$$x_h = ke^{at} \quad ; \quad x_p = -\frac{bu}{a} \tag{2.8}$$

$$x = ke^{at} - \frac{bu}{a} \tag{2.9}$$

Also, the path grid was symmetric about a reference line from the initial to the final state. This allowed for the geometric determination of physical grid points and the time step between grid points. Consequently, the constant k and the control u necessary to travel between two grid points were determined by solving Eq. (2.10).

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} e^{at_1} & -\frac{b}{a} \\ e^{at_2} & -\frac{b}{a} \end{bmatrix} \begin{bmatrix} k \\ u \end{bmatrix} \tag{2.10}$$

The substitution of the constant control u and Eq. (2.9) into Eq. (2.11) provided an analytical expression for the costs.

$$J = \frac{1}{2}ru^2(t_2 - t_1) + \frac{1}{2}q\int_{t_1}^{t_2}\left(ke^{at} - \frac{bu}{a}\right)^2 \mathrm{d}t \tag{2.11}$$

The integral in Eq. (2.11) may be expanded to obtain an expression of known variables that was used to calculate the cost along each grid segment.

$$\begin{aligned} J = {} & \frac{1}{2}ru^2\left(t_2 - t_1\right) + \frac{1}{2}q\left(\frac{k^2}{2a}e^{2at_2} - 2k\frac{b}{a^2}ue^{at_2} + \frac{b^2}{a^2}u^2t_2\right) \\ & - \frac{1}{2}q\left(\frac{k^2}{2a}e^{2at_1} - 2k\frac{b}{a^2}ue^{at_1} + \frac{b^2}{a^2}u^2t_1\right) \end{aligned}$$

Following the determination of the cost along each segment of the path grid, the optimal path was determined by the DP method. Fig. 2.1 illustrates the solutions for when the scalar variables a, b, q, and r are unity and the initial and final states and times were $x = 1$ and 0 and $t = 0$ and 10, respectively. The DP trajectory follows the same trend as that of the variational method. The true optimal cost in the this case is $J = 2.410$. The solution found from the DP was $J = 2.518$. The performance cost increase of 4.43% is relatively minor. Additionally, the computational run-time was less than one second.

In Fig. 2.1, the circles illustrate the path defined by the dynamic programming algorithm, and the solid line is a third order curve fit used for visualization. It is clear that the DP method trajectory captures the trend of the variational method trajectory, which is illustrated by the dashed line. Following such tests and validation, the algorithm had been demonstrated in application to a dynamic system and in

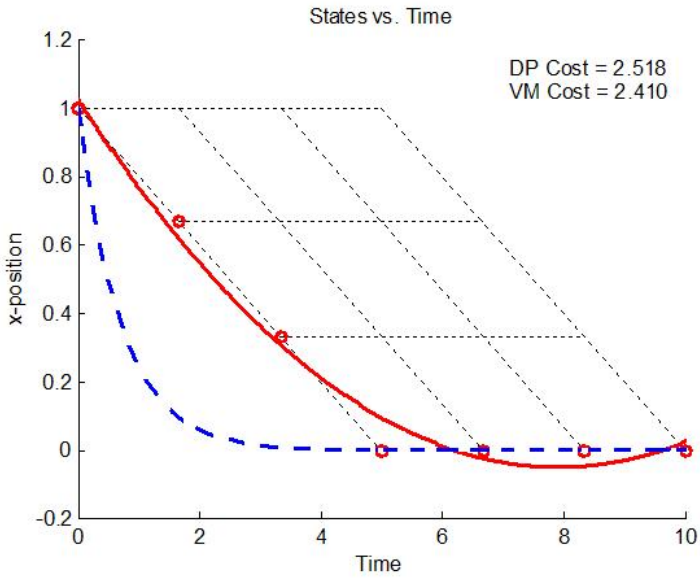


Figure 2.1: Results from Single Degree of Freedom Validation

correct generation of a path with cost on the same order as known true solutions. This motivated its application to the previously investigated complex dynamic system described in the following chapter.

Chapter 3

Problem Definition

A scenario will be considered with the two-dimensional plane populated by two munitions and a single fixed target. The state of each munition is given by its position in two dimensional space, $\boldsymbol{x}_1 = [x_1\ y_1]^\mathsf{T}$ and $\boldsymbol{x}_2 = [x_2\ y_2]^\mathsf{T}$. A constant-speed kinematic model is used to describe the motion of the munitions. The heading angles of the munitions are ψ_1 and ψ_2, and the speed of each munition is v. Here, the heading angles are treated as control variables.

$$\begin{aligned} \dot{x}_1 &= v\cos\psi_1 \qquad ; \qquad \dot{x}_2 = v\cos\psi_2 \\ \dot{y}_1 &= v\sin\psi_1 \qquad ; \qquad \dot{y}_2 = v\sin\psi_2 \end{aligned} \tag{3.1}$$

$$\dot{\boldsymbol{x}}_i = \boldsymbol{f}_i\left(\psi_i\right), \quad i \in \{1, 2\} \tag{3.2}$$

Additionally, each munition is considered to carry a sensor that is capable of measuring the target location in the xy plane. To design trajectories that improve the estimation of the target location, a model is needed of the sensor measurements and their uncertainties. The target has a position described by $\boldsymbol{x}_T = [x_T\ y_T]^\mathsf{T}$. The measurement of this target location by each munition, $\tilde{\boldsymbol{z}}_1 = [\tilde{x}_{T,1}\ \tilde{y}_{T,1}]^\mathsf{T}$ and $\tilde{\boldsymbol{z}}_2 =$

$[\tilde{x}_{T,2}\ \tilde{y}_{T,2}]^{\mathsf{T}}$, is modeled as shown below.

$$\begin{aligned} \tilde{x}_{T,1} &= x_T + w_{x,1}(0, \sigma_{x,1}) \quad ; \quad \tilde{x}_{T,2} = x_T + w_{x,2}(0, \sigma_{x,2}) \\ \tilde{y}_{T,1} &= y_T + w_{y,1}(0, \sigma_{y,1}) \quad ; \quad \tilde{y}_{T,2} = y_T + w_{y,2}(0, \sigma_{y,2}) \end{aligned} \tag{3.3}$$

The measurement errors from each munition are assumed to be independent of the errors from the other munition. The x and y measurement errors from each individual munition, however, are treated as correlated Gaussian random variables with zero mean and standard deviations of $\sigma_{x,i}$ and $\sigma_{y,i}$, where $i \in \{1, 2\}$. It is these uncertainties that will drive the trajectory design, and they can be selected to model a particular sensor design.

The error in the target-location measurements from an individual munition is treated as following a zero-mean jointly-Gaussian distribution that is uncorrelated in the down-range and cross-range directions, relative to the true target and munition locations. The errors in these directions, $w_{d,i}(0, \sigma_{d,i})$ and $w_{c,i}(0, \sigma_{c,i})$, can therefore be treated as independent Gaussian random variables. The standard deviations in the down-range and cross-range directions are modeled as functions of the range from the munition to the target.

$$\sigma_{d,i} = 0.1 r_i \quad ; \quad \sigma_{c,i} = 0.01 r_i \tag{3.4}$$

This models a sensor that is more accurate when close to the target and more accurate in the transverse direction than in the radial direction. The uncertainty in the measurement of the target location by the ith munition is illustrated in Fig. 3.1.

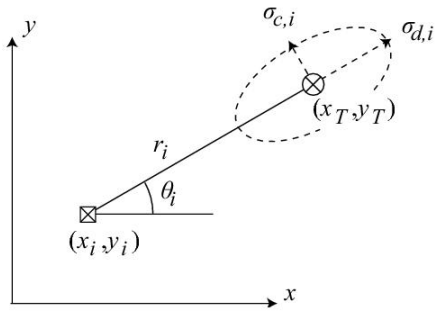


Figure 3.1: Measurement of the target by the ith munition and the associated error probability ellipse.

From the down-range and cross-range variables, the errors and the covariance matrix in the x and y coordinates can be found.

$$\begin{bmatrix} w_{x,i} \\ w_{y,i} \end{bmatrix} = \begin{bmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{bmatrix} \begin{bmatrix} w_{d,i} \\ w_{c,i} \end{bmatrix} \tag{3.5}$$

$$\boldsymbol{P}_i = \begin{bmatrix} \sigma_{x,i}^2 & \sigma_{xy,i} \\ \sigma_{xy,i} & \sigma_{y,i}^2 \end{bmatrix} = \begin{bmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{bmatrix} \begin{bmatrix} \sigma_{d,i}^2 & 0 \\ 0 & \sigma_{c,i}^2 \end{bmatrix} \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix} \tag{3.6}$$

Here, θ_i is the bearing angle of the target relative to the ith munition. The range and bearing angle for each target-munition pair are computed as shown below.

$$r_i = \sqrt{(x_T - x_i)^2 + (y_T - y_i)^2} \tag{3.7}$$

$$\theta_i = \tan^{-1}\left(\frac{y_T - y_i}{x_T - x_i}\right) \tag{3.8}$$

The measurements provided by both munitions can be fused into a single instantaneous estimate of the target location. This is done using a minimum-variance least-squares estimator (MVLSE) [18, 19]. The measurements of the target location from each munition are grouped into a measurement vector $\tilde{\boldsymbol{z}} = [\tilde{x}_{T,1}\ \tilde{y}_{T,1}\ \tilde{x}_{T,2}\ \tilde{y}_{T,2}]^\intercal$. This produces a linear measurement model in terms of the target location.

$$\boldsymbol{z} = \boldsymbol{H}\boldsymbol{x}_T + \boldsymbol{w} \tag{3.9}$$

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}^\intercal ; \qquad \boldsymbol{w} = \begin{bmatrix} w_{x,1} & w_{y,1} & w_{x,2} & w_{y,2} \end{bmatrix}^\intercal \tag{3.10}$$

Here, $\boldsymbol{w}$ is the vector of measurement errors. The covariance of this error vector is given by arranging the covariances from each munition.

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{P}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{P}_2 \end{bmatrix} \tag{3.11}$$

The instantaneous MVLSE of the target location and the associated covariance are given by the following.

$$\hat{\boldsymbol{x}}_T = \left(\boldsymbol{H}^\intercal \boldsymbol{R}^{-1} \boldsymbol{H}\right)^{-1} \boldsymbol{H}^\intercal \boldsymbol{R}^{-1} \tilde{\boldsymbol{z}} \tag{3.12}$$

$$\boldsymbol{P} = \left(\boldsymbol{H}^{\mathsf{T}}\boldsymbol{R}^{-1}\boldsymbol{H}\right)^{-1} \tag{3.13}$$

Considering the first of Eqs. (3.10), the MVLSE reduces to the following.

$$\hat{\boldsymbol{x}}_T = \begin{bmatrix} \hat{x}_T \\ \hat{y}_T \end{bmatrix} = \left(\boldsymbol{P}_1^{-1} + \boldsymbol{P}_2^{-1}\right)^{-1}\left(\boldsymbol{P}_1^{-1}\tilde{\boldsymbol{z}}_1 + \boldsymbol{P}_2^{-1}\tilde{\boldsymbol{z}}_2\right) \tag{3.14}$$

More importantly for the current purposes, the covariance of this combined estimate is related to the individual covariances of the measurements from each munition.

$$\boldsymbol{P} = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix} = \left(\boldsymbol{P}_1^{-1} + \boldsymbol{P}_2^{-1}\right)^{-1} \tag{3.15}$$

The covariance $\boldsymbol{P}$ now models the uncertainty in the combined target-location estimate based on the positioning of the two munitions relative to the target. The task of designing trajectories for the munitions in order to enhance the estimation performance can now be posed as the following optimal control problem. Consider the state vector $\boldsymbol{x} = [x_1\ y_1\ x_2\ y_2]^{\mathsf{T}}$. The heading angles of the munitions can be organized into a control vector $\boldsymbol{u} = [\psi_1\ \psi_2]^{\mathsf{T}}$. The state vector evolves according to the state equation found by grouping Eq. (3.2), $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{u}) = [\boldsymbol{f}_1^{\mathsf{T}}\ \boldsymbol{f}_2^{\mathsf{T}}]^{\mathsf{T}}$. For boundary conditions, the initial positions of the munitions will be considered a given, and the final position of munition 1 is required to be the target location, $x_1(t_F) = x_T$ and $y_1(t_F) = y_T$. The final position of munition 2 is free.

The goal will be to find the trajectories that minimize the following cost function, which is based on the MVLSE covariance.

$$J = \int_0^{t_F} \left(\sigma_x^2 + \sigma_y^2\right) \mathrm{d}t \tag{3.16}$$

The variances of each target location are functions of the states describing the munition configuration. Clearly, this cost function emphasizes the uncertainty over the entire trajectory.

Given the cost function in Eq. (3.16) subject to the state equation in Eq. (3.2) and the equality constraints $x_1(t_F) = x_T$ and $y_1(t_F) = y_T$, the convexity of the SLAP problem may be determined. For the SLAP problem to be convex, the cost function must be convex and the equality boundary conditions must be linear.

The convexity of the cost function will be numerically investigated for five trial solutions, although these solutions will not satisfy the boundary conditions. For the trial solutions, the given conditions are$(x_T, y_T) = (0, 0)$, $(x_1, y_1) = (0, -2000)$, $(x_2, y_2) = (2000, 0)$, $v = 300$, and ψ_1 and ψ_2 are constant during a trial.

Table 3.1 lists the resultant costs associated with each trial solution. Cases 1-3 exhibit convex behavior because the cost of Case 2 is below the linear interpolation of the costs of Cases 1 and 3. However, when the final time is varied while the heading angles remain the same between cases as in Cases 1, 4, and 5, the function evaluation of Case 4 is greater than the linear interpolation of the costs of Cases 1 and 5. Therefore, the cost function J is found to be nonconvex. Figures 3.2 and 3.3 illustrate this behavior.

Table 3.1: Cost of Trial Solutions.

Case	ψ_1	ψ_2	t_F	Cost
1	$\pi/2$	π	5	1.7326733×10^3
2	$\pi/2 - 0.05$	$\pi - 0.05$	5	1.7363854×10^3
3	$\pi/2 - 0.1$	$\pi - 0.1$	5	1.7475124×10^3
4	$\pi/2$	π	5.1	1.7373327×10^3
5	$\pi/2$	π	5.2	1.7414337×10^3

Also, the equality constraints, which are functions of sine and cosine in Eq. (3.17), are nonlinear by inspection. These characteristics demonstrate the nonconvexity of the SLAP problem.

$$
\begin{aligned}
v \int_0^{t_F} \cos\psi_1 \mathrm{dt} &= \mathrm{x_T} - \mathrm{x_1}(0) \\
v \int_0^{t_F} \sin\psi_1 \mathrm{dt} &= \mathrm{y_T} - \mathrm{y_1}(0)
\end{aligned}
\tag{3.17}
$$

The next chapter describes the process of applying the DP method to the SLAP problem.

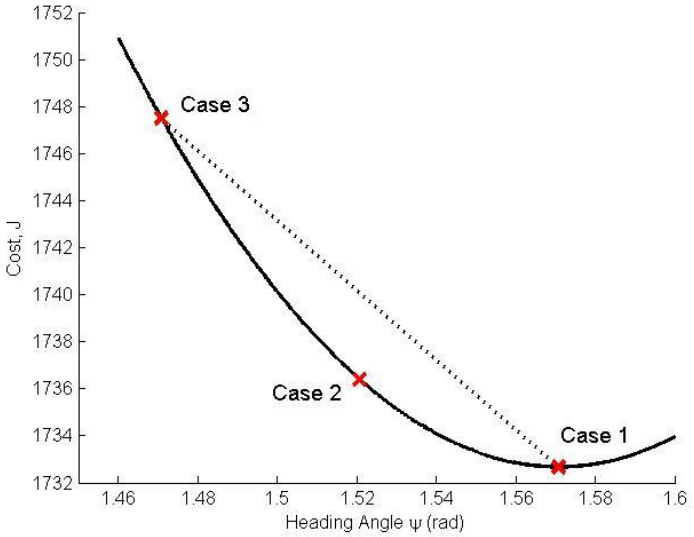


Figure 3.2: Investigation of cost function convexity with constant t_F

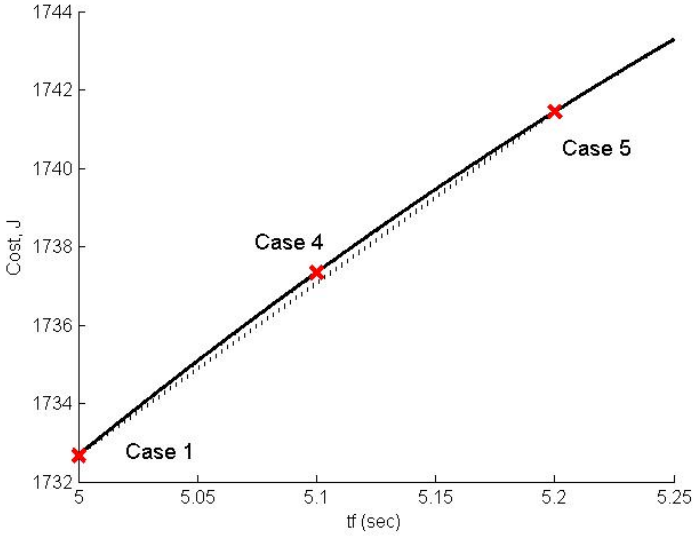


Figure 3.3: Investigation of cost function convexity with constant heading, ψ_1 and ψ_2

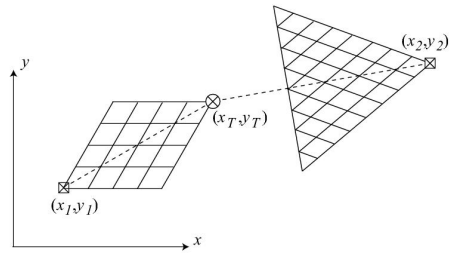
Chapter 4

Application of Dynamic-Programming Approach

The SLAP problem has previously been solved by applying the variational method discussed in Section 1.2, and the purpose of this work is to compare the results of the DP method to those of the variational method [1]. The optimal control problem was converted to a two point boundary value problem and solved by determining the initial costates with an iterative numerical solver in MATLAB. Although these solutions yielded the optimal solution and demonstrated that significant improvements in the target-location estimate could be achieved, the method was too computationally expensive for real-time implementation. This section describes solution of the problem by dynamic programming, with the goal of reducing computational expense.

Whereas variational methods consider a continuous range of heading angles at any instant in time, the approach considered here only allows a discrete number of possible heading at discrete instants in time. The trajectories were limited to two possible heading angles at each decision instant. Between decision points, the trajectories follow constant headings. This discretization generates a grid of possible trajectories, as illustrated in Fig. 4.1. This grid of physical points through which the munitions may travel is referred to as the path grid.

The path grids were laid out for each munition and were structured such that they were symmetric about a reference line from the initial state to the target location. The expansion of this grid is variable about the reference line by an angle, α. Because the results of the variational method showed that the munitions tended to approach the target at orthogonal headings, the path grid was of variable width to allow for outward



(a) (b)

Figure 4.1: Example of (a) path grids and (b) DAG where $n = 64$ and $m = 168$

sweeps. The degree of expansion was determined based on the initial positions of the munitions relative to the target. Munition 1 is constrained to hit the target, but the grid for Munition 2 allows multiple possible terminal points.

The nodes were organized into subsets of nodes that could be reached in a given amount of time. The time increment between layers was also assumed constant over all layers. This time increment is calculated from the initial range of the munition that will strike the target, which is assumed to always be the closer of the two munitions. The same time step is used for both munitions in order to preserve synchronized motion of the munitions.

This DP routine implements the Bellman-Ford model for trajectory optimization through dynamic-programming in Fortran 77 on a 2GHz PC. The combinatorial possibilities of physical node locations for the two munitions were used to form the vertices of a DAG with n vertices and m edges. Each vertex represents a particular location for each munition at a particular instant in time, and each edge corresponds to the cost value associated with the munitions traveling between those particular locations. These costs are calculated according to Eq. (3.16). The graph is directed and acyclic because the paths must follow the flow of time.

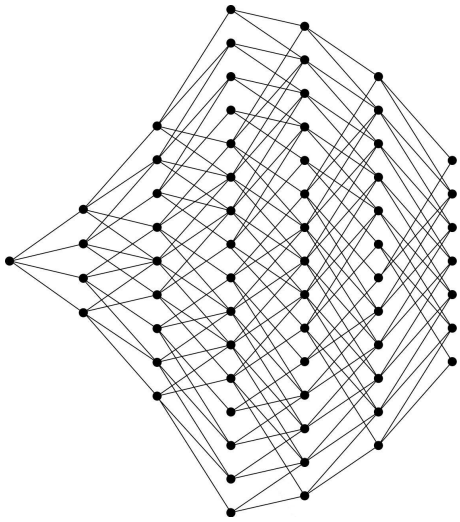
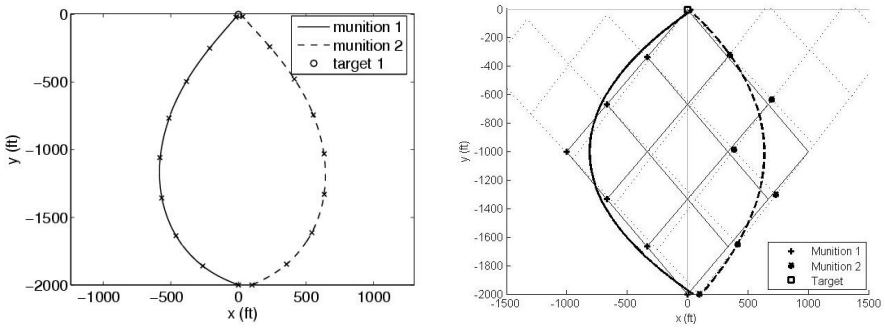


Figure 4.2: DAG where $n = 64$ and $m = 168$

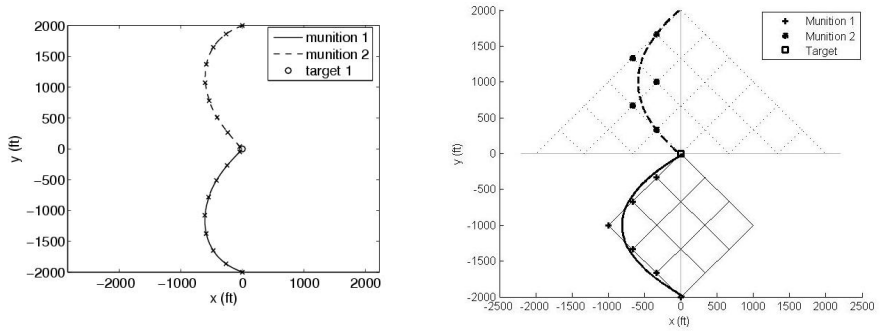
The vertices are arranged into subsets, where each subset represents a particular instant in time. Once the cost along each edge of the DAG is computed, the algorithm marches backward in time from the last layer of vertices to the first vertex to determine the lowest possible cost and the path that produces it. At each subset, the optimal path is computed by comparing the costs to proceed forward. That path and cost is then stored and the algorithm works backwards to the preceding subset, continuing this process until it reaches the initial vertex. The DAG associated with the path grids in Fig. 4.1 is shown in Fig. 4.2. Because the DAG in Fig. 4.2 has seven terminal vertices due to the free final state of Munition 2, the DP routine was run multiple times, in each run one of the vertices was assumed to be the true terminal vertex. The lowest resulting cost of the seven runs was then chosen as the optimal.

Example trajectories produced by the DP method are shown in Figs. 4.3 and 4.4 using $n = 64$ and $m = 168$. The DP trajectories are also compared to trajectories



(a) (b)

Figure 4.3: Problem 1 SLAP trajectories from (a) variational and (b) DP methods.



(a) (b)

Figure 4.4: Problem 2 SLAP trajectories from (a) variational and (b) DP methods.

computed from the variational method. A third-order curve fit was used to obtain the smooth trajectories from the grid points of the path grids. The trends of the DP trajectories capture those of the variational method. This is also evidenced in minor increase in the final cost of approximately 7.75% in each problem, as shown in Table 4.1. When paired with a computational run-time of 0.1 sec, compared to many minutes for the variational method, these cost values confirm the effectiveness of this method in solving the SLAP problem.

In implementing the DP method, the resolution of the path grids and the resulting DAG must be selected. In order to determine an appropriate resolution, the sensitivity of the cost to grid resolution was investigated. A lower resolution DAG with $n = 27$ and $m = 70$ was created. The resulting increase in performance cost for the considered problem was less than 1%, as shown in Table 4.1. Because the process of refining the grid resolution would be nontrivial, this low sensitivity to grid resolution did not motivate the investigation of resolutions greater than $n = 64$.

Table 4.1: Cost for sample SLAP trajectories.

Problem	DP Method Cost $n = 64$	DP Method Cost $n = 27$	Variational Method Cost
1	1.7181×10^4	1.7209×10^4	1.59×10^4
2	2.0317×10^4	2.0318×10^4	1.89×10^4

Chapter 5

Estimation Performance

The impact of the trajectories on the target-location estimation can now be evaluated. Although the trajectories were designed using a cost function based on the variances from a continuous MVLSE algorithm, the estimation performance will be evaluated using a recursive minimum-variance least squares estimation (RMVLSE) algorithm with discrete measurement updates. The estimates computed using the DP trajectories are compared to estimates using the variational-method trajectories and following trajectories from the initial conditions straight to the target location (STT trajectory). In each case, noisy measurements were simulated using the measurement model in Eq. (3.4). The measurements were generated by use of the RANDN command in MATLAB to generate a normal distribution of random numbers.

The munition sensors were assumed to collect measurements of the target location at a rate of 10 Hz. The RMVLSE algorithm operated as follows to determine the estimate and the uncertainty at the kth time step [18, 19]. The current estimate is computed as follows.

$$\boldsymbol{K}_k = \boldsymbol{P}_{k-1}\boldsymbol{H}^{\mathsf{T}}\left(\boldsymbol{H}\boldsymbol{P}_{k-1}\boldsymbol{H}^{\mathsf{T}} + \boldsymbol{R}\right)^{-1} \tag{5.1}$$

$$\hat{\boldsymbol{x}}_k^{(T)} = \hat{\boldsymbol{x}}_{k-1}^{(T)} + \boldsymbol{K}_k\left(\tilde{\boldsymbol{z}}_k - \boldsymbol{H}\hat{\boldsymbol{x}}_{k-1}^{(T)}\right) \tag{5.2}$$

The current covariance matrix is computed as shown.

$$\boldsymbol{P}_k = \begin{bmatrix} \sigma_{x,k}^2 & \sigma_{xy,k} \\ \sigma_{xy,k} & \sigma_{y,k}^2 \end{bmatrix} = \left(\boldsymbol{P}_{k-1}^{-1} + \boldsymbol{H}_k^{\mathsf{T}} \boldsymbol{R}_k^{-1} \boldsymbol{H}_k\right)^{-1} \tag{5.3}$$

To compare the estimation performance along the different trajectories, the size of the one-sigma uncertainty ellipsoid in the target-location estimate can be used as a metric. At the kth time step, this is given by the product of π with the square root of the product of the eigenvalues of $\boldsymbol{P}_k$. In particular, the ellipsoid size at $t_F - 2\,\text{sec}$ will be highlighted. Although t_F is different for each trajectory, at this point in time munition 1 is roughly 600 ft from the target.

Estimation performance will be compared for STT trajectories and SLAP trajectories computed by the variational and dynamic-programming methods at two sets of initial conditions. All DP results were computed with a mesh expansion angle α of $\pi/4$.

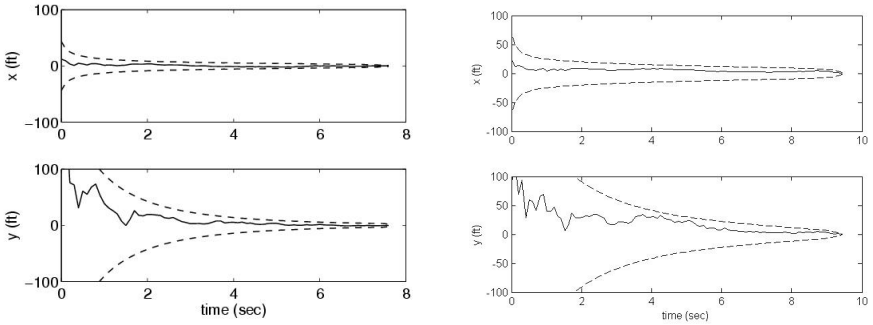
5.1 Estimation Results

5.1.1 Problem 1

Using the initial condition of $x_1(0) = 0\,\text{ft}$, $y_1(0) = -2000\,\text{ft}$, $x_2(0) = 100\,\text{ft}$, and $y_2(0) = -2000\,\text{ft}$, two munitions on STT trajectories generate a one-sigma uncertainty ellipse at with an area of $39.7\,\text{ft}^2$ at $t_F - 2\,\text{sec}$, with $t_F = 6.67\,\text{sec}$. When the two munitions follow the SLAP trajectories shown in Fig. 4.3, the final times increase to 8.09 sec and 9.43 sec for the variational method and DP method, respectively. However, the area of the error ellipse is reduced as shown in Table 5.1. The

Table 5.1: Area of one-sigma uncertainty ellipse.

Problem	STT	Variational Method	DP Method
1	$39.7\,\text{ft}^2$	$9.1\,\text{ft}^2$	$24.5\,\text{ft}^2$
2	$40.8\,\text{ft}^2$	$9.3\,\text{ft}^2$	$23.5\,\text{ft}^2$



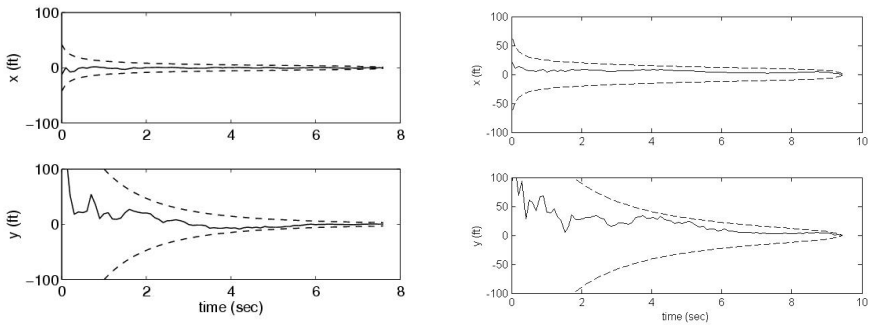
(a) (b)

Figure 5.1: Estimation errors using (a) Variational Method and (b) DP trajectories with $x_2(0) = 100\,\text{ft}$, and $y_2(0) = -2000\,\text{ft}$.

error histories for a sample simulation with noisy measurements and three-sigma error bounds ($\pm 3\sigma_{x,k}$ and $\pm 3\sigma_{y,k}$) generated by the RMVLSE algorithm are shown in Fig. 5.1. Figure 5.1(a) shows the errors in the x and y estimates of the target location using the variational method trajectories. Figure 5.1(b) show the errors using the DP trajectories.

5.1.2 Problem 2

Moving munition 2 to the initial condition $x_2(0) = 0\,\text{ft}$, and $y_2(0) = 2000\,\text{ft}$ results in an uncertainty ellipse with an area of $40.8\,\text{ft}^2$ for the STT trajectories, $9.3\,\text{ft}^2$ for the variational method trajectories, and $23.5\,\text{ft}^2$ for the DP trajectories.



(a) (b)

Figure 5.2: Estimation errors using (a) Variational Method and (b) DP trajectories with $x_2(0) = 0\,\text{ft}$, and $y_2(0) = 2000\,\text{ft}$.

The final time for the STT trajectories and the DP trajectories remained the same while that of the variational method increased to 8.21 sec. For these initial conditions, the error histories for a sample simulation with noisy measurements and three-sigma error bounds generated by the RMVLSE algorithm are shown in Fig. 5.2.

5.2 Discussion of Results

It is significant to note that in problems 1 and 2, the area of the estimation uncertainty ellipse is decreased by the SLAP trajectories as compared to that of the STT trajectories. However, the variational method trajectories improve the target-location estimate more than the DP trajectories, as the DP trajectories have a slower convergence rate. Although the performance cost of the DP trajectories was within 8% of that of the variational method, the area of the DP method's uncertainty ellipse was more than twice the size of that of the variational method at $t_F - 2\,\text{sec}$. This shows the sensitivity of the size of the uncertainty ellipse to trajectory design and the effect of the course grid structure used in the DP method.

Chapter 6

Conclusions

Careful trajectory design can have a significant impact on target-location estimation. In this work, the DP approach was used to demonstrate that SLAP trajectories are practical for real-time implementation. The advantage of this approach is that discretization in both time and spatial coordinates results in a DAG that can be solved in a deterministic amount of computation. This allows grid resolution to be selected based on the available computational resources and desired performance.

The trajectories output by the DP method captured the trends of the variational method and thus produced very similar performance costs. Both trajectories give similar good performance in estimating the target location compared to the STT trajectories, however the DP trajectory has slightly slower convergence than the variational method trajectories. This is due to the constant control between grid points and course grid structure requiring the trajectory to remain symmetric.

The work described here also demonstrated several limitations to the DP method of solving the SLAP problem. The need of keeping the motion of the muntions synchronized forced the selection of a fixed final time. Also, even though the DP trajectories were very similar in cost to the variational-method trajectories, the estimation performance was very sensitive to the slight increase in cost.

More accurate target-location estimation could allow more accurate strike capability of targets that are difficult to detect. Further work is needed to demonstrate the impact of these estimation enhancements on guidance and control performance. In future implementations, heuristic methods may be developed based on insight gained

from solutions of the optimal control problem. The DP approach will still be a useful development tool to cheaply investigate various solutions.

Bibliography

[1] Sinclair, A.J., Prazenica, R.J., and Jeffcoat, D.E., "Simultaneous Localization and Planning for Cooperative Air Munitions", In Murphey, R., Pardalos, P.M., eds.: 7th International Conference on Cooperative Control and Optimization, Springer, New York (2007)

[2] P. R. Chandler, M. Pachter, K. E. Nygard, and D. Swaroop, "Cooperative Control for Target Classification", In Murphey, R., Pardalos, P.M., eds.: Cooperative Control and Optimization, Kluwer, Netherlands (2002) 1-19

[3] Jeffcoat, D.E., "Coupled detection rates: An introduction", In Grundel, D., Murphey, R., Pardalos, P.M., eds.: Theory and Algorithms for Cooperative Systems, World Scientific, New Jersey (2004) 157-167

[4] Frew, E. and Lawrence, D., "Cooperative Stand-off Tracking of Moving Targets by a Team of Autonomous Aircraft", In: AIAA Guidance, Navigation, and Control Conference, San Fancisco, California (August 2005) AIAA-2005-6363

[5] Fawcett, J.A., "Effect of Course Maneuvers on Bearings-only Range Estimation", IEEE Transactions on Acoustics, Speech, and Signal Processing 36(8) (1988) 1193-1199

[6] Hammel, S.E., Liu, P.T., Hilliard, E.J., and Gong, K.F., "Optimal Observor Motion for Localization with Bearing Measurements", Computers and Mathematics with Applications 18(1-3) (1989) 171-180

[7] Oshman, Y. and Davidson, P., "Optimization of Observer Trajectories for Bearings-only Target Localization" IEEE Transactions on Aerospace and Electronic Systems 35(3) (1999) 892-902

[8] Logothetis, A., Isaksson, A., and Evans, R.J., "Comparison of Suboptimal Strategies for Optimal Own-ship Maneuvers in Bearings-only Tracking", In: American Control Conference, Phiadelphia, Pennsylvania (June 1998)

[9] Passerieux, J.M. and VanCappel, D., "Optimal Observer Maneuver for Bearings-only Tracking", IEEE Transactions on Aerospace and Electronic Systems 34(3) (1998) 777-788

[10] Frew, E.W., and Rock, S.M., "Trajectory Generation for Constant Velocity Target Motion Estimation Using Monocular Vision", In: IEEE International Conference on Robotics & Automation, Taipei, Taiwan (September 2003)

[11] Watanabe, Y., Johnson, E.N., and Calise, A.J., "Vision-based Guidance Design from Sensor Trajectory Optimization", In: AIAA Guidance, Navigation, and Control Conference, Keystone, Colorado (August 2006) AIAA-2006-6607

[12] Ousingsawat, J. and Campbell, M.E., "Optimal Cooperative Reconnaissance Using Multiple Vehicles" Journal of Guidance, Control, and Dynamics 30(1) (2007) 122-132

[13] Grocholsky, B., "Information-Theoretic Control of Multiple Sensor Platforms", PhD thesis, University of Sydney, Sydney, Australia (2002)

[14] Brogan, W.L., *Modern Control Theory*, Prentice Hall, NJ (1991)

[15] Boyd, S. and Vandenberghe, L., *Convex Optimization*, Cambridge University Press, Cambridge, UK (2004)

[16] Bellman, R.E. and Dreyfus, S.E., *Applied Dynamic Programming*, Princeton University Press, Princeton, NJ (1962)

[17] Bryson, A.E. and Ho, Y-C., it Applied Optimal Control: Optimization, Estimation, and Control, Hemisphere Publishing Corporation, Washington, DC (1975)

[18] Stengel, R.F., *Optimal Control and Estimation*, Dover, New York (1986)

[19] Crassidis, J.L. and Junkins, J.L., *Optimal Estimation of Dynamic Systems*, Chapman & Hall/CRC, Boca Raton, Florida (2004)

Appendices

Appendix A

FORTRAN Code Used for Dynamic-Programming

```
program CoopControl

implicit double precision (a-h,o-z)
integer NEQ,B1,B2,C1,C2,Dp1,Dp2,E1,E2,F1,F2,G1,G2,nmax,l,count
parameter (NEQ = 5, NV = 2, l = 168, nmax = 4)

dimension Xsend(NEQ),D1(NEQ),D2(NEQ),D3(NEQ),psi(NV,2)
dimension costF2G(2,6,1,7),costE2F(3,5,2,6),costD2E(4,4,3,5)
dimension costB2C(2,2,3,3),costA2B(1,1,2,2),costC2D(3,3,4,4)
dimension costE(3,5,5),costD(4,4,7),costC(3,3,9),costA(1,1,13)
dimension costF(2,6,3),costB(2,2,11),theta(NV)
dimension param(10),grid(3*nmax**2-nmax,2),x(NV),y(NV)
dimension tcostA(1,1,13)

open(unit=30,file='CoopControl.txt')
open(unit=20,file='timeron.txt')
close(20)

!Initial Conditions
x1 = 0.d0
y1 = -2000.d0
x2 = 0.d0
y2 = 2000.d0
xT = 0.d0
yT = 0.d0
cost0 = 0.d0
t = 0.d0
v = 300.d0
pi = acos(-1.d0)

!Define Vehicle 1 as the closer Vehicle
r1 = sqrt(x1**2.d0 + y1**2.d0)
r2 = sqrt(x2**2.d0 + y2**2.d0)
```

```
if (r1.gt.r2) then
x(1) = x2
y(1) = y2
x(2) = x1
y(2) = y1
else
x(1) = x1
y(1) = y1
x(2) = x2
y(2) = y2
end if

!Vehicle Intial Position
grid(1,1) = x(1)
grid(1,2) = y(1)
grid(nmax**2+1,1) = x(2)
grid(nmax**2+1,2) = y(2)

!Determine Bearing Angle, Theta, and Mesh Expansion Angle, a
!================================================================!
do i = 1,2 !number of vehicles
theta(i) = atan((yT-y(i))/(xT-x(i)))

if (xT.lt.x(i)) then
   if (yT.gt.y(i)) then
   theta(i) = pi + theta(i)
   elseif (yT.lt.y(i)) then
   theta(i) = -pi + theta(i)
   else
   theta(i) = pi
   end if
elseif (yT.eq.y(i)) then
   if (xT.le.x(i)) then
   theta(i) = pi
   end if
end if
end do

!determine mesh expansion angle, a
```

```
!=================================================================!
diff = abs(theta(1))-abs(theta(2))
!write(*,*) diff*180.d0/pi
if (abs(diff).eq.pi/2.d0) then
a = pi/7.d0
else
a = pi/3.d0
end if
!write(*,*) 'a', a*180.d0/pi
!=================================================================!

param(1) = xT
param(2) = yT
param(3) = v
param(4) = a

!Time step is based on vehicle that strikes target, Veh1
dt = sqrt(x(1)**2.d0 + y(1)**2.d0)/6.d0/(v*cos(a))
dtcost = dt/20.d0

!Determine Heading Angle, Psi
!=================================================================!
do i = 1,NV

if (xT.ge.x(i)) then
   if (yT.gt.y(i)) then
   psi(i,1) = theta(i) + a
   psi(i,2) = theta(i) - a
   elseif (yT.lt.y(i)) then
   psi(i,1) = theta(i) + a
   psi(i,2) = theta(i) - a
   else
   psi(i,1) = theta(i) + a
   psi(i,2) = theta(i) - a
   end if

elseif (xT.lt.x(i)) then
   if (yT.gt.y(i)) then
   psi(i,2) = theta(i) - a
```

```
   psi(i,1) = theta(i) + a
   elseif (yT.lt.y(i)) then
   psi(i,2) = theta(i) - a
   psi(i,1) = theta(i) + a
   else
   psi(i,1) = -pi + a
   psi(i,2) = pi - a
   end if

end if

do j = 1,2
if (psi(i,j).lt.-pi) then
psi(i,j) = pi - (abs(psi(i,j))- pi)
end if
if (psi(i,j).gt.pi) then
psi(i,j) = -pi + (psi(i,j)-pi)
end if
end do

end do

!First Time Step,  A to B
!================================================================!
do i = 1,2                    !heading choice for veh1
do j = 1,2                    !heading choice for veh2
Xsend(1) = x(1)
Xsend(2) = y(1)
Xsend(3) = x(2)
Xsend(4) = y(2)
Xsend(5) = cost0

param(5) = psi(1,i)
param(6) = psi(2,j)

do jj = 1,20
call RK4(t,dtcost,NEQ,Xsend,param,D1,D2,D3)
end do
```

```
t = t - dt
if (j.eq.1) then
grid(i+1,1) = Xsend(1)
grid(i+1,2) = Xsend(2)
end if
if (i.eq.2) then
grid(j+17,1) = Xsend(3)
grid(j+17,2) = Xsend(4)
end if
CostA2B(1,1,i,j) = Xsend(5)
end do
end do

!Second Time Step,  B to C
!================================================================!
do i = 1,2
do j = 1,2
do m = 1,3
do n = 1,3
CostB2C(i,j,m,n) = -1.d0
end do
end do
end do
end do

t = t + dt
do i = 1,2              !initial point veh1
do ii = 1,2             !initial point veh2
do j = 1,2              !heading choice for veh1
do k = 1,2              !heading choice for veh2

   x1 = grid(i+1,1)
   y1 = grid(i+1,2)
   x2 = grid(ii+nmax**2+1,1)
   y2 = grid(ii+nmax**2+1,2)

   Xsend(1) = x1
   Xsend(2) = y1
   Xsend(3) = x2
```

```
   Xsend(4) = y2
   Xsend(5) = cost0

   param(5) = psi(1,j)
   param(6) = psi(2,k)

do jj = 1,20
call RK4(t,dtcost,NEQ,Xsend,param,D1,D2,D3)
end do
   t = t - dt
   if (j.eq.1.and.k.eq.1) then
   costB2C(i,ii,i,ii) = Xsend(5)
   end if
   if (j.eq.2.and.k.eq.1) then
   costB2C(i,ii,i+1,ii) = Xsend(5)
   end if
   if (k.eq.2.and.j.eq.1) then
   costB2C(i,ii,i,ii+1) = Xsend(5)
   end if
   if (j.eq.2.and.k.eq.2) then
   costB2C(i,ii,i+1,ii+1) = Xsend(5)
   end if

   if (i.eq.1) then
       if (k.eq.1) then
       grid(j+3,1) = Xsend(1)
       grid(j+3,2) = Xsend(2)
       end if
       if (ii.eq.1) then
       grid(k+19,1) = Xsend(3)
       grid(k+19,2) = Xsend(4)
       end if
   end if

   if (i.eq.2) then
   if (j.eq.2) then
   grid(j+4,1) = Xsend(1)
   grid(j+4,2) = Xsend(2)
   end if
```

```
   if (ii.eq.2) then
   grid(k+20,1) = Xsend(3)
   grid(k+20,2) = Xsend(4)
   end if
   end if
   end do
end do
end do
end do

!Third Time Step,  C to D
!================================================================!
do i = 1,3
do j = 1,3
do m = 1,4
do n = 1,4
CostC2D(i,j,m,n) = -1.d0
end do
end do
end do
end do
t = t + dt

do i = 1,3           !initial point veh1
   do ii = 1,3       !initial point veh2
   do j = 1,2        !heading choice of veh1
   do k = 1,2        !heading choice of veh2

   x1 = grid(i+3,1)        !i+#start points
   y1 = grid(i+3,2)
   x2 = grid(ii+19,1)       !i+nmax**2+#start points
   y2 = grid(ii+19,2)
   Xsend(1) = x1
   Xsend(2) = y1
   Xsend(3) = x2
   Xsend(4) = y2
   Xsend(5) = cost0

   param(5) = psi(1,j)
```

```
   param(6) = psi(2,k)
   if (i.eq.3.and.j.eq.1) then
   end if
do jj = 1,20
call RK4(t,dtcost,NEQ,Xsend,param,D1,D2,D3)
end do
   t = t - dt

   if (j.eq.1.and.k.eq.1) then
   costC2D(i,ii,i,ii) = Xsend(5)
   end if
   if (j.eq.2.and.k.eq.1) then
   costC2D(i,ii,i+1,ii) = Xsend(5)
   end if
   if (k.eq.2.and.j.eq.1) then
   costC2D(i,ii,i,ii+1) = Xsend(5)
   end if
   if (j.eq.2.and.k.eq.2) then
   costC2D(i,ii,i+1,ii+1) = Xsend(5)
   end if
   if (i.eq.1) then
   if (k.eq.1) then
   grid(j+6,1) = Xsend(1)
   grid(j+6,2) = Xsend(2)
   end if
   if (ii.eq.1) then
   grid(k+22,1) = Xsend(3)
   grid(k+22,2) = Xsend(4)
   end if
   end if

   if (i.eq.3) then
   if (k.eq.1) then
   grid(j+8,1) = Xsend(1)
   grid(j+8,2) = Xsend(2)
   end if
   if (ii.eq.3) then
   grid(k+24,1) = Xsend(3)
   grid(k+24,2) = Xsend(4)
```

```
   end if
   end if
   end do
   end do
end do
end do

!Fourth Time Step,  D to E
!================================================================!
do i = 1,4
do j = 1,4
do m = 1,3
do n = 1,5
CostD2E(i,j,m,n) = -1.d0
end do
end do
end do
end do

t = t + dt
do i = 1,4                !initial point veh1
do ii = 1,4               !initial point veh2
do j = 1,2                !heading choice veh1
do k = 1,2                !heading choice veh2

   if (i.eq.1.and.j.eq.1) then
   goto 75
   end if
   if (i.eq.4.and.j.eq.2) then
   goto 75
   end if

   x1 = grid(i+6,1)
   y1 = grid(i+6,2)
   x2 = grid(ii+22,1)
   y2 = grid(ii+22,2)

   Xsend(1) = x1
   Xsend(2) = y1
```

```
      Xsend(3) = x2
      Xsend(4) = y2
      Xsend(5) = cost0

      param(5) = psi(1,j)
      param(6) = psi(2,k)

    do jj = 1,20
    call RK4(t,dtcost,NEQ,Xsend,param,D1,D2,D3)
    end do

      t = t - dt

      if (i.eq.1.and.j.eq.2) then
      costD2E(i,ii,i,ii-1+k) = Xsend(5)
      goto 76
      end if
      if (i.eq.4.and.j.eq.1) then
      costD2E(i,ii,i-1,ii-1+k) = Xsend(5)
      goto 76
      end if
      if (j.eq.2.and.k.eq.1) then
      costD2E(i,ii,i,ii) = Xsend(5)
      goto 76
      end if
      if (k.eq.2.and.j.eq.1) then
      costD2E(i,ii,i-1,ii+1) = Xsend(5)
      goto 76
      end if
      if (j.eq.2.and.k.eq.2) then
      costD2E(i,ii,i,ii+1) = Xsend(5)
      goto 76
      end if
      if (j.eq.1.and.k.eq.1) then
      costD2E(i,ii,i-1,ii) = Xsend(5)
      goto 76
      end if

76    continue
```

```
         if (i.eq.1) then
         if (ii.eq.1) then
         grid(k+26,1) = Xsend(3)
         grid(k+26,2) = Xsend(4)
         end if
         if (k.eq.1) then
         grid(k+10,1) = Xsend(1)
         grid(k+10,2) = Xsend(2)
         end if
         end if

         if (i.eq.3) then
         if (k.eq.1) then
         grid(j+11,1) = Xsend(1)
         grid(j+11,2) = Xsend(2)
         end if
         if (ii.eq.3) then
         grid(k+28,1) = Xsend(3)
         grid(k+28,2) = Xsend(4)
         end if
         end if

         if (i.eq.4.and.ii.eq.4) then
         grid(i+27,1) = Xsend(3)
         grid(i+27,2) = Xsend(4)
         end if
         end do
75       continue
         end do
      end do
      end do

      !Fifth Time Step,  E to F
      !====================================================================!
      do i = 1,3
      do j = 1,5
      do m = 1,2
      do n = 1,6
      CostE2F(i,j,m,n) = -1.d0
```

```
end do
end do
end do
end do

t = t + dt
   do i = 1,3                          !starting point veh1
   do ii = 1,5                         !starting point veh2
   do j = 1,2                          !heading for veh1
   do k = 1,2                          !heading for veh2
   x1 = grid(i+10,1)
   y1 = grid(i+10,2)
   x2 = grid(ii+26,1)
   y2 = grid(ii+26,2)

   if(i.eq.1.and.j.eq.1.or.i.eq.3.and.j.eq.2) then
   goto 26
   end if
   Xsend(1) = x1
   Xsend(2) = y1
   Xsend(3) = x2
   Xsend(4) = y2
   Xsend(5) = cost0

   param(5) = psi(1,j)
   param(6) = psi(2,k)

do jj = 1,20
call RK4(t,dtcost,NEQ,Xsend,param,D1,D2,D3)
end do
   t = t - dt

   if (i.eq.1.and.j.eq.2) then
   costE2F(i,ii,i,ii-1+k) = Xsend(5)
   goto 77
   end if
   if (i.eq.3.and.j.eq.1) then
   costE2F(i,ii,i-1,ii-1+k) = Xsend(5)
   goto 77
```

```
         end if
         if (j.eq.2.and.k.eq.1) then
         costE2F(i,ii,i,ii) = Xsend(5)
         goto 77
         end if
         if (k.eq.2.and.j.eq.1) then
         costE2F(i,ii,i-1,ii+1) = Xsend(5)
         goto 77
         end if
         if (j.eq.2.and.k.eq.2) then
         costE2F(i,ii,i,ii+1) = Xsend(5)
         goto 77
         end if
         if (j.eq.1.and.k.eq.1) then
         costE2F(i,ii,i-1,ii) = Xsend(5)
         goto 77
         end if

77       continue
         if (i.eq.2) then
         grid(j+13,1) = Xsend(1)
         grid(j+13,2) = Xsend(2)
         end if
         if (ii.eq.1.or.ii.eq.3.or.ii.eq.5) then
         grid(ii+30+k,1) = Xsend(3)
         grid(ii+30+k,2) = Xsend(4)
         end if

         end do
26       continue
         end do
      end do
      end do

      !Sixth Time Step,  F to G
      !=================================================================!
      t = t + dt

      do i = 1,2
```

```
do j = 1,6
!do m = 1,3
do n = 1,7
CostF2G(i,j,1,n) = -1.d0
end do
end do
end do

do i = 1,2                    !initial point veh1
   do ii = 1,6                !initial point veh2
   do j = 1,2                 !heading choice veh1
   do k = 1,2                 !heading choice veh2
   x1 = grid(i+13,1)
   y1 = grid(i+13,2)
   x2 = grid(ii+31,1)
   y2 = grid(ii+31,2)

   if(i.eq.1.and.j.eq.1.or.i.eq.2.and.j.eq.2) then
   goto 27
   end if
   Xsend(1) = x1
   Xsend(2) = y1
   Xsend(3) = x2
   Xsend(4) = y2
   Xsend(5) = cost0

   param(5) = psi(1,j)
   param(6) = psi(2,k)

do jj = 1,20
call RK4(t,dtcost,NEQ,Xsend,param,D1,D2,D3)
end do
   t = t - dt

   if (i.eq.1.and.j.eq.2) then
   costF2G(i,ii,i,ii-1+k) = Xsend(5)
   goto 78
   end if
   if (i.eq.2.and.j.eq.1) then
```

```
         costF2G(i,ii,i-1,ii-1+k) = Xsend(5)
         goto 78
         end if
         if (j.eq.2.and.k.eq.1) then
         costF2G(i,ii,i,ii) = Xsend(5)
         goto 78
         end if
         if (k.eq.2.and.j.eq.1) then
         costF2G(i,ii,i-1,ii+1) = Xsend(5)
         goto 78
         end if
         if (j.eq.2.and.k.eq.2) then
         costF2G(i,ii,i,ii+1) = Xsend(5)
         goto 78
         end if
         if (j.eq.1.and.k.eq.1) then
         costF2G(i,ii,i-1,ii) = Xsend(5)
         goto 78
         end if

78       continue
         if (i.eq.2.and.j.eq.1) then
         grid(j+15,1) = Xsend(1)
         grid(j+15,2) = Xsend(2)
         end if
         if (ii.eq.1.or.ii.eq.3.or.ii.eq.5.or.ii.eq.6) then
         grid(ii+k+36,1) = Xsend(3)
         grid(ii+k+36,2) = Xsend(4)
         end if
         end do
27       continue
         end do
      end do
      end do

      do i = 1,44
         write(30,*) grid(i,1), grid(i,2)
      end do
```

```
!=================================================================!
!DYNAMIC PROGRAMMING
!=================================================================!
t = t + dt
!Cost G2F
costA(1,1,1) = -1.d0
do k = 1,2*nmax-1  !G point for veh2
!=================================================================!

do i = 1,2             !F points veh1
   do j = 1,6          !F points veh2
   costF(i,j,1) = -1.d0
   tempcost = costF2G(i,j,1,k)
   if(costF(i,j,1).eq.-1.d0.and.tempcost.ne.-1.d0)then
   costF(i,j,1) = tempcost
   costF(i,j,2) = 1
   costF(i,j,3) = k
   elseif(tempcost.ne.-1.d0.and.tempcost.lt.costF(i,j,1))then
   costF(i,j,1) = tempcost
   costF(i,j,2) = 1
   costF(i,j,3) = k
   end if
end do
end do

!Cost F2E
!=================================================================!
do i = 1,3
   do j = 1,5
   costE(i,j,1) = -1.d0
   do m = 1,2
   do n = 1,6
   if(costE2F(i,j,m,n).eq.-1.d0.or.costF(m,n,1).eq.-1.d0)then
   tempcost = -1.d0
   else
   tempcost = costE2F(i,j,m,n)+costF(m,n,1)
   end if
   if(tempcost.ne.-1.d0.and.costE(i,j,1).eq.-1.d0)then
   costE(i,j,1) = tempcost
```

```
   costE(i,j,2) = m
   costE(i,j,3) = n
   costE(i,j,4) = costF(m,n,2)
   costE(i,j,5) = costF(m,n,3)
   elseif(tempcost.ne.-1.d0.and.tempcost.lt.costE(i,j,1))then
   costE(i,j,1) = tempcost
   costE(i,j,2) = m
   costE(i,j,3) = n
   costE(i,j,4) = costF(m,n,2)
   costE(i,j,5) = costF(m,n,3)
   end if
   end do
   end do
   end do
end do

!Cost E2D
!================================================================!
do i = 1,4
   do j = 1,4
   costD(i,j,1) = -1.d0
      do m = 1,3
      do n = 1,5
      if(costD2E(i,j,m,n).eq.-1.d0.or.costE(m,n,1).eq.-1.d0)then
      tempcost = -1.d0
      else
      tempcost = costD2E(i,j,m,n) +  costE(m,n,1)
      end if
      if(tempcost.ne.-1.d0.and.costD(i,j,1).eq.-1.d0)then
      costD(i,j,1) = tempcost
      costD(i,j,2) = m
      costD(i,j,3) = n
      costD(i,j,4) = costE(m,n,2)
      costD(i,j,5) = costE(m,n,3)
      costD(i,j,6) = costE(m,n,4)
      costD(i,j,7) = costE(m,n,5)
      elseif(tempcost.ne.-1.d0.and.tempcost.lt.costD(i,j,1))then
      costD(i,j,1) = tempcost
      costD(i,j,2) = m
```

```
      costD(i,j,3) = n
      costD(i,j,4) = costE(m,n,2)
      costD(i,j,5) = costE(m,n,3)
      costD(i,j,6) = costE(m,n,4)
      costD(i,j,7) = costE(m,n,5)
      end if

      end do
      end do
      end do
end do

!Cost D2C
!================================================================!
do i = 1,3
   do j = 1,3
   costC(i,j,1) = -1.d0
   do m = 1,4
   do n = 1,4
   if(costC2D(i,j,m,n).eq.-1.d0.or.costD(m,n,1).eq.-1.d0)then
   tempcost = -1.d0
   else
   tempcost = costC2D(i,j,m,n) + costD(m,n,1)
   end if
   if(tempcost.ne.-1.d0.and.costC(i,j,1).eq.-1.d0)then
   costC(i,j,1) = tempcost
   costC(i,j,2) = m
   costC(i,j,3) = n
   costC(i,j,4) = costD(m,n,2)
   costC(i,j,5) = costD(m,n,3)
   costC(i,j,6) = costD(m,n,4)
   costC(i,j,7) = costD(m,n,5)
   costC(i,j,8) = costD(m,n,6)
   costC(i,j,9) = costD(m,n,7)
   elseif(tempcost.ne.-1.d0.and.tempcost.lt.costC(i,j,1))then
   costC(i,j,1) = tempcost
   costC(i,j,2) = m
   costC(i,j,3) = n
   costC(i,j,4) = costD(m,n,2)
```

```
   costC(i,j,5) = costD(m,n,3)
   costC(i,j,6) = costD(m,n,4)
   costC(i,j,7) = costD(m,n,5)
   costC(i,j,8) = costD(m,n,6)
   costC(i,j,9) = costD(m,n,7)
   end if
   end do
   end do
   end do
end do

!Cost C2B
!=====================================================================!
do i = 1,2
   do j = 1,2
   costB(i,j,1) = -1.d0
   do m = 1,3
   do n = 1,3
   if(costB2C(i,j,m,n).eq.-1.d0.or.costC(m,n,1).eq.-1.d0)then
   tempcost = -1.d0
   else
   tempcost = costB2C(i,j,m,n) + costC(m,n,1)
   end if
   if(tempcost.ne.-1.d0.and.costB(i,j,1).eq.-1.d0)then
   costB(i,j,1) = tempcost
   costB(i,j,2) = m
   costB(i,j,3) = n
   costB(i,j,4) = costC(m,n,2)
   costB(i,j,5) = costC(m,n,3)
   costB(i,j,6) = costC(m,n,4)
   costB(i,j,7) = costC(m,n,5)
   costB(i,j,8) = costC(m,n,6)
   costB(i,j,9) = costC(m,n,7)
   costB(i,j,10) = costC(m,n,8)
   costB(i,j,11) = costC(m,n,9)
   elseif(tempcost.ne.-1.d0.and.tempcost.lt.costB(i,j,1))then
   costB(i,j,1) = tempcost
   costB(i,j,2) = m
   costB(i,j,3) = n
```

```
   costB(i,j,4) = costC(m,n,2)
   costB(i,j,5) = costC(m,n,3)
   costB(i,j,6) = costC(m,n,4)
   costB(i,j,7) = costC(m,n,5)
   costB(i,j,8) = costC(m,n,6)
   costB(i,j,9) = costC(m,n,7)
   costB(i,j,10) = costC(m,n,8)
   costB(i,j,11) = costC(m,n,9)
   end if
   end do
   end do
   end do
end do

!CostB2A
!================================================================!
tcostA(1,1,1) = -1.d0
i = 1
j = 1
do m = 1,2
do n = 1,2
   if(costA2B(i,j,m,n).eq.-1.d0.or.costB(m,n,1).eq.-1.d0)then
   tempcost = -1.d0
   else
   tempcost = costA2B(i,j,m,n) + costB(m,n,1)
   end if
if(tempcost.ne.-1.d0.and.tcostA(1,1,1).eq.-1.d0)then
   tcostA(i,j,1) = tempcost
   tcostA(i,j,2) = m
   tcostA(i,j,3) = n
   tcostA(i,j,4) = costB(m,n,2)
   tcostA(i,j,5) = costB(m,n,3)
   tcostA(i,j,6) = costB(m,n,4)
   tcostA(i,j,7) = costB(m,n,5)
   tcostA(i,j,8) = costB(m,n,6)
   tcostA(i,j,9) = costB(m,n,7)
   tcostA(i,j,10) = costB(m,n,8)
   tcostA(i,j,11) = costB(m,n,9)
   tcostA(i,j,12) = costB(m,n,10)
```

```
   tcostA(i,j,13) = costB(m,n,11)
elseif(tempcost.ne.-1.d0.and.tempcost.lt.tcostA(1,1,1))then
   tcostA(i,j,1) = tempcost
   tcostA(i,j,2) = m
   tcostA(i,j,3) = n
   tcostA(i,j,4) = costB(m,n,2)
   tcostA(i,j,5) = costB(m,n,3)
   tcostA(i,j,6) = costB(m,n,4)
   tcostA(i,j,7) = costB(m,n,5)
   tcostA(i,j,8) = costB(m,n,6)
   tcostA(i,j,9) = costB(m,n,7)
   tcostA(i,j,10) = costB(m,n,8)
   tcostA(i,j,11) = costB(m,n,9)
   tcostA(i,j,12) = costB(m,n,10)
   tcostA(i,j,13) = costB(m,n,11)
end if
end do
end do

if(costA(1,1,1).eq.-1.d0)then
do i = 1,13
costA(1,1,i) = tcostA(1,1,i)
end do
elseif(tcostA(1,1,1).lt.costA(1,1,1))then
do i = 1,13
costA(1,1,i) = tcostA(1,1,i)
end do
end if

end do !k loop
costmin = costA(1,1,1)
write(*,*) 'The minimum cost is ', costmin

B1 = int(costA(1,1,2))
B2 = int(costA(1,1,3))
C1 = int(costA(1,1,4))
C2 = int(costA(1,1,5))
Dp1 = int(costA(1,1,6))
Dp2 = int(costA(1,1,7))
```

```
E1 = int(costA(1,1,8))
E2 = int(costA(1,1,9))
F1 = int(costA(1,1,10))
F2 = int(costA(1,1,11))
G1 = int(costA(1,1,12))
G2 = int(costA(1,1,13))

Bptx = grid(B1+1,1)
Bpty = grid(B1+1,2)
Cptx = grid(C1+3,1)
Cpty = grid(C1+3,2)
Dptx = grid(Dp1+6,1)
Dpty = grid(Dp1+6,2)
Eptx = grid(E1+10,1)
Epty = grid(E1+10,2)
Fptx = grid(F1+13,1)
Fpty = grid(F1+13,2)

V2Bx = grid(B2+17,1)
V2By = grid(B2+17,2)
V2Cx = grid(C2+19,1)
V2Cy = grid(C2+19,2)
V2Dx = grid(Dp2+22,1)
V2Dy = grid(Dp2+22,2)
V2Ex = grid(E2+26,1)
V2Ey = grid(E2+26,2)
V2Fx = grid(F2+31,1)
V2Fy = grid(F2+31,2)
V2Gx = grid(G2+37,1)
V2Gy = grid(G2+37,2)

write(30,*) grid(1,1), grid(1,2)
write(30,*) Bptx, Bpty
write(30,*) Cptx, Cpty
write(30,*) Dptx, Dpty
write(30,*) Eptx, Epty
write(30,*) Fptx, Fpty
write(30,*) xT, yT
write(30,*) grid(17,1), grid(17,2)
```

```
      write(30,*) V2Bx, V2By
      write(30,*) V2Cx, V2Cy
      write(30,*) V2Dx, V2Dy
      write(30,*) V2Ex, V2Ey
      write(30,*) V2Fx, V2Fy
      write(30,*) V2Gx, V2Gy
      write(30,*) costmin, t

      close(30)

      open(unit=10,file='timeroff.txt')

      close(10)
      stop
      end



      !####################################################################!
      !--------------------------SUBROUTINES-----------------------------!
      !####################################################################!

      !RK(4,4) Subroutine
      !====================================================================!
       subroutine RK4(T,DT,NEQ,X,param,D1,D2,D3)
       implicit double precision (a-h,o-z)
       integer NEQ

       dimension X(NEQ),D1(NEQ), D2(NEQ), D3(NEQ),param(10)

       call deriv(X,D1,NEQ,param)
       do i = 1,NEQ
       D1(i) = D1(i)*DT
       D2(i) = X(i) + 0.5d0*D1(i)
       end do

!       TT = T + 0.5d0*DT
       call deriv(D2,D3,NEQ,param)
```

```
 do i = 1,NEQ
 D3(i) = D3(i)*DT
 D1(i) = D1(i) + 2.0d0*D3(i)
 D2(i) = X(i) + .5d0*D3(i)
 end do

 call deriv(D2,D3,NEQ,param)
 do i = 1,NEQ
 D3(i) = D3(i)*DT
 D1(i) = D1(i) + 2.0d0*D3(i)
 D2(i) = X(i) + D3(i)
 end do

 T = T + DT

 call deriv(D2,D3,NEQ,param)

 do i = 1,NEQ
 X(i) = X(i) + (D1(i) + D3(i)*DT)/6.0d0
 end do

 return
 end

!Derivative Subroutine
!================================================================!
 subroutine deriv(X,DX,NEQ,param)
 implicit double precision (a-h,m-z)
 integer NEQ
 dimension X(NEQ), DX(NEQ),param(10)

v = param(3)
psi1 = param(5)
psi2 = param(6)
xT = param(1)
yT = param(2)
param(7) = X(1)
param(8) = X(2)
param(9) = X(3)
```

```
 param(10)= X(4)

 r1=sqrt((xT-X(1))**2.d0+(yT-X(2))**2.d0)
 theta1=atan((yT-X(2))/(xT-X(1)))    !bearing measured by vehicle1
 r2=sqrt((xT-X(3))**2.d0+(yT-X(4))**2.d0)
 theta2=atan((yT-X(4))/(xT-X(3)))    !bearing measured by vehicle2

 call thetafix(param,theta1,theta2)

 sigmaxT =(101.d0*r1**2.d0*r2**4.d0+99.d0*r1**2.d0*r2**4.d0*cos(
&2.d0*theta1)+101.d0*r1**4.d0*r2**2.d0+99.d0*r1**4.d0*r2**2.d0*cos(
&2.d0*theta2))/(-980100.d0*r1**2.d0*r2**2.d0*cos(2.d0*theta1-2.d0*
&theta2)+20000.d0*r1**4.d0+20000.d0*r2**4.d0+1020100.d0*r1**2.d0*
&r2**2.d0)

 sigmayT =(99.d0*r1**2.d0*r2**4.d0*cos(2.d0*theta1)-101.d0*r1**2.d0
&*r2**4.d0+99.d0*r1**4.d0*r2**2.d0*cos(2.d0*theta2)-101.d0*r1**
&4.d0*r2**2.d0)/(980100.d0*r1**2.d0*r2**2.d0*cos(2.d0*theta1-2.d0*
&theta2)-20000.d0*r1**4.d0-20000.d0*r2**4.d0-1020100.d0*r1**2.d0
&*r2**2.d0)

  DX(1) =  v*cos(psi1)           !xdot1
  DX(2) =  v*sin(psi1)           !ydot1
  DX(3) =  v*cos(psi2)           !xdot2
  DX(4) =  v*sin(psi2)           !ydot2
  DX(5) = sigmaxT + sigmayT      !Jdot

  return
  end

 !Theta and Psi Subroutine
 !=============================================================!
 subroutine thetafix(param,theta1,theta2)
 implicit double precision (a-h,o-z)
 dimension param(10)

 xT = param(1)
 yT = param(2)
 x1 = param(7)
```

```
y1 = param(8)
x2 = param(9)
y2 = param(10)

pi = acos(-1.d0)

if (xT.lt.x1) then
   if (yT.ge.y1) then
   theta1 = pi + theta1
   else
   theta1 = -pi + theta1
   end if
end if

if (xT.lt.x2) then
   if (yT.ge.y2) then
   theta2 = pi + theta2
   else
   theta2 = -pi + theta2
   end if
end if

return
end
```

Appendix B

MATLAB Code Used for Plots and Error Estimation

```
function coop2(pr,pb)
clc
close all

format long
dlmread('CoopControl.txt');
x = ans(1:58,1);
y = ans(1:58,2);
x01 = x(1);
y01 = y(1);
x02 = x(17);
y02 = y(17);

c = ans(59,1);
t = ans(59,2);
T = t;
clear ans;

for m = 1:6
    if x(m+45)-x(m+44) > 0
        f(m) = 1;
    else
        f(m) = -1;
    end
end
if sum(f) == 6 | sum(f) == -6
    polym = polyfit(x(45:51),y(45:51),pr);
    xim = linspace(min(x(45:51)),max(x(45:51)),10*T);
    yim = polyval(polym,xim);
else
    polym = polyfit(y(45:51),x(45:51),pr);
    yim = linspace(min(y(45:51)),max(y(45:51)),10*T);
    xim = polyval(polym,yim);
end

for n = 1:6
    if x(n+52)-x(n+51) > 0
        g(n) = 1;
    else
```

```
        g(n) = -1;
    end
end
if sum(g) == 6 | sum(g) == -6
    polyn = polyfit(x(52:58),y(52:58),pb);
    xin = linspace(min(x(52:58)),max(x(52:58)),10*T);
    yin = polyval(polyn,xin);
else
    polyn = polyfit(y(52:58),x(52:58),pb);
    yin = linspace(min(y(52:58)),max(y(52:58)),10*T);
    xin = polyval(polyn,yin);
end
xT = 0;
yT = 0;

% Create polynomial curve fit for vehicle one.
hold on
plot(xim,yim,'k','Linewidth',2)
s1 = [xim; yim]';
% Create polynomial curve fit for vehicle two.
hold on
plot(xin,yin,'k--','Linewidth',2)
s2 = [xin;yin]';
% Recreate graph
figure
xlim([min(x)-abs(min(x)-max(x))/20 max(x)+abs(min(x)-max(x))/20])
ylim([min(y)-abs(min(y)-max(y))/20 max(y)+abs(min(y)-max(y))/20])
xlabel('x (ft)'); ylabel('y (ft)');
movie(ans)

xT = 0;
yT = 0;
x11 = xim;
y11 = yim;
x22 = xin;
y22 = yin;

numxs = length(x11);
step = T/(numxs-1);

%Properly order state from x0 to xT
if and(y02 > yT, y01 < yT)
for jj = 1:numxs
    x2(jj) = x22(numxs-jj+1);
    y2(jj) = y22(numxs-jj+1);
    x1(jj) = x11(jj);
    y1(jj) = y11(jj);
```

```
end
end
if and(y01 > yT, y02 < yT)
for ii = 1:numxs
    x1(ii) = x11(numxs-ii+1);
    y1(ii) = y11(numxs-ii+1);
    x2(ii) = x22(ii);
    y2(ii) = y22(ii);

end
end
if and(y01 > yT, y02 > yT)
    for ii = 1:numxs
    x1(ii) = x11(numxs-ii+1);
    y1(ii) = y11(numxs-ii+1);
    x2(ii) = x22(numxs-ii+1);
    y2(ii) = y22(numxs-ii+1);
    end
end
if and(y01<yT,y02<yT)
    x1 = x11;
    y1 = y11;
    x2 = x22;
    y2 = y22;
end

state1 = [x1; y1]';
state2 = [x2; y2]';

randn('state',500);

%Error Estimation

for k = 1:numxs
H = [eye(2);eye(2)];
theta1(k) = atan2((yT-y1(k)),(xT-x1(k)));
r1(k) = sqrt((xT-x1(k))^2+(yT-y1(k))^2);
sigd1(k) = 0.1*r1(k);
sigc1(k) = 0.01*r1(k);
wd1 = sigd1(k)*randn;
wc1 = sigc1(k)*randn;
cosp1 = [cos(theta1(k)) sin(theta1(k)); -sin(theta1(k)) cos(theta1(k))];
cosm1 = [cos(theta1(k)) -sin(theta1(k)); sin(theta1(k)) cos(theta1(k))];
xT1 = [xT;yT]+cosp1*[wd1;wc1]; %target seen by veh1
sigm1 = [.1*r1(k)^2 0; 0 .01*r1(k)^2];
P1 = cosp1*sigm1*cosm1;  %uncertainty from veh1
```

```
%munition 2
theta2(k) = atan2((yT-y2(k)),(xT-x2(k)));
r2(k) = sqrt((xT-x2(k))^2+(yT-y2(k))^2);
sigd2(k) = 0.1*r2(k);
sigc2(k) = 0.01*r2(k);
wd2 = sigd2(k)*randn;
wc2 = sigc2(k)*randn;
cosp2 = [cos(theta2(k)) sin(theta2(k)); -sin(theta2(k)) cos(theta2(k))];
cosm2 = [cos(theta2(k)) -sin(theta2(k)); sin(theta2(k)) cos(theta2(k))];
xT2 = [xT;yT]+cosp2*[wd2;wc2]; %target seen by veh1
sigm2 = [sigd2(k)^2 0; 0 sigc2(k)^2];
P2 = cosp2*sigm2*cosm2; %uncertainty from veh2
R = [P1 zeros(2);zeros(2) P2];

    if k == 1
    P(:,:,k) = inv(inv(P1)+inv(P2));
    xThat(:,k) = P(:,:,k)*(inv(P1)*xT1 + inv(P2)*xT2);
    else
    K=P(:,:,k-1)*H'*inv(H*P(:,:,k-1)*H'+R);
    xThat(:,k) = xThat(:,k-1) + K*([xT1 ; xT2]-H*xThat(:,k-1));
    P(:,:,k)=inv(inv(P(:,:,k-1))+H'*inv(R)*H);
    Pkeigs = eig(P(:,:,k));
    Area(k) = sqrt(Pkeigs(1)*Pkeigs(2))*pi;
    end

    sigx(k)=sqrt(P(1,1,k));
    sigy(k)=sqrt(P(2,2,k));

end

theta1deg = theta1*180/pi;
theta2deg = theta2*180/pi;
thetas = [theta1deg; theta2deg]';
alength=length(Area);
ti = [0:step:T];
elipse = [ti; Area]';
tfmin2 = ti(length(ti)-20)
Atfmin2 = Area(length(Area)-20)

figure
subplot(2,1,1)
plot(ti,xThat(1,:),'k')
hold on
plot(ti,3*sigx,'k--')
plot(ti,-3*sigx,'k--')
ylabel('x (ft)')
%title('Estimation errors')
```

```
subplot(2,1,2)
plot(ti,xThat(2,:),'k')
hold on
plot(ti,3*sigy,'k--')
plot(ti,-3*sigy,'k--')
xlabel('time (sec)')
ylabel('y (ft)')
axis([0 T -100 100])
```