The Pennsylvania State University
The Graduate School

UNMANNED AERIAL VEHICLE TRAJECTORY PLANNING
WITH DIRECT METHODS

A Dissertation in
Aerospace Engineering
by
Brian Geiger

© 2009 Brian Geiger

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2009
The dissertation of Brian Geiger was reviewed and approved* by the following:

Joseph F. Horn  
Associate Professor of Aerospace Engineering  
Dissertation Advisor, Chair of Committee

Lyle N. Long  
Distinguished Professor of Aerospace Engineering and Mathematics

Jack W. Langelaan  
Assistant Professor of Aerospace Engineering

Alok Sinha  
Professor of Mechanical Engineering

George A. Lesieutre  
Professor of Aerospace Engineering  
Head of the Department of Aerospace Engineering

*Signatures are on file in the Graduate School.
Abstract

A real-time method for trajectory optimization to maximize surveillance time of a fixed or moving ground target by one or more unmanned aerial vehicles (UAVs) is presented. The method accounts for performance limits of the aircraft, intrinsic properties of the camera, and external disturbances such as wind. Direct collocation with nonlinear programming is used to implement the method in simulation and onboard the Penn State/Applied Research Lab’s testbed UAV. Flight test results compare well with simulation. Both stationary targets and moving targets, such as a low flying UAV, were successfully tracked in flight test.

In addition, a new method using a neural network approximation is presented that removes the need for collocation and numerical derivative calculation. Neural networks are used to approximate the objective and dynamics functions in the optimization problem which allows for reduced computation requirements. The approximation reduces the size of the resulting nonlinear programming problem compared to direct collocation or pseudospectral methods. This method is shown to be faster than direct collocation and pseudospectral methods using numerical or automatic derivative techniques. The neural network approximation is also shown to be faster than analytical derivatives but by a lesser factor. Comparative results are presented showing similar accuracy for all methods. The method is modular and enables application to problems of the same class without network retraining.
# Table of Contents

## List of Figures

## List of Tables

## List of Symbols

## Acknowledgments

## Chapter 1

**Introduction**

1.1 Introduction ................................................. 1
1.2 Contributions ................................................ 2
1.3 Methods Used in Trajectory Optimization ..................... 3
   1.3.1 Indirect Methods ........................................ 3
   1.3.2 Direct Methods .......................................... 4
     1.3.2.1 Direct Shooting ...................................... 4
     1.3.2.2 Direct Collocation ................................... 5
     1.3.2.3 Pseudospectral Methods .............................. 6
   1.3.3 Mixed Integer Linear Programming ....................... 6
   1.3.4 Dynamic Programming .................................... 7
   1.3.5 Genetic Algorithms ...................................... 7
1.4 UAV Specific Research ........................................ 8
   1.4.1 Direct Collocation ...................................... 8
   1.4.2 Pseudospectral Methods ................................ 9
   1.4.3 Mixed Integer Linear Programming ...................... 9
   1.4.4 Dynamic Programming .................................... 10
   1.4.5 Genetic Algorithms ...................................... 10
Chapter 2

**Direct Nonlinear Trajectory Optimization**

- **2.1 The Basic Problem**
- **2.2 Direct Collocation with Nonlinear Programming**
- **2.3 Pseudospectral Methods**
- **2.4 Examples**
  - **2.4.1 Brachistochrone Example**
  - **2.4.2 Moon Lander Example**
- **2.5 Receding Horizon**

Chapter 3

**Neural Network Method**

- **3.1 Introduction**
- **3.2 Method Motivation and Overview**
- **3.3 Previous and Related Research**
- **3.4 Neural Network Formulation**
  - **3.4.1 Derivative Calculation**
- **3.5 Training**

Chapter 4

**UAV Path Planning Surveillance Problem**

- **4.1 Equations of Motion**
- **4.2 Discretization Scheme**
  - **4.2.1 Direct Collocation and Pseudospectral Parameterization**
  - **4.2.2 Neural Network Approximation Parameterization**
- **4.3 Constraints**
- **4.4 Objective Function**
  - **4.4.1 Integrating External World Data**
  - **4.4.2 Perspective Driven Surveillance**
- **4.5 Neural Network Specific Details**
  - **4.5.1 Network Structure**
  - **4.5.2 Neural Network Training**
- **4.6 Simulation Results**
- **4.7 Convergence Results**
  - **4.7.1 Discretization Convergence**
  - **4.7.2 Receding Horizon Convergence**
- **4.8 Comparative Results**
  - **4.8.1 Training**
B.1 Dynamics Network .................................................. 120
B.2 Distance-to-target Network ........................................ 124
B.3 Target-in-view Network ............................................. 127
B.4 Scaling ................................................................. 130

Bibliography ..................................................................... 132
# List of Figures

1.1 ARL/PSU Sig Kadet Senior UAV and Tankbot .......................... 12

2.1 Illustration of direct collocation discretization scheme and defect . 16
2.2 Illustration of Chebyshev pseudospectral method discretization . 19
2.3 Brachistochrone problem ........................................... 21
2.4 Brachistochrone problem ........................................... 22
2.5 Moon lander problem ............................................... 24

3.1 Illustration of the segmented state trajectory ...................... 29

4.1 Axes used in the homography ......................................... 40
4.2 Target-in-view cost functions ....................................... 42
4.3 Path data illustration ............................................... 43
4.4 Illustration of the perspective driven objective function .......... 45
4.5 Neural network illustrations for the target surveillance problem ... 46
4.6 Common artificial neural network transfer functions .............. 47
4.7 Simulation of a single UAV and stationary target ................ 49
4.8 Steady wind and target motion comparison .......................... 50
4.9 Simulation of a single UAV and fast moving target ............... 50
4.10 Simulation of a single UAV and slow moving target .............. 51
4.11 Simulation of a UAV pair and stationary target .................. 52
4.12 Distance from UAV to target ....................................... 52
4.13 Path planner simulation with road data ............................ 53
4.14 Path planner simulation with road data and fast target .......... 54
4.15 Simulation of a single UAV and stationary target (perspective driven) ............................................... 55
4.16 Simulation of a single UAV and slow moving target (perspective driven) ............................................... 56
4.17 Observation ratio time history, stationary target ................. 57
4.18 Network training regression results (normalized) ................. 62
4.19 Stationary target comparison results ................................ 64
4.20 Controls (longitudinal acceleration and bank angle) ............ 65
4.21 Moving target comparison results, 10 ft/sec .................................. 66
4.22 Moving target comparison results, 20 ft/sec .................................. 67
4.23 Moving target comparison results, 25 ft/sec .................................. 68
4.24 Moving target comparison results, 27 ft/sec .................................. 69
4.25 Moving target comparison results, 35 ft/sec .................................. 70
4.26 Relative path computation time (16-sec horizon) ............................ 71
4.27 Relative path computation time (30-sec horizon) ............................ 72
4.28 Ground track and coverage timeline for two UAVS and a target ...... 72

5.1 The Applied Research Lab/Penn State UAV Testbed ...................... 77
5.2 Kadet UAV construction photos ....................................................... 78
5.3 Block diagram for the airborne and ground systems ....................... 79
5.4 Airfield Photos and test target photos ............................................. 80
5.5 Stationary target observation in calm winds .................................. 85
5.6 Stationary target observation with 5 knots wind from the west ......... 87
5.7 Observation of a walking person ...................................................... 88
5.8 Observation of a moving truck ...................................................... 90
5.9 Initial acquisition of the target UAV from a parking orbit ................ 91
5.10 Tracking of the target UAV around a rectangular path .................. 92
5.11 Tracking of the target UAV around a figure-8 pattern .................... 94
5.12 Tracking of the target UAV around a figure-8 pattern (match speed) . 95
5.13 Sample frame captures from the onboard video ............................ 96
5.14 Schematic of the hardware-in-the-loop simulation ......................... 97
5.15 Webcam HIL rig and calibration result ........................................... 98
5.16 Comparison of actual and simulated aerial views ......................... 99
5.17 Integrated path planner and target geolocation in HIL simulation ...... 100
5.18 Histogram of processing times ..................................................... 101

B.1 Dynamics network, Layer 1 ......................................................... 121
B.2 Dynamics network, Layer 2 ......................................................... 121
B.3 Dynamics network, Layer 3 ......................................................... 122
B.4 Distance-to-target network, Layer 1 ............................................. 124
B.5 Distance-to-target network, Layer 2 ............................................. 124
B.6 Distance-to-target network, Layer 3 ............................................. 125
B.7 Target-in-view network, Layer 1 .................................................. 127
B.8 Target-in-view network, Layer 2 .................................................. 128
B.9 Target-in-view network, Layer 3 .................................................. 128
## List of Tables

4.1 Objective Function Listing ................................. 42
4.2 UAV performance constraints ............................. 48
4.3 Target coverage convergence, 16 second horizon ........ 58
4.4 Target coverage convergence, 30 second horizon .......... 59
4.5 Target coverage convergence for increasing horizon, 1 sec node spacing 59
4.6 Target coverage convergence for increasing horizon, 2 sec node spacing 60
4.7 Target coverage convergence for increasing horizon, 4 sec node spacing 60
4.8 Path generation time (mean and standard deviation) ...... 73
4.9 Path generation time relative to neural network method ... 74
4.10 Target coverage (percentage of total simulation time) .... 74

5.1 Path planner configurations used in flight testing ........ 83
5.2 UAV performance constraints ............................. 84

A.1 Leading-point partials for the North direction for $J_{dist}$ .... 112
A.2 Trailing-point partials for the North direction for $J_{dist}$ .... 112
A.3 Leading-point partials for the East direction for $J_{dist}$ .... 113
A.4 Trailing-point partials for the East direction for $J_{dist}$ .... 113

B.1 Dynamics network, Layer 1 .................................. 123
B.2 Dynamics network, Layer 2 .................................. 123
B.3 Dynamics network, Layer 3 .................................. 123
B.4 Distance-to-target network, Layer 1 ......................... 126
B.5 Distance-to-target network, Layer 2 ......................... 126
B.6 Distance-to-target network, Layer 3 ......................... 126
B.7 Target-in-view network, Layer 1 ............................ 129
B.8 Target-in-view network, Layer 2 ............................ 129
B.9 Target-in-view network, Layer 3 ............................ 129
List of Symbols

\[ b \] Neural network bias vector

\[ C_m \] Rotation matrix relating the camera axes to the UAV axes

\[ D \] Pseudospectral method differentiation matrix

\[ \Delta \] Defect vector

\[ g \] Acceleration due to gravity

\[ \gamma \] Azimuth (rad)

\[ H \] Homography matrix

\[ J \] General scalar objective function

\[ J_{tiv} \] “Target-in-view” objective function

\[ K \] Camera intrinsic parameters matrix

\[ \lambda \] Elevation (rad)

\[ k \] Active rows of the states Jacobian

\[ k_{ac} \] Active rows of state constraints Jacobian

\[ N \] Number of nodes

\[ n \] Number of UAVs

\[ \nabla \] Gradient operator

\[ P_m \] Overall parameter vector
\( \mathbf{p} \) Position of the UAV with respect to the target, \( \mathbf{p} = \mathbf{p}_{uav} - \mathbf{p}_{tgt} \)

\( \mathbf{p}_c \) Control vector for \( n \) UAVs

\( \mathbf{p}_s \) State vector for \( n \) UAVs

\( \psi \) Heading (\( rad \))

\( q \) Number of targets

\( \mathbf{q} \) Normalized pixel coordinates, \( [q_x \ q_y \ q_z]^T \)

\( \mathbf{\bar{q}} \) Unnormalized pixel coordinates, \( [\bar{q}_x \ \bar{q}_y \ \bar{q}_z]^T \)

\( \mathbf{p}_r \) Offset attraction point used with the perspective driven method

\( R \) Rotation matrix relating UAV body axes to world North-East-Down coordinates

\( s \) Nondimensional time

\( \tau \) Segment length (\( s \))

\( T_h \) Horizon time (\( s \))

\( \mathbf{u} \) UAV Control vector

\( u_a \) Longitudinal acceleration command (\( m/s^2 \))

\( \mathbf{u}_i \) Control vector at the \( i_{th} \) node

\( u_{\phi} \) Bank angle command (\( radians \))

\( V_t \) True airspeed (\( m/s \))

\( \mathbf{x}_i \) State vector at the \( i_{th} \) node

\( \mathbf{x}_t \) Target state vector

\( \mathbf{x}_u \) UAV state vector

\( W \) Neural network weight matrix

\( x, x_t \) North position of UAV and target, respectively (\( m \))

\( Y(\cdot) \) Multi-input, feedforward artificial neural network

\( y, y_t \) East position of UAV and target, respectively (\( m \))
I thank my advisor Dr. Joe Horn for guidance, advice, knowledge, and opportunities given throughout both my Masters and Ph.D. work. I also thank my fellow students Len Lopes, Chris Hennes, Wei Guo, James Erwin, James Ross, and Greg Sinsley. I am especially grateful to Al Niessner. Without Al’s time and dedication, the Penn State UAV Team would not exist. Dr. Lyle Long and Dr. Jack Lange-laan also provided thoughts and advice for this project. I also thank Mike Roeckel and Dr. Richard Stern for guidance and financial support through the Applied Research Laboratory.

Finally, I deeply grateful to my fiancée Laura for her love and support.
Dedication

For my dad.

...and my Grandma, from whom I got all my smarts.
Chapter 1

Introduction

1.1 Introduction

Past and current research into UAV path planning has grown from the demand for increased autonomous behavior capability from UAVs. Given the ability to plan a trajectory based on human or sensor input, the UAV gains the capability to avoid obstructions or other aircraft, track targets, optimize certain performance characteristics such as endurance, and generally adapt to a dynamic situation. There has been much research into numerical trajectory planning methods. These methods can be divided into two main subsets: direct and indirect methods [1]. Indirect methods are based on calculus of variations, while direct methods transform the problem into a nonlinear programming problem. Generally, direct methods are preferred over indirect methods due to simplicity. Indirect methods require the formulation of the optimality conditions which can be complex. In addition, the initial guess required to solve the resulting two-point boundary value problem can be difficult to find. Direct methods do not require the derivation of optimality conditions and can be solved as a nonlinear programming problem, for which methods of solution are mature and relatively robust. Other methods of trajectory optimization that are receiving increased attention today include genetic algorithms, linear programming, and Lyapunov vector field methods.

The general trajectory optimization problem is described as a minimization of a scalar function $J$ subject to the equations of motion and optionally subject to state trajectory constraints, control constraints, initial and final conditions, and
time constraints. The optimization problem is stated as such:

Find the optimal control that produces the optimal trajectory which minimizes

\[ J = E(x(t_f), t_f) + \int_{t_0}^{t_f} F(x(t), u(t), t) dt \]  \hspace{1cm} (1.1)

subject dynamic equations

\[ \dot{x} = \frac{dx}{dt} = f(x(t), u(t), t) \]  \hspace{1cm} (1.2)

and state, control, and time (equality or inequality) constraints

\[ c_l \leq c(x(t), u(t), t) \leq c_u \]  \hspace{1cm} (1.3)

Equation 1.1 is known as the Bolza cost functional. Its components are the Mayer cost \( E(\cdot) \), based on the final states and time, and the Lagrange cost \( F(\cdot) \), based on the integration along the state trajectory.

### 1.2 Contributions

The focus of this work is to develop a direct trajectory optimization method to be used onboard an unmanned aerial vehicle in real-time for a surveillance task. One of the main missions for UAVs is surveillance; an automatic method to generate a near optimal path tailored to UAV and sensor hardware would be useful. Toward this goal, a new real-time implementation of a direct collocation trajectory optimization is presented. The method will be used to direct a UAV to maximize the time a target is visible to an onboard camera, given the intrinsic parameters of the camera (focal length, sensor size, orientation) and performance limits of the aircraft. The real-time direct collocation method requires the use of analytic derivatives in the constraint and objective derivative calculations which are used by the nonlinear solver. The direct collocation method is tested in both simulation and in flight test aboard the Applied Research Laboratory/Penn State (ARL/PSU) testbed UAV, a heavily modified Sig Kadet Senior.

Because derivation of analytical derivatives for nontrivial nonlinear optimization problems can be tedious, error-prone, and inflexible, a new neural network
based trajectory optimization method is also presented. The new method uses feed-forward neural networks to approximate the dynamics and objective functions. Feed-forward neural networks have been shown to be universal function approximators; they can also be used to compute derivatives of the function they are approximating. Using this property, the requirement for specific analytical derivatives is removed for real-time operation with the neural network method. In addition, the method is set up to remove collocation constraints which reduces the problem size compared to the direct collocation and pseudospectral methods.

1.3 Methods Used in Trajectory Optimization

This section provides a quick overview of several methods currently used in trajectory optimization.

1.3.1 Indirect Methods

The following is taken from [2]. Indirect methods are based on calculus of variations. The Hamiltonian is defined as

$$H = F(x, u, t) + \lambda^T f$$  \hspace{1cm} (1.4)

where $\lambda$ adjoins the equations of motion constraints to the path objective function. For this brief overview, path constraints are neglected. The conditions for optimality require that $\lambda$ satisfy

$$\dot{\lambda} = - \frac{\partial H^T}{\partial x}$$ \hspace{1cm} (1.5)

Additionally,

$$\frac{\partial H}{\partial u} = 0$$ \hspace{1cm} (1.6)

To obtain the adjoint vector, Equation 1.7 is integrated back in time starting from the terminal condition given in Equation 1.8.

$$\dot{\lambda} = - \left( \frac{\partial f}{\partial x} \right)^T \lambda - \left[ \frac{\partial L}{\partial x} \right]$$ \hspace{1cm} (1.7)
\lambda(t_f) = \frac{\partial E(x_{t_f}, t_{t_f})}{\partial x} \tag{1.8}

Now, the optimal control can be computed over the time period by solving for \( u \).

\[ \frac{\partial F^T}{\partial u} + \frac{\partial f^T}{\partial u} \lambda = 0 \tag{1.9} \]

A numerical method used in solution of the above equations is called indirect shooting. Indirect shooting iteratively solves the initial value problem and then evaluates constraints to adjust initial conditions. It suffers from a high sensitivity in the final results from small changes in the initial conditions. Betts [1] notes that indirect shooting is best used when the dynamics are benign due to this high initial condition sensitivity. An example of benign dynamics is a low-thrust orbit trajectory where the states evolve slowly over a long time period. Finally, the indirect shooting method requires a good initial guess which can be difficult to obtain.

### 1.3.2 Direct Methods

Unlike indirect methods, direct methods can be used to solve the optimal control problem without derivation of the necessary conditions for optimality. Direct methods operate by parameterizing the optimal control problem into a non-linear programming problem. Direct shooting methods integrate the state equations directly between the nodes, while direct collocation methods use a polynomial approximation to the integrated state equations between the nodes. The following sections give more detail.

#### 1.3.2.1 Direct Shooting

The direct shooting method integrates the trajectory during the optimization. The controls are piecewise between each point and can be piecewise constant, piecewise linear, etc. An integration is performed using the piecewise control and the constraints are then evaluated. Based on some function of the constraints, the initial conditions are adjusted and the process iterates until convergence [1]. A problem with direct shooting is the sensitivity of the final state to minute changes in the
initial state. In order to overcome this, the integration can be restarted at intermediate points, thus breaking the trajectory into smaller segments to which the direct shooting method can be more easily applied successfully. This is called multiple shooting. Direct shooting has been widely used and was originally developed for military space applications. [1].

1.3.2.2 Direct Collocation

Direct collocation differs slightly. It similarly discretizes the state trajectory into a series of points and approximates the segments between the points with polynomials. However, the difference between the first derivative of the interpolating polynomial at the midpoint of a segment and the first derivative calculated from the equations of motion at the segment midpoint is used as the defect. If this defect approaches zero, the interpolating polynomials are ensured to be a good approximation of the actual states.

Direct collocation was introduced by Dickmanns [3] as a general method for solving optimal control problems. The direct collocation method has seen wide use in spacecraft and satellite research. One of the first, Hargraves and Paris [4] applied it to a low earth orbit booster problem and a supersonic aircraft time-to-climb problem. The method has also been used in determining finite-thrust spacecraft trajectories [5] and optimal trajectories for multi-stage rockets in [6]. The problem of low-thrust spacecraft trajectories is investigated in [7, 8, 9, 10]. In particular, Reference [7] uses higher order Gauss-Lobatto quadrature rules instead of the original trapezoidal and Simpson rules. This change allows for increased accuracy with a reduced number of subintervals. The number of nonlinear programming parameters is smaller as a result. Rendezvous between two power limited spacecraft of different efficiencies are studied in [8], showing that DCNLP can be applied to more than one vehicle. Tang and Conway [9] studied low thrust interplanetary transfer using the collocation method and noted that no a priori assumptions about the optimal control solution were required to reach a solution. In [10], DCNLP is identified to be in a general class of direct transcription methods. The relationship between the original optimal control problem and the approximation afforded by DCNLP is examined. The method has also been used in satellite detumbling problems [11]. Again, the authors note that the initial guesses did not require any
information about the optimal control solution. Horie and Conway [12] studied optimal aeroassisted orbital intercept using DCNLP. They noted that the direct method allowed for easy inclusion of the complicated heating limit constraints required for this problem compared to the two point boundary value problem formulation, which is an indirect method. Additionally, they found that DCNLP has an advantage over the two-point boundary value problem formulation in terms of problem size and robustness.

Regardless of the use of direct shooting or direct collocation, the resulting problem is a nonlinear programming problem for which there are many solvers available. One such package is called SNOPT [13].

1.3.2.3 Pseudospectral Methods

Pseudospectral methods are a class of direct methods that discretize the states and controls of a trajectory optimization problem with unevenly spaced nodes. High-order (order equal to the number of nodes) polynomials of the Lagrange interpolating form are used to approximate the states and controls over the interval of interest. These methods offer increased accuracy with fewer nodes compared to direct methods due to the uneven discretization scheme. Razzaghi and Elnagar [14] were among the first to apply these methods to control of dynamic systems.

1.3.3 Mixed Integer Linear Programming

By writing the path planning problem as a series of discrete decisions between linear constraints, the problem can be expressed as linear constraints on a mixture of continuous and integer variables [15]. This formulation is know as a mixed integer linear program (MILP). The constraints that give rise to nonlinearity when solving aircraft path planning problems include limits on bank angle (turn rate) or maximum airspeed. This nonlinearity can be avoided by modeling the aircraft as a point mass acted on by a limited force and moving at a limited speed. In this way, a turn rate limit can be imposed by merely limiting the magnitudes of the force and velocity vectors that can act upon the point mass: \( \omega_{\text{max}} = f_{\text{max}}/(\mathbf{v}_{\text{max}}) \).

A limit on the magnitude of a 2-d vector is nonlinear (a circle), but it can be approximated linearly in the worst case by an inscribed square. For greater accu-
racy, more constraints can be imposed by inscribing a polygon with an increasing number of sides in the limiting circle. Collision avoidance constraints can simply be added by a rectangle around the aircraft. Once converted to MILP form, there exists a good number of solvers to calculate the solution. Reference [15] gives a good explanation of the method.

1.3.4 Dynamic Programming

Richard Bellman proposed this method in the 1950s. Its basis is breaking up the optimization problem into smaller and smaller subproblems until a simple case is reached that can be easily solved. Dynamic programming is most easily applied to discrete systems, however by using the Hamilton-Jacobi-Bellman equation, dynamic programming can be used with continuous systems [2]. In essence, dynamic programming applied to path planning is the calculation of the shortest path from a starting point to an ending point over a group of connected nodes. The cost to travel from one node to another adjacent node is the smallest decomposition possible and is simple to evaluate. Over the entire grid, the cost at all nodes to travel to all other adjacent node is calculated. Then starting backwards from the ending node, the minimal cost path to the starting node is calculated through summation.

1.3.5 Genetic Algorithms

Optimization using genetic algorithms has received increasing attention over the past years in the UAV field. It was initially presented by Holland [16] in the 1970s. This method starts with a population of possible solutions for a particular problem. A function is applied to each individual solution in order to evaluate their ‘fitness’ (i.e. how well the individual solves the problem). Then, using a series of operations inspired by genetics, a new population of solutions are generated. The basic genetic operators are selection (based on fitness), recombination (mating), and mutation (introducing small random changes). Genetic algorithms are global searches and are less susceptible to getting stuck in local minima. No specific initial conditions are required [17]. Reference [18] provides a tutorial in using genetic algorithms to solve multi-objective problems. An interesting thing mentioned in this article is the fact that using a genetic algorithm enables the discovery of multiple solutions
for different objective instead of using a weighted sum to combine them. A list of survey papers on the overall field of genetic algorithms is also given in [18]. Several uses of genetic algorithms in the UAV field are discussed in following section.

1.4 UAV Specific Research

1.4.1 Direct Collocation

Direct collocation has been applied to unmanned glider [19, 20] dynamic soaring and powered UAV [21] dynamic soaring, wherein the aircraft recovers energy from the atmosphere by cyclicly crossing wind velocity gradients. Dynamic soaring can be cast into an optimal control problem by seeking an energy neutral trajectory, a maximum altitude trajectory, or a minimum cycle time. The authors noted that DCNLP was well suited to solving this problem.

In related work, Qi and Zhao [22] use direct collocation to minimize the thrust required from an engine on a generic UAV when flying through a thermal. Using a two-dimensional thermal vertical velocity profile, the method finds a path with minimal constant thrust over a fixed distance, extracting energy from the thermal. Two specific behaviors were observed. In the first behavior, the UAV airspeed varies inversely with the thermal wind profile. With increasing vertical thermal wind speed, aircraft speed decreases. This allows the UAV to fly slowly across the upwelling air, while flying quickly through the downdraft. In the second behavior the UAV flies in union with the thermal, rising and falling as it crosses the thermal. The authors state these behaviours correspond with the patterns for optimal thermal crossing in sailplanes. In summary, thermals can be used to maximize either UAV range or speed.

Borrelli and others [23] investigated the method to provide centralized path planning along with collision avoidance for UAVs. The collision avoidance applies to both other aircraft and ground based obstacles. Both a collocation based non-linear programming method and a mixed integer linear programming method were investigated. An extensive set of tests were done across a large random problem initial conditions. It was shown that the MILP method was always faster than the NLP method, and that both methods produced optimal solutions with similar
costs. The MILP formulation uses simple linear dynamics, and the authors suggest that future work should focus on using MILP to initialize an NLP method with detailed dynamics.

Misovec, Inanc, Wohletz, and Murray [24] use a collocation method to generate flight paths while considering radar signature of the UAV. The method is not DCNLP, rather it uses the NTG [25] package developed at Caltech (collocation, but not solved with nonlinear programming). The work models radar detection based on the attitude of the aircraft relative to the radar station. An interesting characteristic of the detection model is that for paths flown directly toward the radar site (‘nose-in’) were less detectable than paths that approached at an oblique angle (‘nose-out’). Therefore, path heading plays an important part in detectability. The path planner incorporated this dependence of detectability on relative attitude to generate low observability paths while enabling the aircraft to reach all waypoints.

\subsection*{1.4.2 Pseudospectral Methods}
Williams [26] used the Legendre pseudospectral method for a three-dimensional, terrain following path planner. He demonstrated that with the use of analytical derivatives, the method could be made to run in real time. Yakimenko, Yunjun, and Basset [27] examined pseudospectral methods for use with short-time aircraft maneuvers and reported on several configurations of two MATLAB based optimization packages that were suitable for real time operation. An sub-optimal but very fast inverse dynamics method was also presented.

\subsection*{1.4.3 Mixed Integer Linear Programming}
How, King, and Kuwata [28] use MILP on their testbed of 8 UAVs. The UAVs are almost-ready-to-fly kits and use the Piccolo autopilot. A small part of their overall system, MILP is the basis for a receding horizon path planner. The algorithm is efficient enough to run in real time. All calculations are performed on the ground, and waypoints for the optimal path are sent to the UAV over the Piccolo 900 MHz link. The authors report several successful flight tests of two UAVs with the path planner operating, including tests in significant winds and formation flight.
Toupet and Mettler [29] use MILP in combination with dynamic programming for path planning of an unmanned helicopter. The area in which the helicopter is to fly is decomposed into cells free of obstacles. Then the cost to go from one cell to another can be calculated. Using cell size enables UAV performance constraints to be introduced. The particular objective for this work was to minimize travel time. Therefore, dynamic programming was used to find the shortest global path while avoiding obstacles. This path was then used in the receding horizon trajectory generation process to generate a local trajectory using MILP. The local trajectory is not global and may not even reach to the target waypoint. However since receding horizon control is used, the local trajectory will eventually include the target waypoint. For a typical city block environment, the authors report real time capability.

1.4.4 Dynamic Programming

Flint et al. [30] present an algorithm based on dynamic programming to generate near-optimal paths for several UAVs to follow to search for target cooperatively. The area to be searched is divided into a grid. UAV motion is modeled as a path made up of lines connecting various points in the grid. The UAV moves by way of a planning tree, and can make $m$ decisions at each node in the tree (1 time step). As dynamic programming works back from the final time, the method only solves ahead by a certain number of steps, thus avoiding the problem of expanding the search tree indefinitely. Other UAVs are treated as stochastic elements to model the possibility that they will have observed the target before the subject UAV. Thus the UAV would choose the path with the least likelihood of observations from other UAVs. The authors showed that their algorithm identified significantly more targets than a standard Zamboni search (so called because it resembles the path a Zamboni machine take over an ice rink).

1.4.5 Genetic Algorithms

Anderson et al. [31] use a genetic algorithm to maximize the number of targets seen, maximize the time they are seen, and minimize turn acceleration for a small UAV. The research demonstrated the advantages of the genetic algorithms’ global search
compared to a method that only ensured local optimality. It also demonstrated that GA outperforms a boundary reflection with random incidence angle method.

Shima et al. \cite{32} present research on using genetic algorithms to solve a multiple task assignment problem: assigning multiple UAVs to perform multiple tasks on multiple targets. The method produces feasible solutions quickly compared to more traditional methods. The authors mention that the method can make real time operation feasible.

Nikolos et al. \cite{33} use genetic algorithms for three dimensional path planning over terrain. There are two parts to the planner: offline and online. The offline planner generates a single B-spline path from the starting to the end points through an environment with known obstacles. The online planner builds on this path though radar readings to account for the unknown environment using a receding horizon strategy. For both planners, a potential field is used around obstacles to drive avoidance. Both versions were shown to be effective in generating a collision free path through terrain. Furthermore, the online planner effectively avoided any sensed obstacles, and produced a feasible path in a small number of generations, which the authors mention would be feasible for real time operation.

1.4.6 ARL/PSU UAV Group Research

A quick overview of the research areas investigated by the ARL/PSU UAV Group is given here. The main driver for the group’s research is the Applied Research Lab’s Intelligent Controller architecture \cite{34, 35, 36}. This software supports collaboration amongst heterogeneous vehicles, fuzzy decision making using continuous inference networks, and overall mission control. Because the software is behavior based, autonomous behaviors for the UAV needed to be developed. The focus of this dissertation is the creation of a path planning algorithm that can serve as a behavior for a target surveillance or search task. Initial results were presented in \cite{37}. Additional behaviors developed include vision based target identification and geolocation \cite{38, 39}, range and vision sensor fusion for terrain generation and target recognition \cite{40}, and basic collision avoidance for two UAVs (unpublished). With these unmanned aircraft behaviors, and in addition to ground vehicle-specific behaviors, the intelligent controller architecture can enable heterogeneous teams of
vehicles to communicate, delegate tasks based on vehicle capability, recover from vehicle loss through reallocation of mission tasks, and enable sensor fusion with data from multiple vehicles. Figure 1.1 is a photo of the Sig Kadet UAV next to an ARL tankbot which will be used in a future collaboration demonstration. Both vehicles support onboard operation of the Intelligent Controller software.

Figure 1.1. ARL/PSU Sig Kadet Senior UAV and Tankbot
Two common methods used in direct trajectory optimization are discussed in this chapter. These methods are discussed for three reasons: (1) to define some of the more popular methods in direct trajectory optimization (2) to compare with the neural network approximation method in Chapter 3 (3) to provide a basis for discussion on the application of each method to the UAV surveillance problem in Chapter 4.

2.1 The Basic Problem

The basic problem structure of a trajectory optimization problem can be described by finding the time dependent state vector $x(t)$, time dependent control vector $u(t)$, initial time $t_0$, and final time $t_f$ that minimize the Bolza cost functional.

$$J = E(x_f, t_f) + \int_{t_0}^{t_f} F(x(t), u(t), t)dt \quad (2.1)$$

The cost is subject to dynamics constraints,

$$\dot{x} = \frac{dx}{dt} = f(x(t), u(t), t) \quad (2.2)$$
state, control, and time (equality or inequality) constraints,

\[ c_l \leq c(x(t), u(t), t) \leq c_u \] (2.3)

and possibly fixed initial and final conditions.

\[ x(t_0) = x_0 \]
\[ x(t_f) = x_{tf} \] (2.4)

The general approach of direct, nonlinear trajectory optimization methods is to discretize the above equations and convert them into a nonlinear programming problem to allow an easier solution. Any number of the state or control variables can have their initial or final values specified, and the final time can be fixed or free to vary. Examples of two different methods of discretization and conversion are the direct collocation method and the pseudospectral method.

### 2.2 Direct Collocation with Nonlinear Programming

The direct collocation method uses equally spaced nodes in time to discretize the optimization problem given in Equations 2.1-2.3. This work uses the method presented by Hargraves and Paris [4]. Segments between each node are approximated with Hermite cubic interpolating polynomials. Hermite interpolating polynomials are defined in terms of the endpoint values and first derivatives at the endpoints. They are a natural fit with the optimization problem because the state vector \( x \) and state derivatives \( \dot{x} \) are readily available from the process model.

Let \( x(t) \) and \( u(t) \) be approximated by a piecewise Hermite cubic polynomial \( x_p(s) \) and a piecewise linear function \( u_p(s) \) where \( s \in [0, 1] \) is nondimensional segment time: \( s = (t - t_i)/\tau \) where \( t \) is time and \( t_i \) is time at the start of the \( i^{th} \)
segment. For the $i^{th}$ segment,

$$\begin{align*}
    x_{pi}(s) &= [2(x_i - x_{i+1}) + \dot{x}_i + \dot{x}_{i+1}]s^3 \\
    &+ 3(x_{i+1} - x_i) - 2\dot{x}_i - \dot{x}_{i+1}]s^2 \\
    &+ \dot{x}_i s + x_i \\
    u_{pi}(s) &= u_i + (u_{i+1} - u_i)s
\end{align*}$$

(2.5)

Given $n$ segments, each of length $\tau$ seconds, the problem is discretized with the states and controls defined at each node (segment endpoints). Using $x$ and $\dot{x}$, the Hermite cubics defining state behavior between the nodes are easily computed using Equation 2.5.

To ensure the approximating polynomials accurately represent the equations of motion, the derivative of the midpoint of each polynomial segment, $x_{pi}(0.5)$, is compared to the equations of motion evaluated using the states at the interpolated segment midpoint, $f[x_{pi}(0.5), u_{pi}(0.5)]$. This “collocation” of the approximated and actual derivatives gives the method its name. Carrying out the expansion for the interpolated states and controls at the collocation points (accounting for the nondimensionalized segment time$^1$) results in:

$$\begin{align*}
    x_{ci} &= x_{pi}(0.5) = \frac{1}{2}(x_i + x_{i+1}) + \frac{\tau}{8}(\dot{x}_i - \dot{x}_{i+1}) \\
    u_{ci} &= u_{pi}(0.5) = (u_i + u_{i+1})/2
\end{align*}$$

(2.6) (2.7)

Additionally, the slope$^2$ of the interpolated states at the collocation points are:

$$\dot{x}_{ci} = -\frac{3}{2\tau}(x_i - x_{i+1}) - \frac{1}{4}(\dot{x}_i + \dot{x}_{i+1})$$

(2.8)

The defect is defined as

$$\Delta = f(x_{ci}, u_{ci}) - \dot{x}_{ci}$$

(2.9)

When $\Delta$ is driven toward zero by choosing appropriate values of $x_i$ and $x_{i+1}$, the approximating polynomials will accurately represent the equations of motion if a

$^1$Because the approximating polynomials are written in nondimensional time, $\dot{x}_i = \tau f(x_i, u_i)$

$^2$Dimensional time derivatives are required here, so $\dot{x}_{ci} = \dot{x}_{pi}(0.5) = \frac{1}{\tau} \frac{d}{ds} x_{pi} \bigg|_{s=0.5}$
cubic polynomial is capable of doing so. Qualitatively, constraining the defect to zero places a lower bound on the number of nodes required to accurately represent a problem. Figure 2.1 shows an illustration of the discretization scheme and the defect. Thus, the constraints on the problem are $\Delta = 0$ and additional limits on the states and controls at each node to account for performance and mission limits.

Only cost functions of the Mayer type (terminal costs only) are considered in Hargraves and Paris [4]. When DCNLP is used with a Lagrange cost included in the Bolza cost functional, a numerical or analytic quadrature method is required (the pseudospectral method differs in this respect). In this work, a combination of analytical and numerical integration methods are used. Since the objective function can be expressed in terms of piecewise cubic polynomials, analytical integration may be simplified. However, when the objective function is complicated or highly nonlinear, an analytic expression for the integral may be too difficult to write. In this situation, numeric quadrature such as trapezoidal, Simpson, or Gauss are more appropriate. Accuracy is then adjusted by dividing up each segment in a given number of sub-nodes using the interpolating polynomials. Note that this directly affects the time required for the solution of the nonlinear programming problem.

Enright and Conway [10] present a slightly different formulation of DCNLP. Instead of using the derivatives at segment midpoints as the defect, an implicit Hermite-Simpson integration is used. The “Hermite-Simpson defects” are com-
puted recursively from the preceding node, state derivatives at the preceding and current node, and the interpolated midpoint. The defects differ from the original formulation by a constant factor of $2\tau/3$. The motivation for this change is to tie in DCNLP with a general class of transcription methods that had been studied previously and to mathematically provide a measure of the accuracy of the method.

2.3 Pseudospectral Methods

Spectral and pseudospectral methods were developed to solve partial differential equations and historically used in fluid dynamics applications. The solution is approximated by global, orthogonal polynomials. Spectral methods such as the Galerkin or Tau methods seek to approximate the solution with a weighted sum of a set of $N$ continuous functions. The weighting coefficients are time dependent. Pseudospectral or “collocation” methods use a set of discrete points by which constant weighting coefficients\(^3\) can be calculated based on the underlying approximating functions. The word “spectral” refers to the error convergence rate with respect to an increasing number of nodes. Spectral convergence means that error decreases faster than the rate of $O(N^{-m})$ for any $m > 0$ ($m$ is simply any positive number)\cite{41, 43, 44}, or in simpler terms: error decreases exponentially with increasing $N$.

With pseudospectral methods, states and controls are approximated with polynomials of degree $N$ in the Lagrange interpolating form. The various pseudospectral methods use different discretization schemes for the collocation points and are generally named for the scheme used. Some typical schemes include the Gauss quadrature nodes, extrema of Legendre polynomials, roots of first derivatives of Chebyshev polynomials (alternatively, the Chebyshev nodes are projections onto the $x$-axis of equally spaced points on the circumference of the unit circle \cite{44}). The selection and implementation of the discretization schemes has consequences in terms of the accuracy and smoothness of the costates \cite{45}. Compared to equally spaced nodes, these unequal node distributions offer increased performance and

\(^3\)For example, the Galerkin spectral method uses an approximation in the form of $f(x, t) = \sum_{n=1}^{N} x_i(t) L_{i}(x)$ compared to $x_p(\tau) = \sum_{n=1}^{N} x_i L_{i}(t)$, which is the form used in the pseudospectral method. \cite{41, 42, 43}
improved numerical behavior.

The problem setup is similar to the direct collocation method in that optimization is discretized. However, nodes unequally spaced in time are used and the entire trajectory is approximated with a single high-order Lagrange interpolating polynomial. The Chebyshev pseudospectral method\[46, 47\] is outlined here as it is used in following chapters. Other pseudospectral methods follow similar derivations \[48, 49, 50, 45, 51\].

The specific Chebyshev discretization is now discussed. The optimization problem is first transformed to the time interval on which the Chebyshev-Gauss-Lobatto points are defined \([-1, 1]\) so that the states with respect to time are given by \(x[t(\tau)]\) where \(t(-1) = t_0\) and \(t(1) = t_f\).

\[
t(\tau) = [(t_f - t_0)\tau + (t_f + t_0)]/2 \quad (2.10)
\]

The CGL points are used in Clenshaw-Curtis numerical quadrature, which is similar to Gaussian quadrature. These points are the extrema of the \(N^{th}\) order Chebyshev polynomial, which has a closed form expression.

\[
\tau_k = \cos(\pi k/N) \quad \text{for} \quad k = 0, 1, \ldots, N \quad (2.11)
\]

Choosing these interpolating points with the above distribution clusters more nodes at the endpoints which avoids the Runge phenomenon (divergence of the approximating polynomial at the endpoints). In addition, these nodes are used because the max-norm of the corresponding polynomial approximation of a function over the CGL points is minimized \[46\]. Figure 2.2 shows a discretized trajectory with six nodes.

Now, let \(x(t)\) and \(u(t)\) be approximated by the following form

\[
x_p(\tau) = \sum_{i=0}^{N} x_i \phi_i(t) \quad (2.12)
\]

\[
u_p(\tau) = \sum_{i=0}^{N} u_i \phi_i(t) \quad (2.13)
\]

The state approximation at the \(i^{th}\) node is given by an \(N^{th}\) order polynomial in
Figure 2.2. Illustration of Chebyshev pseudospectral method discretization

the Lagrange interpolating form [46]. Note that the \( i^{th} \) Lagrange interpolating polynomial is defined in terms of the states at the \( i^{th} \) node point. To ensure the approximating polynomial accurately represents the state equations, the derivatives at the nodes are computed with the \((N + 1) \times (N + 1)\) differentiation matrix.

\[
\dot{x}_{pi}(\tau_k) = \sum_{i=0}^{N} x_i \dot{\phi}_i(\tau_k) = \sum_{i=0}^{N} D_{ki} x_i
\]

(2.14)

\[
D_{ki} = \begin{cases} 
(c_k/c_i)[(-1)^{i+k} / (t_k - t_i)], & j \neq i \\
-t_k /[2(1 - t_k^2)], & 1 \leq j = k \leq N - 1 \\
(2N^2 + 1)/6, & j = k = 0 \\
-(2N^2 + 1)/6, & j = k = N 
\end{cases}
\]

(2.15)

\[
c_i = \begin{cases} 
2, & i = 0, N \\
1, & 1 \leq i \leq N - 1 
\end{cases}
\]

(2.16)

Thus, given a vector of states at each node, the derivatives of the approximating polynomial for that state at each node may be computed by multiplying the vector by \( D \). Note that the definition of \( D \) requires \( t_k = [1 \cdots -1] \). As shown in Reference [46], the differentiation matrix can be negated to reverse the sorting in time of the nodes. This makes the application of the Chebyshev pseudospectral method to trajectory optimization more intuitive as the time interval runs in the direction of perceived time.

To integrate the path cost, Clenshaw-Curtis [44, 52, 53, 54] quadrature is used. This simply involves the summation of the multiplication of the value of the objective function at each node by the corresponding node weight \( w_k \). Node weights
are given in [46, 47]. The discretized formulation of the optimization problem can now be written as: Find the parameters

\[ x_0, x_1, \ldots, x_N, \ u_0, u_1, \ldots, u_N \quad (2.17) \]

that minimize

\[ J = E(x_N, t_f) + \frac{t_f - t_0}{2} \sum_{i=0}^{N} F(x_i, u_i, t_i) w_i \quad (2.18) \]

subject to

\[ \frac{2}{(t_f - t_0)} XD - f(x, u, t) = 0 \quad (2.19) \]

\[ c_l \leq c(x_i, u_i, t_i) \leq c_u \quad (2.20) \]

An abuse of notation for Equation 2.19 is that \( X \) is a matrix of column vectors \( x_0 \ldots x_N \); the states at each node. Furthermore, \( f(x, u, t) \) is actually computing a matrix of \( \dot{x} \) column vectors (\( \dot{x}_0 \ldots \dot{x}_N \); the state derivatives at each node). Additionally, Equation 2.19 can be rewritten in an equivalent form to take advantage of linear versus nonlinear constraints [55].

\[ X \text{offdiag}(D) + X \text{diag}(D) - \frac{t_f - t_0}{2} f(x, u, t) = 0 \quad (2.21) \]

In this form, \( X \) multiplied by the off-diagonals of \( D \) is always constant and can be represented as linear constraints in the nonlinear programming problem. Solving a problem with linear constraints is usually faster than equivalent nonlinear constraints. The number of nonlinear constraints is significantly reduced and only consists of \( X \) multiplied by the matrix formed from the diagonal of \( D \). To see why this is true, note that only the states at a particular node affect the state derivatives at a particular node. This corresponds to the diagonal of \( D \) post-multiplying \( X \). Thus the off-diagonals post-multiplying \( X \) are constant and can be treated as linear constraints.
2.4 Examples

2.4.1 Brachistochrone Example

The following brachistochrone example problem compares results from the two methods. The brachistochrone problem is to find the shape of a wire such that a frictionless bead sliding down it moves from point A to B in minimum time. The problem has a single control input $\theta$ which is the angle of the wire with respect to time. The equations of motion for this problem are

\[
\begin{align*}
\dot{x} &= V \sin(\theta) \\
\dot{y} &= V \cos(\theta) \\
\dot{V} &= g \cos(\theta)
\end{align*}
\]

(2.22)

Horizontal and vertical position are given by $x$ and $y$, speed is $V$, and gravity is $g$.

![Figure 2.3. Brachistochrone problem](image)

The analytical solution to the brachistochrone problem is given by the equations
of a cycloid.

\[
\theta(\tau) = \frac{\pi \tau}{2 \tau_f}
\]

\[
x(\tau) = (g\tau_f/\pi)(\tau - (\tau_f/\pi) \sin[2\theta(\tau)])
\]

\[
y(\tau) = (2g\tau_f^2/\pi^2) \sin[\theta(\tau)]^2
\]

Given the desired final horizontal position, \( \tau_f \) can be computed.

\[
\tau_f = \sqrt{\frac{\pi x(\tau_f)}{g}}
\]

Letting \( x_0 = y_0 = 0 \), \( x_f = 5 \), and \( g = 1.0 \), the optimal final time is \( \sqrt{5\pi} = 3.9633272976 \). Eleven nodes are used for both methods. The direct collocation method results in an optimal time of 3.9633289715 with an absolute error of \( 1.7 \times 10^{-6} \). The Chebyshev method gives an optimal time of 3.9633272972 with an absolute error of \( 4 \times 10^{-10} \). The trajectory and control time history are shown in Figure 2.4. Note that for the \( x-y \) plot, spacing is distorted because the states are not shown with respect to time.

![Figure 2.4. Brachistochrone problem](image-url)
2.4.2 Moon Lander Example

The moon lander problem [56] is a simple three state system in which the objective is to land softly given an initial height, vertical speed, and mass while using minimum fuel. The single control is vertical thrust. Mass decreases as fuel is burned off when the thruster is engaged. The state equations are

\[
\begin{align*}
\dot{h} &= v \\
\dot{v} &= -g + \frac{T}{m} \\
\dot{m} &= -\frac{T}{I_{sp}g}
\end{align*}
\] (2.25)

Thrust, \( T \) is bounded from 0 to \( T_{max} \). Gravity \( g \) and specific impulse \( I_{sp} \) are given constants. Initial conditions are given and the desired final conditions for an intact landing are \( h_f = v_f = 0 \). Additionally, the lander must not run out of fuel, so \( m_f > 0 \). To minimize fuel burn, final mass is maximized.

The problem setup here is similar to [46]. Choosing \( g = I_{sp} = 1.0, T_{max} = 1.1, h_0 = m_0 = 1.0, \) and \( v_0 = -0.05 \), the problem is solved with both methods using 21 nodes. The direct collocation method results in a final mass of \( m_f = 0.1804 \) while the pseudospectral method results in \( m_f = 0.1800 \). The direct collocation method is able to more accurately capture the optimal bang-bang control than the pseudospectral method because the unevenly spaced nodes of the pseudospectral method are less dense in the middle interval which leads to increased approximation errors at the control switch point. However, multi-phase schemes for the pseudospectral method could eliminate this by adding a free phase boundary corresponding to the control switch point. Figure 2.5 shows a height-velocity diagram and thrust control time history.

2.5 Receding Horizon

In order to accommodate the real-time operation constraints on the trajectory optimization, a receding horizon approach is used. Instead of solving for the entire trajectory over the length of the mission duration, only the trajectory over a smaller horizon time \( T_h \) is computed. After a certain amount of time passes (the horizon
Figure 2.5. Moon lander problem

update interval, $T_u$, the optimization is repeated with new initial conditions. This affords two advantages: (1) changing situational information can be accounted for in the optimization (2) a compromise of accuracy versus computation time can be made. By decreasing $T_h$, the number of varied parameters in the optimization can be reduced, decreasing required computation time. Comparisons will be made between long and short horizon times to check for optimization convergence.
Chapter 3

Neural Network Method

3.1 Introduction

A limiting factor in real-time trajectory optimization is computational requirements. A main driver of the computational requirement is the calculation of objective and constraint derivatives for use in the solution of the nonlinear program. Generally, these derivatives are calculated through numerical methods which are slow and can be inaccurate. Analytical derivative calculation will provide significant speedup, but are tedious and error-prone. There are methods of solution of aircraft trajectory optimization problems that do not require the use of derivatives such as the work by Yakimenko [57]. However, initial and final states must be specified in that method. This section however focuses on a neural network approximation method that removes the need to numerically compute the objective and constraint derivatives and removes the need for collocation, thus reducing the nonlinear programming problem size. Similar to a multiple shooting method, the controls are parameterized and state equations are integrated between nodes. The method will be described in this chapter and is applied to a UAV surveillance problem in comparison to the methods presented in the previous chapter. The basis for this method was originally presented in Reference [58]. Extended results will appear in Reference [59].
3.2 Method Motivation and Overview

Direct collocation [4] and pseudospectral methods [46, 50, 48], while different, share one common characteristic in that the dynamics are collocated at discrete points along the trajectory. This allows the conversion of the continuous optimization problem to a discrete, nonlinear programming problem using approximating polynomials of various forms. Collocation is used to ensure the approximating polynomials are accurate representations of the states’ behavior between the discrete nodes. However, collocation adds additional constraints to the nonlinear programming problem. These constraints directly add to the computational cost of the problem because nonlinear solvers generally make use of the constraint derivatives with respect to the varied parameters. Numerical derivatives computed with finite differencing are slower and less accurate compared to analytical derivatives. In addition, the finite differencing step size must be chosen with care. Derivative accuracy directly influences the speed and accuracy of an optimal solution. Automatic methods for computing derivatives that match analytic accuracy exist[60, 61], but are generally only moderately faster than finite differencing and do not approach the performance of analytic derivatives.

The neural network approximation method removes the need for collocation and numeric or automatic derivative calculation by approximating the dynamics with a neural network over a small, given time period. The trajectory is then built recursively by chaining these segments together. The objective function over the trajectory is computed in the same manner. This allows the dynamics constraint to be met in the offline network training phase, avoiding the collocation requirement when solving the problem. An additional advantage of the neural network approximation is that the objective and constraint derivatives are easily calculated analytically for any feedforward network. Hornik, Stinchombe, and White showed the existence of a neural network with one hidden layer can be a universal approximator given certain conditions [62, 63, 64]. If a smooth hidden layer is used, they show that the network converges to the function’s first derivatives as well. Only the existence of such a network is proven; there is no theory on how many neurons should be used to match a particular function. Thus it is important to check for convergence after training. Alternatively, Basson and Engelbrecht [65] present a
method for explicitly learning the first derivatives along with the target function. This may provide an alternative to simply relying on network convergence.

Note that currently the method is limited to problems with fixed final time, however, it could be applied to free time problems by discretizing along one of the states with a fixed final value and training the network appropriately. Work in inverting a feedforward neural network [66] such that the desired outputs are used to compute the inputs may also facilitate using the method in this situation.

3.3 Previous and Related Research

Some previous research using neural networks in trajectory optimization is now discussed. A large body of work exists on using neural networks to directly solve nonlinear programming problems. In 1957, Dennis [67] proposed the idea of solving linear and quadratic programming problems by implementing them with analog electrical networks. These networks provided very fast solutions. Out of this work lead to the idea of using a neural network as a dynamic system whose equilibrium point is the solution to the linear, quadratic, or nonlinear programming problem. Effati and Baymani [68] propose a method for quadratic programming (sequential quadratic programming is widely used method in solving nonlinear programs) that uses a Hopfield model of a neural network. The network is treated as a dynamic system that converges over time to the optimal solution of the primal and dual problems. Reifman and Feldman [69] discuss a method in which the inverse dynamics are modeled by a feed-forward network in order to convert the nonlinear programming problem to an unconstrained form and solve it via a bisection method. Many others have reported results in this area. However, our method is only tangentially related to this area of research.

Work by Niestroy [70], Seong and Barrow [71], and Peng et al. [72] use a network to approximate various optimal controllers; this is known as “neural dynamic optimization”. Given the system model, the network weights and biases are optimized offline to produce an optimal controller that minimizes an objective function of the states and controls. The trained network is then used online to control the system. The authors are in general agreement that the resulting controller is robust to modeling errors. However, it may be difficult to change the controller without re-
training the network if the dynamics or objective changes. Yeh [73] approximated the strength of concrete given various ingredients and experimental data with a neural network and then used the network in a nonlinear programming problem to optimize the mix. The network was only used to provide an experimental model. Similarly, Inanc et al. [74] use a neural network to approximate the signature and probability detection functions of a UAV for a low-observable trajectory generation framework. Specifically, they used the neural network to approximate tabular data in order to provide a differentiable function for the trajectory optimization. The authors report that the neural network approximation increases the smoothness of the resulting trajectory compared to a B-spline approximation. While the neural network required fewer constraints, the authors do not give any computation time comparison.

3.4 Neural Network Formulation

As before, the basic optimization problem is one of minimizing an objective function with any type of nonlinear constraints. Consider the nonlinear dynamic system given in Equation 2.2 and the nonlinear constraints given in Equation 2.3

\[
\dot{x} = f(x, u) \quad (3.1)
\]
\[
c(x, u) \leq 0 \quad (3.2)
\]

where \(x\) and \(u\) are the state and control input vectors. We seek the control input \(u(t)\) that minimizes a scalar objective function of the form:

\[
J = \int_{t_0}^{t_f} \gamma(x, u) dt \quad (3.3)
\]

In general, any number of the state variables can have their initial values, \(x_0\), or their final values, \(x_{tf}\), specified, and the final time \(t_f\) must be fixed. Because the neural network approximation method is currently limited to problems where the final time is known, \(t_f\) is specified, but the final state vector is free to vary. A receding horizon approach, with horizon time \(T_h\), is used to make the computation feasible in real time and to incorporate new information as conditions change.
The trajectory is discretized into \( n \) equal segments of length \( \tau \) seconds, where \( n\tau = T_h \). The endpoints of these segments are called nodes. Given \( x_0, u_{[0...n]} \), a neural network is then used to recursively approximate succeeding values on the path (Figure 3.1). Using the controls at the endpoints of a segment, a linear interpolation is used to represent the control time history over the length of a segment, as shown in Equation 3.4. The state at the end of a segment is calculated by integrating the equations of motion over the segment (Equation 3.5). Given the initial conditions of the states at the beginning of a segment and the starting and ending controls, the neural network is trained to output of the state equation integration. The training set size is reduced by only training the network over a segment. A consequence of this method is that the linear control interpolation and the length of a segment in time is embedded into the neural network. Therefore, the segment length must be short enough so that the linear control assumption remains a valid approximation.

\[
\begin{align*}
  u(t) &= u_0 + (u_1 - u_0) \frac{t - t_0}{t_1 - t_0} \\
  x(t_0 + \tau) &= \int_{t_0}^{t_0+\tau} f(x(t), u(t)) dt
\end{align*}
\] (3.4) (3.5)

The neural network approximation results in

\[
\begin{align*}
  x_1 &= Y_d(x_0, u_0, u_1) \\
  x_2 &= Y_d(x_1, u_1, u_2) = Y_d(Y_d(x_0, u_0, u_1), u_1, u_2)
\end{align*}
\] (3.6) (3.7)

The states at each node are recursively computed from \( x_0 \) and \( u_{[0...n]} \).

\[
\begin{align*}
  x_{i+1} &= Y_d(x_i, u_i, u_{i+1}) \quad \text{for} \quad i \in [0, 1, \ldots, n-2]
\end{align*}
\] (3.8)
Similarly, to approximate the objective function, the neural network is trained to approximate the value of the objective along a segment. Again, the value of the objective function along such a segment depends only on the initial state and the control history.

\[ J_0 = \int_{t_0}^{t_0+\tau} \gamma(x, u) dt \] (3.9)

Approximated by the neural network:

\[ J_0 = Y_J(x_0, u_0, u_1) \] (3.10)

The objective function over the length of the horizon time can be built up by using the state at the end of a previous segment as the initial condition for the next. Thus, the objective function value depends only on the initial state at the first node and the controls at each node.

\[ J = \sum_{i=0}^{n-2} J_i = \sum_{i=0}^{n-2} Y_J(x_i, u_i, u_{i+1}) \] (3.11)

The optimization problem is reduced from finding all of the states and controls to simply finding the controls. It is no longer a collocation problem because the approximating functions, be they Hermite cubics or \( N^{th} \) degree Lagrange polynomials, have been replaced by the neural network. The ‘defect,’ which is used in the direct collocation and pseudospectral methods to ensure the approximating polynomials accurately represent the true equations of motion, is now accounted for in the neural network training. Multiple objective functions can be combined in a weighted sum. Overall limits on the controls are present in the optimization problem and do not require additional optimization parameters. To apply limits on the state variables, additional optimization parameters are required. It is then advantageous to write the dynamic equations to avoid state limit requirements or to automatically adhere to state limits. Once the problem is parameterized, SNOPT [13] or another suitable nonlinear solver is used to solve the nonlinear programming problem.
3.4.1 Derivative Calculation

By approximating the problem with neural networks, the Jacobians of the system equations, objective, and constraint functions become a straightforward calculation. Assuming the network is well trained, the approximated derivatives will converge to the target function derivatives \([62, 63, 64]\). The current work uses a three layer feed forward network: one input layer, one hidden layer, and one output layer. The transfer functions used are linear, hyperbolic tangent sigmoid, and logarithmic sigmoid. The corresponding functions in the Neural Network Toolbox for MATLAB are \texttt{purelin}, \texttt{tansig}, and \texttt{logsig}. Consider a three-layer feed-forward network with weight matrices \(W_{[i,h,o]}\) and bias vectors \(b_{[i,h,o]}\) corresponding to the inner, hidden, and outer layers, respectively. Layer transfer functions are denoted by \(y(\cdot)\). In a feed-forward network, each node in a particular layer receives the weighted sum of the output from all nodes in the previous layer. Given input vector \(\mathbf{x}\), the equation for network output \(\mathbf{z}\) is

\[
\mathbf{z} = y(W_o y(W_h y(W_i \mathbf{x} + \mathbf{b}_i) + \mathbf{b}_h) + \mathbf{b}_o)
\]  

(3.12)

The hyperbolic tangent sigmoid and its derivative is given by

\[
y(x) = \frac{2}{1 + e^{-2x}} - 1
\]  

(3.13a)

\[
\nabla y(x) = 1 - x^2
\]  

(3.13b)

The logarithmic sigmoid and its derivative is given by

\[
y(x) = \frac{1}{1 + e^{-x}}
\]  

(3.14a)

\[
\nabla y(x) = x(1 - x)
\]  

(3.14b)

The linear transfer function is simply \(y(x) = x\). Using the chain rule, the derivatives of the entire network can be easily computed. For the three layer network, the derivatives are

\[
\nabla \mathbf{z} = (\text{diag}(\nabla y_o) W_o) (\text{diag}(\nabla y_h) W_h) (\text{diag}(\nabla y_i) W_i)
\]  

(3.15)
where $\nabla y_{i,h,o}$ denote the gradient of the transfer function of the input, hidden and output layers, respectively. The \texttt{diag} function creates a diagonal matrix from the vector argument. Note that this is only the derivative for one segment of the overall trajectory. To compute the total derivative, the chain rule must be applied again across all segments.

Recall that the state equations, objective, and constraint functions are recursively calculated. Thus the state and controls at the first node affect every derivative for the following nodes. The objective gradient calculation is expanded to illustrate the procedure. Consider a path with $n$ nodes with $m$ states and $p$ controls at each node. Let the objective function value for the $i$th segment be given by $J_i = f(x_i, u_i, u_{i+1})$ for $i \in [0,1,\ldots,n-2]$, and the states at the $i$th node be given by $x_i = g(x_{i-1}, u_{i-1}, u_i)$ for $i \in [1,2,\ldots,n-1]$. The initial state $x_0$ and control $u_0$ are given and control vectors $u_0, \ldots, u_{n-1}$ are computed by the optimization. Note that $J = J_0 + J_1 + \cdots + J_{n-2}$. Thus, $\nabla J = \nabla J_0 + \nabla J_1 + \cdots + \nabla J_{n-2}$, and $\nabla J \in \mathbb{R}^{m+np}$. At the first node, $J_0 = f(x_0, u_0, u_1)$, and the gradient is

$$\nabla J_0 = \begin{bmatrix} \frac{\partial J_0}{\partial x_0}, \frac{\partial J_0}{\partial u_0}, [0], [0], \ldots, [0] \end{bmatrix} \quad (3.16)$$

Additionally, the state vector at the second node is given by $x_1 = g(x_0, u_0, u_1)$. Since $g(\cdot)$ is a vector-valued function, the Jacobian, $\nabla g \in \mathbb{R}^{m \times m+np}$, is shown below (note $[0] \in \mathbb{R}^m$ and is a column vector).

$$\nabla x_1 = \nabla g(x_0, u_0, u_1) = \begin{bmatrix} \frac{\partial x_1}{\partial x_0}, \frac{\partial x_1}{\partial u_0}, \frac{\partial x_1}{\partial u_1}, [0], [0], \ldots, [0] \end{bmatrix} \quad (3.17)$$

At the second node, $J_1 = f(x_1, u_1, u_2) = f(g(x_0, u_0, u_1), u_0, u_1)$. Thus, by the chain rule

$$\nabla J_1 = \frac{\partial J_1}{\partial x_1} \nabla x_1 + \begin{bmatrix} [0], [0], \frac{\partial J_1}{\partial u_1}, \frac{\partial J_1}{\partial u_2}, [0], [0], \ldots, [0] \end{bmatrix} \quad (3.18)$$

$$= \frac{\partial J_1}{\partial x_1} \nabla x_1 + \begin{bmatrix} [0], \frac{\partial J_1}{\partial u} [0, \ldots, 0] \end{bmatrix}$$
The state vector at the third node is given by

\[ x_2 = g(x_1, u_1, u_2) = g(g(x_0, u_0, u_1), u_1, u_2) \] (3.19)

Therefore, the Jacobian is

\[ \nabla x_2 = \left[ \frac{\partial x_2}{\partial x_1}, \frac{\partial x_2}{\partial x_0} \frac{\partial x_1}{\partial u_0}, \frac{\partial x_2}{\partial u_1}, \frac{\partial x_2}{\partial u_2}, [0], [0], \ldots, [0] \right] \]

\[ = \frac{\partial x_2}{\partial x_1} \nabla x_1 + \left[ [0], [0], \frac{\partial x_2}{\partial u_1}, [0], [0], \ldots, [0] \right] \] (3.20)

The remaining segments are iterated over in similar fashion. Recursively, the derivative computation is given by

\[ \nabla x_0 = [1 \times m \times m \times n_{lim}] \] (3.21a)

\[ \nabla J = \nabla J + \frac{\partial J_i}{\partial x_i} \nabla x_{i-1} + \left[ [0], \frac{\partial J_i}{\partial u_{[0\ldots n-1]}} \right] \] (3.21b)

\[ \nabla x_i = \frac{\partial x_i}{\partial x_{i-1}} \nabla x_{i-1} + \left[ [0], \frac{\partial x_i}{\partial u_{[0\ldots n-1]}} \right] \] (3.21c)

In the final result, the first \( m + p \) columns are not used because these are the derivatives with respect to \( x_0 \) and \( u_0 \) which remain constant.

It is also convenient to extract out the state constraint derivatives as needed during iteration over the segments. The affected rows of the state constraint Jacobian at the \( i^{th} \) segment are given by

\[ \nabla C(k_{sc}) = \nabla x_i(k) \] (3.23)

where \( n_{lim} \) is the number of states subject to constraints. The vector \( k \) contains the indices of the states that are subject to a constraint. At each segment, the rows in the state constraint Jacobian identified in Equation 3.22 are computed.
In words, Equation 3.23 states that the affected rows ($k$) of the states Jacobian for the $i^{th}$ segment are put into the affected rows ($k_{sc}$) of the state constraint Jacobian. For example, if a constraint is placed on the third state of a system, the third row of $\nabla x_i$ would be placed in the $i^{th}$ row of $C$. Note that while the constraints on the state are simple limits, nonlinear constraints are necessary because the states are not part of the varied parameters (only the controls are) and are nonlinear functions of the control input. While no other general constraints were imposed in this problem, they are simple to add and would follow the implementation of the objective function.

### 3.5 Training

Typical neural network training methods are used. The state and control limits are defined and a set of random points within these limits is used as training data. If possible, the training data should use relative values (such as the relative distance from a UAV to a target) to reduce the size of the function space. Given a random initial state and random initial and final controls, the state equations are integrated over the timestep using a linearly interpolated control time history. If the resulting states are within limits, the state vector is saved. If the resulting states are out of bounds, new initial state and initial and final control vectors are generated.

More details on the training procedure are given in the following chapter. The neural network approximation method is applied to UAV surveillance problem described in the following chapter. It is compared with the direct collocation and psuedospectral methods in terms of trajectory results and computation time.
Chapter 4

UAV Path Planning Surveillance Problem

In this chapter, a UAV surveillance problem is described as a non-trivial application of the direct collocation, pseudospectral, and neural network approximation methods. The problem is stated as follows: given one or more UAVs, a fixed, onboard camera whose intrinsic parameters are known, and a ground target whose location is estimated from onboard sensors, maximize surveillance time while observing the performance limits of the aircraft and accounting for external disturbances such as wind or target motion. Simulation results produced in Matlab are presented in this chapter; in the following chapter, flight testing of the algorithm in real time onboard a UAV is discussed.

4.1 Equations of Motion

The problem is approached with a reduced set of equations of motion for the UAV and target. Motion in only two dimensions is assumed, keeping altitude constant. Let $x_u$ be the state vector of an aircraft, $x_t$ be the state vector of a target, and $u$ be the aircraft control vector. For the simplified dynamic model, the states are North and East position of the UAV and target ($x, y, x_t, y_t$), airspeed of the UAV ($V_t$), and heading of the UAV ($\psi$). The controls are longitudinal acceleration...
command \((u_a)\) and bank angle command \((u_\phi)\).

\[
x_u = \begin{bmatrix} x \\ y \\ V_t \\ \psi \end{bmatrix} \quad u = \begin{bmatrix} u_a \\ u_\phi \end{bmatrix} \quad x_t = \begin{bmatrix} x_t \\ y_t \end{bmatrix}
\]

(4.1)

(4.2)

The complete dynamic model is given below for one aircraft and one target. The aircraft equations account for a constant wind speed.

\[
\dot{x} = \dot{x}_{u_1} = V_t \cos(\psi) - V_{\text{wind}N} \\
\dot{y} = \dot{x}_{u_2} = V_t \sin(\psi) - V_{\text{wind}E} \\
\dot{V}_t = \dot{x}_{u_3} = u_a \\
\dot{\psi} = \dot{x}_{u_4} = g \tan(u_\phi)/V_t \\
\dot{x}_t = \dot{x}_{t_1} = V_{\text{tgt}N} \\
\dot{y}_t = \dot{x}_{t_2} = V_{\text{tgt}E}
\]

(4.3)

Constraints on the problem include minimum and maximum airspeed, bank angle limits, and longitudinal acceleration limits.

\[
V_{\text{stall}} < V_t < V_{\text{max}} \\
-\phi_{\text{max}} < u_\phi < \phi_{\text{max}} \\
-a_{\text{max}} < u_a < a_{\text{max}}
\]

(4.4)

4.2 Discretization Scheme

When the continuous problem is transcribed into a nonlinear programming problem, the states and controls at each node are solved for. The vector containing all of these values is called the parameter vector. As discussed previously, a receding horizon approach is used. To apply any of the direct methods, the horizon time is discretized with a number of nodes. The specific discretization differs between the direct collocation/pseudospectral methods and the neural network method.
4.2.1 Direct Collocation and Pseudospectral Parameterization

The direct collocation and pseudospectral methods solve for both the states and controls at each node. Let $p_s$ be the vector of all the states of all $n$ UAVs and $q$ targets at a single node, and let $p_c$ be the vector of all $n$ control vectors (for $n$ UAVs) at a single node (targets do not have control inputs):

$$p_s = \begin{bmatrix} x_u^{[0]} \\ x_u^{[1]} \\ \vdots \\ x_u^{[n-2]} \\ x_u^{[n-1]} \\ x_t^{[0]} \\ x_t^{[1]} \\ \vdots \\ x_t^{[q-2]} \\ x_t^{[q-1]} \end{bmatrix} \quad p_c = \begin{bmatrix} u^{[0]} \\ u^{[1]} \\ \vdots \\ u^{[n-2]} \\ u^{[n-1]} \end{bmatrix}$$

Now the problem matrix $P_m$ is assembled for $N$ node points:

$$P_m = \begin{bmatrix} p_{s0} & p_{s1} & \cdots & p_{s_{N-2}} & p_{s_{N-1}} \\ p_{c0} & p_{c1} & \cdots & p_{c_{N-2}} & p_{c_{N-2}} \end{bmatrix}$$

The states are organized in the matrix such that each row contains the values of a single (particular) state at each node point; each column represents a node. The nonlinear solver expects a single column vector. The problem matrix is reshaped column-wise and appended to the state vector column.

4.2.2 Neural Network Approximation Parameterization

The neural network approximation does not solve for the states at each node, only the controls. Therefore, only the controls section of the above parameter vector is used in the nonlinear program.
4.3 Constraints

The complete list of problem constraints specific to the above formulation is given in Equation 4.7. The defects $\Delta$ are constrained to zero. Note that the initial state and controls $(x_0, u_0)$ are the state and controls of the UAV and target at the time the optimization is started and remain constant for the optimization. Through the use of constraints on the bank angle command $(u_\phi)$, turn rate can be limited. Similarly, longitudinal acceleration is limited by the constraint on $u_a$.

\begin{align*}
\Delta &= 0 \\
x_0 &= x_0 \\
u_0 &= u_0 \\
V_{min} &\leq V_{ti} \leq V_{max} \\
u_{amin} &\leq u_{ai} \leq u_{amax} \\
u_{amin} &\leq u_{ai} \leq u_{amin} \\
 u_{amin} &\leq u_{ai} \leq u_{amin}
\end{align*}

(4.7)

When the path planner is started, an initial guess for the path over the horizon is required. Since the direct collocation and pseudospectral methods require no specific initial guess to converge, the current heading and speed of the UAV is used to extrapolate a straight path out to the end of the horizon time.

The neural network method is slightly different. Because only the controls are solved for in the nonlinear program, any state constraints must be added separately. The airspeed limits are the only state constraints for this particular problem. State constraints take the form of a constrained nonlinear function of the controls (the approximated dynamics).

4.4 Objective Function

The desired behavior for the UAV is to maximize sensor coverage of the target. The objective function that drives this behavior is a weighted sum of four separate objectives.

\begin{equation}
J = \int_{t_0}^{t_f} \left[ w_1 u_a^2 + w_2 u_\phi^2 + w_3 ((x - x_t)^2 + (y - y_t)^2) + w_4 J_{tiv} \right]
\end{equation}

(4.8)
The first two terms penalize control effort (longitudinal acceleration and bank angle), and the third term weights the square of the distance to the target. The fourth term, \( J_{\text{tiv}} \), is a “target-in-view” cost function which has its minimum value when the target is at the center of the image plane of the onboard camera and reaches its maximum value when the target is out of the camera frame. This function is described in more detail below. Note that the third term (distance to target) is required so that in the case that the target is too far away to be viewed by the aircraft, it will attempt to get closer to the target. In the DCLNP solution the cost function is calculated by performing a numerical integration of Equation 4.8 along the state and control trajectory defined by the nodes. The objective function is readily expanded to include cost associated with multiple aircraft by including additional control effort and distance to target terms. The target-in-view cost can be also expanded to include multiple aircraft by selectively using only a particular aircraft’s cost depending on certain criteria or by computing a weighted average of all aircraft costs. This could be tailored any number of ways to produce various UAV interaction behavior. The method used for multiple UAVs in this work is discussed in more detail after the target-in-view equations.

The target-in-view cost is calculated by transforming the target’s world coordinates into image plane coordinates (pixel location, \( q = [q_x \ q_y \ 1]^T \)) using a perspective plane transformation called a homography [75] (see Figure 4.1). This transformation takes into account the attitude of the UAV, its position relative to the target, the focal length of the camera, the sensor size of the camera, and the orientation of the camera as it is mounted in the airframe. The homography matrix calculation is given in Equation 4.9

\[
H = KC_mT = KC_m \begin{bmatrix} R & -Rp \\ 0 & 1 \end{bmatrix}
\]  

(4.9)

where \( K \) is the camera intrinsic properties matrix, \( C_m \) is the rotation matrix between camera axes and aircraft axes, \( R \) is the direction cosine matrix of the UAV’s Euler angles, and \( p = p_{\text{uav}} - p_{\text{tgt}} \) is the position of the UAV relative to the target in world coordinates. The pixel coordinates are then calculated from the world
coordinates in Equation 4.10.

\[
\bar{q} = H \begin{bmatrix} p^T & 1 \end{bmatrix}^T
\]  \hspace{1cm} (4.10)

The z-element of \( \bar{q} \) must equal one:

\[
q = \left(1/\bar{q}_z\right) \begin{bmatrix} \bar{q}_x & \bar{q}_y \end{bmatrix}^T
\]  \hspace{1cm} (4.11)

This calculation can be simplified by removing the Z-coordinate column from \( R \) in the top left corner of \( T \) because the model uses 2-dimensional motion at a fixed altitude. In this formulation only the roll and yaw attitudes of the aircraft are accounted for, further simplifying the calculation above. Pitch attitude is assumed to be zero, but the effect of pitch could readily be added if more detailed equations of motion are used. The camera orientation relative to the airframe is also assumed to be constant. However, different camera installation orientations or a gimbaled camera can be accounted for by a time dependent \( C_m \) matrix. The target-in-view cost is zero if the target is in the center of the image and varies parabolically to the edge of the image where its reaches its maximum value of one. Outside of the

Figure 4.1. Axes used in the homography
image bounds, the cost function is held constant at one. The motion of the target in sensor coordinates is very nonlinear due to both the coordinate and perspective transformations. The “min-max” target-in-view cost for a single aircraft and a single target is given by Equation 4.12.

$$J_{tiv} = \min \left( \max \left( \frac{4q_x^2}{I_{x_{max}}^2}, \frac{4q_y^2}{I_{y_{max}}^2} \right), 1.0 \right)$$ (4.12)

where $I_{x_{max}}$ and $I_{y_{max}}$ are the size in pixels of the image plane. If the camera intrinsic matrix $K$ is normalized by image size, Equation 4.12 can be simplified:

$$J_{tiv} = \min \left( \max \left( q_x^2, q_y^2 \right), 1 \right)$$ (4.13)

An alternative formulation using the dot product of the pixel location is given by

$$J_{tiv} = \min (q_x^2 + q_y^2, 1) = \min (q \cdot q, 1)$$ (4.14)

This “quadratic formulation” formulation results in a smooth variation of the target-in-view objective function without the discontinuities on the image diagonals. The min-max formulation was initially used as it provides full image coverage. All flight test results use this formulation. The quadratic formulation, due to being a smooth function, allows the optimization to converge much more quickly. The comparisons between the methods use this formulation. A side effect of using a neural network to approximate the objective function is that discontinuous functions are smoothed. This effect was noted when the “min-max” function was approximated with a network, and is discussed by Inanc [74]. Figure 4.2 shows a visualization of the target-in-view cost: the quadratic cost is Figure 4.2(a) and the mix-max cost is Figure 4.2(b).

For multiple aircraft performing surveillance on a single target, only the minimum $J_{tiv}$ of all aircraft is used. Thus, the cost is based on the aircraft that has the best view of the target at a particular point in time and it is assumed that there is no particular benefit gained from multiple aircraft simultaneously observing the target. Therefore, when the UAVs observe the target non-simultaneously, the target-in-view objective function is lesser than if the target is observed simultaneously. However, if it is determined that there is a benefit to have multiple
Figure 4.2. Target-in-view cost functions

simultaneous observations, a weighted sum of all target-in-view-costs would produce this behavior.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Effort</td>
<td>Control input squared</td>
<td>Adjust magnitude of control response</td>
</tr>
<tr>
<td>Distance-to-target</td>
<td>Distance squared from the UAV to target</td>
<td>Drive the UAV towards the target</td>
</tr>
<tr>
<td>Target-in-view</td>
<td>Target location transformed to pixel coordinates; maximum value when pixel coordinates exceed the image boundaries, minimum value at the center of the image</td>
<td>Shape the path so that views of the target are captured</td>
</tr>
</tbody>
</table>

Table 4.1. Objective Function Listing

As a final overview of the entire process, the problem is parameterized by the method in Section 4.2. The problem is then solved by searching for the parameter vector (transformed form of Equation 4.6) that minimizes the objective function (Equation 4.12) subject to the constraints (Equations 2.9 & 4.7). The various objective functions used are summarized in Table 4.1.
4.4.1 Integrating External World Data

The initial work for the path planner algorithm resulted from fairly simple requirements: create a path such that the time spent with the sensor on the target is maximized. However, by expanding the requirements, the planner becomes more useful. One such way to increase capability of the planner is to include external information of the surrounding area (e.g. roads). With the use of known roads, the planner becomes less reliant on observations of target motion to be able to predict future positions. For example, if the path planner has road data for the area it is observing, it may be reasonable to assume a vehicle will follow the road it is on, and therefore, the path planner already knows the path the vehicle will most likely take. With this information, the path planner has access to a more complete model of where a vehicle might go. As shown in Figure 4.3, estimates of target speed, $V$, and position, $s$, along the road are used to make a prediction. Since the vehicle’s motion is 1-dimensional along a road, the equation of motion is simply

$$\dot{s} = V$$  \hspace{1cm} (4.15)

The vehicle’s location is given by

$$[x_t, y_t] = f(s)$$  \hspace{1cm} (4.16)

The formulation of the trajectory optimization allows the use of any kind of predictive target motion model without modification to the underlying problem parameterization. Simulation results using this feature are given in Section 4.6.
4.4.2 Perspective Driven Surveillance

The trajectory optimization is initially set up to view the target from above. This behavior results from the distance-to-target function causing the UAV to fly directly over the target. The target-in-view function is not biased toward overhead views. However, a simple modification will allow the optimization to capture views of the target from a specific direction. The observation may be made more useful if the UAV can be directed to observe from a particular view point, for example the north side of a building or a profile shot of a vehicle. Instead of the path planner being driven to get any observation of the target, it is driven to get a particular view.

The behavior can be achieved by biasing the reference point used in the distance-to-target cost. This change essentially moves the target’s location with respect to the UAV. As UAV minimizes the distance to this pseudo-target, the target-in-view objective is minimized if the UAV banks to point the camera at the target. Let azimuth $\gamma$ and elevation $\lambda$ describe the unit vector $\hat{o}$ that points toward the location from which the target is to be observed. Using the altitude of the UAV $h$ and $\hat{o}$, a bias $\delta p$ is computed and added to the target location $p$ to produce the new target reference point $p_r$. When used in the optimization, the biased target reference point tends to align the onboard sensor line of sight $\hat{s}$ with the desired viewing direction $\hat{o}$.

$$\delta p = \begin{bmatrix} \cos(\gamma) & 0 \\ \sin(\gamma) & 0 \end{bmatrix} \begin{bmatrix} h \tan(\pi - \lambda) \\ \end{bmatrix}$$

(4.17)

Setting $\lambda = \pi$ results in $\delta p = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$, producing observations directly over the top of the target. Figure 4.4 shows an illustration.

4.5 Neural Network Specific Details

This section discusses some details specific to using neural network method to solve the UAV surveillance problem. The problem as discussed above has four major components: the state equations, the control effort objective, the distance-to-target objective, and the target-in-view objective. The control effort objective function
Figure 4.4. Illustration of the perspective driven objective function

is kept as an analytical expression because it is very simple - expressions for its
derivative are easily written. The remaining three components are approximated
with feed-forward neural networks. However, the exact structure of the problem is
not used. Instead of separate states for the UAV and target locations, the relative
distance from the UAV to the target is used to reduce the dimension of the function
approximated by the neural network. This also reduces the required range of the
positional states since for relative distances over a certain magnitude the only
option is to fly toward the target. For this specific problem, relative distance to
the target was limited to ±1000 ft. Limits were also applied to airspeed, bank
angle, and longitudinal acceleration corresponding to aircraft performance limits.

The input to the dynamics network is \([V_t, \psi, u_{a0}, u_{\phi0}, u_{a1}, u_{\phi1}]^T\). Its output is
\([\delta x, \delta y, V_t, \psi]^T\) where \(\delta x\) and \(\delta y\) are change in position from the point corresponding
to the input data. The input for the distance-to-target and target-in-view functions
is \([x_{rel}, y_{rel}, V_t, \psi, u_{a0}, u_{\phi0}, u_{a1}, u_{\phi1}]^T\). Output is a scalar value that approximates
the objectives. The networks used for the UAV surveillance problem are shown
schematically in Figure 4.5.
4.5.1 Network Structure

The neural networks used for all three functions are three layer, feed forward, fully connected networks with 15 neurons in the input and hidden layers, and the appropriate number of output neurons in the output layer. The dynamics and distance-to-target networks used hyperbolic tangent sigmoid transfer functions in the input and hidden layers with a linear output layer. Based on trial-and-error, a transfer function with a logarithmic sigmoid output layer is found to produce the most accurate approximation of the target-in-view function. Figure 4.6 shows the three transfer functions used for the UAV surveillance problem.

4.5.2 Neural Network Training

The Neural Network Toolbox [76] for MATLAB is used for training. The toolbox is not used for simulation because the standard transfer function code is slow (in version 2007a) due to boilerplate code used for backwards compatibility among other things. The transfer functions shown in Figure 4.6 are set up in new files that only include the function and its derivative. This is very important when doing relative performance comparisons – if the standard implementation is used, results will be biased to other methods due to the unnecessary boilerplate code. This modification does not unfairly bias the results toward the neural network method because a reasonable implementation of these transfer functions only includes the function itself and its derivative. The new files are generally only four lines of code in MATLAB, compared to around 100 lines for the toolbox implementation.
To train the network, a set of random input and output vectors are generated on the interval corresponding to the input limits. The training method used in backpropagation as implemented in the MATLAB function `train`. During training, only 70% of the input vectors are used. Of the remaining 30%, 15% are used to stop training early to prevent overfitting the training vectors, and the other half is used to measure network generalization to new inputs. Training performance is discussed in Section 4.8. One training step is called an epoch. During an epoch, all of the training vectors (70% of the total generated) are presented to the network. The output from all the vectors is computed, and the weights and biases are adjusted to minimize the difference between the true output and the network output.
### 4.6 Simulation Results

The following results are generated only with the direct collocation method in order to explore the details of the UAV surveillance problem. Comparisons with the pseudospectral and neural network approximation methods are presented in a following section. In addition, specific computational performance results are presented. The limits shown in Table 4.2 were used in the following simulation and flight test results. They are based on the estimated performance limitations of the Sig Kadet Senior. As a safety margin, the bank angle limit is lower than that which the UAV is capable.

The path planning algorithm was implemented initially in MATLAB using the built-in `fmincon` function. However, `fmincon` is found to be less consistent in its results and slower compared to the SNOPT package [13]. Therefore, SNOPT is used in all results. The SNOPT routines are available for FORTRAN, C, C++, and MATLAB.

While the MATLAB environment is well suited for development and prototyping, it is less than ideal for onboard the aircraft. The MATLAB implementation of the path planner is used for initial simulation and testing. It also serves as a model and validation tool for the C++ implementation of the path planner for use onboard the aircraft. It was highly useful in this respect. The following simulation results were created using the SNOPT optimization routines.

The first case, shown in Figure 4.7, is a single UAV with a stationary target. The resulting trajectory is a cloverleaf pattern. The bold parts of the ground track indicate that the target is in the view of the onboard camera. Initially, the UAV accelerates to maximum speed to close the distance to the target, and then drops to minimum speed to surveil the target. The cloverleaf pattern develops because

#### Table 4.2. UAV performance constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stall Speed</td>
<td>$V_{\text{min}} = 22\text{kts}$</td>
</tr>
<tr>
<td>Maximum Speed</td>
<td>$V_{\text{max}} = 50\text{kts}$</td>
</tr>
<tr>
<td>Maximum Longitudinal Acceleration</td>
<td>$u_{\text{max}} = 10\text{ft/s}^2$</td>
</tr>
<tr>
<td></td>
<td>$u_{\text{min}} = -10\text{ft/s}^2$</td>
</tr>
<tr>
<td>Maximum Bank Angle</td>
<td>$\phi = \pm 30^\circ$</td>
</tr>
</tbody>
</table>
of the bottom pointing camera. The UAV must fly directly over the target to allow the camera to capture a view and then double back to capture another view.

![Graph Image](image-url)

(a) Ground track

(b) Observation time history (black bars indicate target is in view of the camera)

Figure 4.7. MATLAB simulation of a single UAV (dash-dot) and a stationary target (circled ‘x’)

A steady wind can afford advantages to the surveillance problem. In general, a steady wind is found to increase the coverage percentage as wind speed increases. Figure 4.8(c) shows the resulting path a with a 10 ft/sec wind from the North. A target moving North at 10 ft/sec in calm winds is exactly equivalent. Figure 4.8(a) shows the resulting path in the world frame for a moving target. Figure 4.8(b) shows the same path transformed to the target frame. The path is identical to Figure 4.8(c). Therefore, while wind is accounted for in the dynamics, for the remainder of the results only moving targets will be presented because a steady wind and a steady moving target are equivalent when viewed in the target frame.

A trivial case where the target is moving at a speed able to be matched by the UAV is shown in Figure 4.9 for completeness. The target is moving East at 65 ft/sec. The UAV accelerates to cover the distance to the target, then matches
speed with and flies directly over the target.

A more interesting case occurs when the target is moving slower than the stall speed of the UAV. In this situation, the UAV flies at its minimum safe speed and circles back over the target. In general, only upwind turns are made in order to minimize deviation from the target location during the turn. Figure 4.10 shows
the ground track and observation time history for this situation.

Figure 4.10. MATLAB simulation of a single UAV (dash-dot) and a slow moving target at 10 ft/s (solid red starting at circled ‘x’)

Target surveillance with two UAVs allows for nearly continuous coverage. Two identical UAVs are equipped with identical cameras. To avoid the question of collision avoidance, one UAV is simulated at 300 ft altitude and the other is simulated at 400 ft (anticollision methods could be included in the problem formulation at computational cost). Because the target is stationary, the cloverleaf pattern of the previous calm wind case develops for both UAVs. However, because the optimization considers both aircraft together, the resulting trajectory staggers the passes over the target so that for a majority of the time, only one UAV has the target in view. This behavior is a result of the way the target-in-view objective function is set up for multiple aircraft. No specific priority is given to one UAV over another, and there is no benefit assumed to having both aircraft view the target simultaneously. Figure 4.11 show the paths of the two UAVs, and Figure 4.12 shows the distance of each UAV to the target. The bold sections of the lines in Figure 4.12
indicate when the target was in view of that particular UAV. Note that the target is alternately covered between aircraft as they pass overhead. With two UAVs, the target sensor coverage percentage approaches 100% while the UAVs are on location.

![Figure 4.11. MATLAB simulation of a UAV pair (dash-dot and grey lines) and a stationary target (circled ‘x’ at (500,1000))](image)

**Figure 4.11.** MATLAB simulation of a UAV pair (dash-dot and grey lines) and a stationary target (circled ‘x’ at (500,1000))

![Figure 4.12. Distance from UAV to target where bold lines indicate the target is imaged by the UAV. The offset in distance is due to a difference in altitude of the UAVs](image)

**Figure 4.12.** Distance from UAV to target where bold lines indicate the target is imaged by the UAV. The offset in distance is due to a difference in altitude of the UAVs

The next results are a moving target along a road. Recall from Section 4.4.1 that the optimization makes use of known road data to predict future target motion instead of assuming straight line target motion. The following simulation results is chosen to show an advantage road data gives to the path planner. The test road makes a $90^\circ$ turn with a 50 ft radius every 300 feet and is simulated for 120
seconds. The path generated is shown below in Figure 4.13(a). The solid red line represents the target’s path along the road, and the black dash-dot line is the UAV’s path. In this case, the speed of the target was within the speed range of the UAV. The target is in view of the UAV’s camera 88% of the time, as seen in Figure 4.13(b). An interesting situation occurs when the vehicle is capable of traveling faster than the UAV, but the vehicle must use a curvy road. The path planner accounts for this by following the target in a ‘looser’ fashion; making up for the speed difference by taking advantage of the curves in the road. Figure 4.14(a) illustrates this behavior. For this run, the target speed is set to 100 ft/sec while the maximum UAV speed is 87 ft/sec. The path planner creates a path that cuts across the curves in the road to make up for the speed difference. Even though the target is moving faster than the UAV, the camera is viewing the target 100% of the time because the UAV banks to keep it in view. With a gimbaled camera, the banking would not be necessary.

Several perspective driven case are now shown. In the first case, the target is stationary, and the UAV is to capture view of the north side from 400 ft altitude at an elevation angle of 45°. Figure 4.15 shows the ground track and observation time history. In this case, the downward pointing camera is advantageous because it points outward in a turn. The UAV is in a constant bank turn at its maximum

Figure 4.13. Path planner simulation with road data, target slower than max UAV speed
bank angle of 30°. Therefore, the target is not in the center of the image, but it is visible.

For the second perspective driven case, the target is moving east at 10 ft/s. The UAV is set to capture observations of the target’s North side from an elevation angle of 45°. Figure 4.16 shows the ground track and target observation time history of the case. Note that compared to the previous moving target case, observation time has fallen sharply, however, all observations are made from a specific viewpoint which, given the situation, can potentially be more valuable than overhead views. The sensor coverage percentage dropoff is certainly expected when using a fixed, downward pointing camera mounted in a UAV.

The perspective driven cases do require more weights tuning than the standard cases. There is a fine balance between too much distance-to-target weight compared to target-in-view weight. If the target-in-view weight is too high, the resulting path favors flying directly over the aircraft because this maximizes the time the target is in view (driven by the downward pointing camera couple to aircraft dynamics). Conversely, a high distance-to-target weight simply causes the resulting path to circle the biased target location. Starting with a larger target-in-view cost and then iteratively decreasing it until the resulting path approached the biased target reference point while still aiming the camera at the target when possible is found to work well. Generally, the control weights only need to be
A consequence of using a weighted sum of distance and sensor pixel location is that the two objectives compete against each other and in certain situations, can lead to possibly optimal but useless solutions. Such a situation occurs when the UAV is directed to capture views of a moving target from the point of view ahead of where the target is moving to, the optimization produces a figure-8 shaped path that simply flies in front of the target and does not produce any target surveillance. Because the UAV is able to keep so close to the target, the overall cost is minimized without capturing any views of the target. Distance-to-target and controls costs would increase sharply in order to capture views of the target. Increasing target-in-view weight does not seem to alleviate the situation. This may be related to the fact that no gradient information is given if the target is out of view of the camera.

This formulation has another weakness in that the views of the target aren’t
Figure 4.16. MATLAB simulation of a single UAV (dash-dot) and a slow moving target target capturing a specific view (solid red starting at circled ‘x’)

actually along a 45° angle, but are closer to 60°. This is due to the fact that none of the objective functions explicitly derive a cost number from the desired alignment of the sensor line of sight and the desired observation line of sight. Adding an additional objective that is minimum only when the sensor is pointing at the target from the desired line of sight may help here.

Because the target-in-view function is held constant when the target is not visible to the camera, no gradient information is available to the optimization. A revised target-in-view objective with a small negative slope outside of the image plane boundaries may perform better in perspective driven cases.

4.7 Convergence Results

Convergence of the above solutions is now discussed. Two different notions of convergence are investigated: 1) convergence of the collocation method with increasing
number of nodes distributed over a constant horizon length and 2) receding horizon convergence with increasing horizon length using constant node spacing in time. For each case, the target observation ratio is computed. This ratio is the amount of time the target is viewed by the onboard camera divided by the total maneuver time. However, some care needs to be taken when calculating this number. As the target falls in and out of view, this ratio will vary cyclically making the final result dependent on when the maneuver ends. Therefore, running averages of different window lengths are used. For these results, windows of 30, 60, and 120 seconds are used. These windows generally cover several cycles of the behavior shown by the path planner when observing targets in various situations. The final number is an average of these three windowed averages. Figure 4.17 shows the time history of the observation ratio for a single stationary target along with the windowed averages.

![Figure 4.17. Observation ratio time history with windowed averages for a stationary target](image)

4.7.1 Discretization Convergence

In order to say the result of the collocation methods are optimal, their results computed with an increasing number of nodes over a constant horizon length are compared. Discretization convergence test results are presented here for target
speeds of 0, 10, 20, 25, 27, and 35 \( ft/sec \). The uneven choice in speeds is used because they generally represent uniform increases in the target observation time ratio. Two horizon lengths are given, 16 and 30 seconds. For the 16 second horizon (Table 4.3), the target coverage percentage increases by 5% to 11% as the number of nodes are increased, depending on target speed. The percentage increase also generally decreases as the number of nodes are doubled, which indicates the collocation method does converge to a local minimum. However, there are some anomalies. For \( V_{targ} = 20 \text{ ft/s} \), the observation ratio falls by 3% as the number of nodes are doubled to 36 from 18. This can be explained by the difference in the transient starting path between the two cases. Running both cases for a longer simulation time reduces this difference. Such is the case for \( V_{targ} = 27 \text{ ft/s} \). When \( V_{targ} = 35 \text{ ft/s} \), the UAV is able to surveil the target 100% of the time by slowly rolling back and forth. When 9 nodes are used, the integration is not accurate enough to produce this behavior. However, for 18 nodes and above, this behavior is seen. Note that the observation ratio is not 100%, but would be if the maneuver is allowed to continue indefinitely as the behavior requires around 120 seconds (in all cases) to develop. With longer horizon length, the behavior develops much more quickly.

The 30-second horizon case behaves similarly. However, for \( V_{targ} = 25 \text{ ft/s} \) the observation fraction decreases as the number of nodes are increased. This appears to be related to how the numerical integration is computed. Ten intermediate segments are used in the quadrature computation, regardless of how small node spacing is. This may contribute to a spike in roll angle which the observation ratio computation does not see, but the optimization does. For the case of \( V_{targ} = \text{Table 4.3. Target coverage convergence using a 16 second horizon (mean of windowed means)\n\begin{tabular}{|c|c|c|c|c|}
\hline
\text{Number of Nodes} & 9 & 18 & 36 & 72 \\
\hline
\text{Coverage \%} & 40.4\% & 42.5\% & 43.2\% & 43.1\% \\
\hline
\text{Coverage \%} & 45.2\% & 47.4\% & 49.0\% & 50.3\% \\
\hline
\text{Coverage \%} & 53.9\% & 57.5\% & 55.6\% & 60.0\% \\
\hline
\text{Coverage \%} & 66.0\% & 72.2\% & 70.8\% & 72.3\% \\
\hline
\text{Coverage \%} & 73.6\% & 73.6\% & 77.6\% & 77.5\% \\
\hline
\text{Coverage \%} & 89.9\% & 94.3\% & 95.7\% & 96.7\% \\
\hline
\end{tabular}\n
Table 4.4. Target coverage convergence using a 30 second horizon (mean of windowed means)

<table>
<thead>
<tr>
<th>$V_{targ}(ft/s)$</th>
<th>Number of Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>41.8%</td>
</tr>
<tr>
<td>10</td>
<td>45.8%</td>
</tr>
<tr>
<td>20</td>
<td>52.7%</td>
</tr>
<tr>
<td>25</td>
<td>68.6%</td>
</tr>
<tr>
<td>27</td>
<td>68.7%</td>
</tr>
<tr>
<td>35</td>
<td>99.2%</td>
</tr>
</tbody>
</table>

27 ft/s, observation performance is worse than the 16 second horizon. This may be related to cycle time of the resulting path.

4.7.2 Receding Horizon Convergence

By comparing paths computed with different horizon lengths, the effects of horizon length on the optimality of the resulting path can be discussed. Three different node spacings are used: 4, 2, and 1 seconds. One characteristic of longer horizon times is that they require more time to reach a cyclic condition. Four horizon lengths are tested: 16, 32, 64, and 128 seconds.

For the one-second node spacing, it is found that a shorter horizon generally produces increased observation time. For stationary and slow moving targets, the effect is not pronounced. When target speed is between 60% and 80% of UAV minimum speed, observation time is reduced as horizon time increases. Table 4.5 summarizes the simulation results.

Table 4.5. Target coverage convergence for increasing horizon length using 1 sec node spacing (mean of windowed means)
For the two-second node spacing, similar behavior is observed.

<table>
<thead>
<tr>
<th>$V_{\text{targ}} (\text{ft/s})$</th>
<th>Horizon Length (sec)</th>
<th>Coverage %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>0</td>
<td>40.4%</td>
<td>41.0%</td>
</tr>
<tr>
<td>10</td>
<td>45.2%</td>
<td>45.3%</td>
</tr>
<tr>
<td>20</td>
<td>53.9%</td>
<td>51.6%</td>
</tr>
<tr>
<td>25</td>
<td>66.0%</td>
<td>69.3%</td>
</tr>
<tr>
<td>27</td>
<td>73.6%</td>
<td>64.1%</td>
</tr>
<tr>
<td>35</td>
<td>89.9%</td>
<td>99.2%</td>
</tr>
</tbody>
</table>

Table 4.6. Target coverage convergence for increasing horizon length using 2 sec node spacing (mean of windowed means)

A node spacing of four seconds is shown in Table 4.7. At this node spacing, the cubic Hermite approximation begins to break down. The polynomial approximation is no longer able to accurately represent the dynamics of the UAV over this length of time. However, the trend of increasing horizon time corresponding to increased coverage percentage is clearly visible, though the percentage itself is not as high compared to smaller node spacings. This reversal from the previous cases is likely tied to the approximation accuracy reduction. While over short time periods the path planner is not able to produce high observation time ratios, a longer horizon may alleviate the approximation accuracy problem by possibly time averaging the error.

<table>
<thead>
<tr>
<th>$V_{\text{targ}} (\text{ft/s})$</th>
<th>Horizon Length (sec)</th>
<th>Coverage %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>0</td>
<td>35.5%</td>
<td>35.5%</td>
</tr>
<tr>
<td>10</td>
<td>26.6%</td>
<td>32.0%</td>
</tr>
<tr>
<td>20</td>
<td>38.3%</td>
<td>38.6%</td>
</tr>
<tr>
<td>25</td>
<td>48.6%</td>
<td>58.9%</td>
</tr>
<tr>
<td>27</td>
<td>46.2%</td>
<td>55.7%</td>
</tr>
<tr>
<td>35</td>
<td>82.6%</td>
<td>99.3%</td>
</tr>
</tbody>
</table>

Table 4.7. Target coverage convergence for increasing horizon length using 4 sec node spacing (mean of windowed means)

These results suggest that a shorter horizon length provides increased observation time if the target is mobile but slower than the UAV and the segment time length is sufficiently small for acceptable approximation accuracy. The reason a
longer horizon reduces performance in this case may be that the chances of the optimization stopping in a less optimal local minimum are increased with a longer horizon time. In any case, the above tests provide evidence that shorter horizon lengths can be used without loss in observation time. This is beneficial for real time performance. In addition, it may be useful to vary the horizon length based on target behavior. If a long segment time is required, it is beneficial to use a longer horizon time to make up for increased approximation error.

4.8 Comparative Results

In this section, results are presented comparing the direct collocation, pseudospectral, and neural network approximation methods.

4.8.1 Training

The dynamics network generally only required 5000 training samples to accurately represent the true dynamics. The distance-to-target objective required 10000. The target-in-view function is the most complex of the three since it involves a coordinate rotation and perspective transform. While decent performance was obtained with 15000 input vectors, 40000 input vectors greatly increased performance in terms of the solution speed of the optimization solver. Figure 4.18 shows the correspondence between scaled true output and scaled network output of each network after training is complete. Note that only 70% of the total training set was used for training. The remaining data is split evenly between validation and testing. The validation data is used to stop network training early if the approximation error increases. This ensures the network does not overfit the training data (the network is highly accurate at the training points but has very poor accuracy on points it has not be trained on). Test data is used to measure network generalization, or how well the network performs on data it has not seen in training. Figure 4.18 shows the complete, scaled training set. The network outputs are plotted against the true outputs. Both the dynamics and distance-to-target networks’ plots are centered on the line $y = x$, which shows these networks approximate their functions very accurately. A large majority of the target-in-view output points are equal to one,
since most of the time, the target is not in view of the camera. The target-in-view function is the most complex therefore its regression plot is somewhat fuzzy. However, as will be shown in the results, the network performed on par with the true target-in-view function.

4.8.2 Results

The results given here are generated in MATLAB on a circa 2007 laptop with 2 GB RAM and an Intel T7500 processor (2.2 GHz). One specific optimization was used to speed up the neural network method. The transfer functions in the Neural Network Toolbox (\texttt{purelin}, \texttt{tansig}, \texttt{logsig}) are burdened with unnecessary boilerplate code that significantly slows down execution time but is necessary only when used with the MATLAB command \texttt{sim}. By putting only the relevant code in a separate function, network computation time is greatly reduced. The three
optimization methods are otherwise coded in similar fashion to minimize differences in performance due to implementation. Based on past experience [77], one can expect a minimum of 5x-10x increase in computation speed by porting to C or some other compiled language.

4.8.2.1 Qualitative Comparison with other methods

To evaluate the usefulness of the neural network approximation method, it is compared to the DCNLP and Chebyshev pseudospectral methods. Results using derivatives obtained with analytic, complex step, and automatic differentiation methods are presented for the DCNLP method. The pseudospectral method results are presented with automatic and complex-step derivatives. The Chebyshev pseudospectral method is used; Yakimenko, Xu, Basset [27] and Huntington, Benson, and Rao [45] note that the various pseudospectral methods have comparable computational performance for similar accuracy. Derivatives using finite differences are not examined. The INTLAB [60] automatic differentiation toolbox is used to automatically derive closed form calculations of the gradients. INTLAB is open-source and competitive with other commercial automatic differentiation toolboxes. The complex-step differentiation method used takes advantage of sparsity to reduce the required number of function evaluations to compute the Jacobian [78]. A good overview on the complex-step method is given by Matrins [61]. The method used here is a custom solution, however a complex step differentiation toolbox by Shampine [79] is available for MATLAB.

By making the comparison between derivative calculation methods, we aim to provide a comparison of execution speed (analytic is fastest) versus complexity of problem setup (complex-step and automatic differentiation are very easy to set up). The optimization results are essentially the same because all three methods compute derivatives generally to the same precision. The main performance advantage of the neural network approximation stems from the removal of the ‘defect’ constraints. Consequently, only the controls at each node need to be optimized. Recall that the direct collocation and pseudospectral methods must minimize the difference between the approximate and true state derivatives. However, the performance of the neural network method depends entirely on the training accuracy. Therefore, it is useful to use one of the other methods as validation (using an
automatic differentiation method for ease of setup). The framework created in MATLAB for this paper uses common problem definitions making it fairly easy to run the same problem on the different methods.

4.8.2.2 Stationary Target

The simplest case is the observation of a stationary target. Figure 4.19 shows a comparison of the performance of the DCNLP and neural network method using a 30 second horizon simulated for 100 sec. The solid parts of the path indicate the target is in view of the UAV camera. Each symbol marks 5 seconds along the path. The target is the red symbol. Interestingly, each figure is different. However, the difference is trivial and is caused by the direction in which the UAV turns after it

(a) Neural network method: 42% coverage time, 0.8 s mean path generation time

(b) DCNLP method: 41% coverage time, 3.0 s mean path generation time

(c) Pseudospectral method: 42% coverage time, 8.0 s mean path generation time

Figure 4.19. Stationary target observation
passes over the target. Each pattern is equivalent. This equivalence can be seen by imagining cloning one of the “lobes” of the cloverleaf pattern and rotating it around the target. Any of the three patterns above can be generated. All methods result in similar observation time. The controls from the above simulation are shown in Figure 4.20. Because there are no roll inertia effects in the model, large bank angle controls sometimes result, as seen in the first few seconds of the neural network simulation. This can be corrected by introducing an extra state in the model, however, only the position and speed of the UAV is sent to the controller[77]. The trajectory results are suitable for use in an outer loop controller.

4.8.2.3 Moving Target Results

As target speed increases, the path that the UAV must fly to surveil it changes. To make a full comparison between the neural network method and the other methods, various target speeds are used and the UAV paths are compared. A 30 second horizon time is used for these plots with 15 nodes. The pseudospectral

![Figure 4.20. Controls (longitudinal acceleration and bank angle)]
method performs more accurately with an increased number of nodes. However, it
does not remain competitive in terms of computation time with the neural network
or analytic direct collocation method. The target-in-view function likely requires
more nodes to be accurately integrated.

![Comparison results for a target moving at 10 ft/sec for the neural network method (NN), DCNLP, and the pseudospectral (PS) method with target coverage percentages given](image)

Figure 4.21. Comparison results for a target moving at 10 ft/sec for the neural network method (NN), DCNLP, and the pseudospectral (PS) method with target coverage percentages given

As shown in the plots, the neural network method results in paths very sim-
lar to the direct collocation and pseudospectral methods. In addition, a 16 sec-
ond horizon with 8 nodes was used in simulation. Note that the psuedospectral
method’s result is different than the direct collocation and neural network method.
The cause of this difference is the roll control input. The neural network and di-
rect collocation methods tend to choose to turn in one direction only while the
pseudospectral method tends to alternate turn directions. Note that in for certain
target speeds, one method may have a significantly different target observation ra-
tio while still have a similar path. This is simply a question of tuning the weighted
objective function. For this comparison, the weights were kept constant. However,
it is trivial to schedule weights with target speed.

Figure 4.26 plots the relative computation times for the various methods for the
16 second horizon. Figure 4.27 shows the same for the 30 second horizon. Table 4.8
Figure 4.22. Comparison results for a target moving at 20 ft/sec for the neural network method (NN), DCNLP, and the pseudospectral (PS) method with target coverage percentages given.

summarizes the mean path generation time for the various cases. Relative to the direct collocation method, the neural network approximation method requires between 2.5-5 times less computation time than the fastest method using analytic derivatives. The ratio jumps to 60-80 times faster when using complex-step derivatives. Automatic differentiation performed similarly to complex-step derivatives. Table 4.9 lists the relative computation time for the different methods. Finally, all methods generally produce the same amount of target coverage. The simulation was run for 100 seconds. Coverage time is simply the time the target is in view divided by the simulation time. Table 4.10 lists the coverage time for the various methods for a 30 second horizon. The 16 second horizon results are not shown, as the only main difference is the lessened prediction common to all methods. For example, at $V_{targ} = 35 \text{ ft/s}$ and 16 second horizon, none of the methods are able
Figure 4.23. Comparison results for a target moving at 25 ft/sec for the neural network method (NN), DCNLP, and the pseudospectral (PS) method with target coverage percentages given to achieve 100% observation time because they are unable predict far enough into the future.

4.8.2.4 Dual UAV Comparison

Transforming the single UAV trajectory optimization into a multiple UAV formulation is straightforward with the neural network method. *No network retraining is required* for this extension to the problem because the problem is modular. The varied parameter vector is augmented with the controls of the second UAV. In addition, the dynamics and objective networks are expanded with appropriate loops so they are called repeatedly for multiple UAVs. Finally, the gradient calculation for individual UAVs is generally combined in block diagonal form with constraints.
from each UAV in each block. Other than minor housekeeping issues, the extension to multiple UAVs is straightforward. The dual-UAV covering a moving target scenario is presented in Figure 4.28. After the initial settling time, the UAVs assume a regular pattern of alternating passes over the target.

### 4.8.3 Comparative Conclusions

As shown in the above results, for the given UAV path planning problem, the neural network approximation method generally matches the optimization performance of the direct collocation and pseudospectral methods but requires 2-5 times less computation time than the fastest method using analytical derivatives and 5+ times less for automatic derivative methods. This shows good promise for real-time
Figure 4.25. Comparison results for a target moving at 35 ft/sec for the neural network method (NN), DCNLP, and the pseudospectral (PS) method with target coverage percentages given.

Of course, speed up will be obtained by converting all methods to a compiled programming language and certain methods shown would be suitable for real time operation. However, the fact that the neural network approximation does not require analytic derivatives to achieve the necessary speed is an advantage. It does require more setup time than the other methods for training the network, but this is generally an automatic process. Training the network required approximately 1.5 hours for 40,000 training vectors. Coupled with one of the other methods using automatic derivative calculation for verification purposes, the neural network approximation method could be quite useful for real time optimization control.
Figure 4.26. Relative computation time for various path planning methods using a 16 second horizon
Figure 4.27. Relative computation time for various path planning methods using a 30 second horizon

(a) Ground Path (9 nodes, 16 sec horizon, 93% coverage, 0.76 sec average calculation time)

(b) Target coverage timeline

Figure 4.28. Ground track and coverage timeline for two UAVs and a target at 15 ft/s
<table>
<thead>
<tr>
<th>$T_h$ (s)</th>
<th>Method</th>
<th>Deriv.</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>25</th>
<th>27</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>NN</td>
<td></td>
<td>0.2 (0.1)</td>
<td>0.2 (0.1)</td>
<td>0.2 (0.1)</td>
<td>0.2 (0.1)</td>
<td>0.2 (0.1)</td>
<td>0.2 (0.1)</td>
</tr>
<tr>
<td></td>
<td>DCNLP</td>
<td>AN</td>
<td>0.7 (0.2)</td>
<td>0.9 (0.2)</td>
<td>0.9 (0.3)</td>
<td>0.9 (0.4)</td>
<td>0.9 (0.5)</td>
<td>1.1 (0.6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>14.8 (4.2)</td>
<td>15.8 (5.0)</td>
<td>16.7 (5.1)</td>
<td>16.1 (3.9)</td>
<td>16.7 (3.6)</td>
<td>14.0 (3.8)</td>
</tr>
<tr>
<td></td>
<td>Pseudo.</td>
<td>AD</td>
<td>2.8 (1.0)</td>
<td>2.5 (0.8)</td>
<td>2.3 (0.6)</td>
<td>2.3 (0.7)</td>
<td>2.4 (0.7)</td>
<td>2.4 (2.4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>1.7 (0.6)</td>
<td>1.5 (0.4)</td>
<td>1.4 (0.4)</td>
<td>1.4 (0.4)</td>
<td>1.4 (0.4)</td>
<td>1.5 (1.5)</td>
</tr>
<tr>
<td>30</td>
<td>NN</td>
<td></td>
<td>0.8 (0.3)</td>
<td>1.3 (0.6)</td>
<td>1.0 (0.5)</td>
<td>1.0 (0.3)</td>
<td>1.1 (0.4)</td>
<td>1.4 (0.6)</td>
</tr>
<tr>
<td></td>
<td>DCNLP</td>
<td>AN</td>
<td>3.0 (0.8)</td>
<td>2.8 (0.9)</td>
<td>3.0 (0.7)</td>
<td>3.2 (0.6)</td>
<td>3.8 (0.9)</td>
<td>4.3 (0.8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>98.0 (26.7)</td>
<td>82.5 (15.9)</td>
<td>74.4 (15.5)</td>
<td>70.0 (16.9)</td>
<td>72.5 (17.9)</td>
<td>69.0 (11.8)</td>
</tr>
<tr>
<td></td>
<td>Pseudo.</td>
<td>AD</td>
<td>8.0 (1.9)</td>
<td>6.7 (1.5)</td>
<td>7.4 (2.2)</td>
<td>6.4 (2.1)</td>
<td>6.3 (1.5)</td>
<td>6.9 (1.1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>9.0 (2.2)</td>
<td>5.8 (1.3)</td>
<td>6.4 (1.9)</td>
<td>5.6 (1.7)</td>
<td>5.6 (1.3)</td>
<td>6.3 (0.9)</td>
</tr>
</tbody>
</table>

**Table 4.8.** Path generation time, mean (std. dev.) sec, $N_{nodes} = T_h/2$ (AN - analytic derivatives, AD - automatic derivatives, CS - complex-step derivatives)
Table 4.9. Relative path generation time compared to neural network method, \( N_{\text{nodes}} = T_h/2 \) (AN - analytic derivatives, AD - automatic derivatives, CS - complex-step derivatives)

<table>
<thead>
<tr>
<th>( T_h(s) )</th>
<th>Method</th>
<th>Deriv.</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>25</th>
<th>27</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>NN</td>
<td>-</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>DCNLP</td>
<td>AN</td>
<td>3.5</td>
<td>4.5</td>
<td>4.5</td>
<td>4.5</td>
<td>4.5</td>
<td>5.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>74.0</td>
<td>79.0</td>
<td>83.5</td>
<td>80.5</td>
<td>83.5</td>
<td>70.0</td>
</tr>
<tr>
<td></td>
<td>Pseudo.</td>
<td>AD</td>
<td>14.0</td>
<td>12.5</td>
<td>11.5</td>
<td>11.5</td>
<td>12.0</td>
<td>12.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>8.5</td>
<td>7.5</td>
<td>7.0</td>
<td>7.0</td>
<td>7.0</td>
<td>7.5</td>
</tr>
</tbody>
</table>

Table 4.10. Target coverage (percentage of total simulation time)

<table>
<thead>
<tr>
<th>( T_h(s) )</th>
<th>Method</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>25</th>
<th>27</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>NN</td>
<td>35%</td>
<td>42%</td>
<td>50%</td>
<td>61%</td>
<td>63%</td>
<td>87%</td>
</tr>
<tr>
<td></td>
<td>DCNLP</td>
<td>38%</td>
<td>40%</td>
<td>39%</td>
<td>53%</td>
<td>43%</td>
<td>74%</td>
</tr>
<tr>
<td></td>
<td>Pseudo.</td>
<td>41%</td>
<td>43%</td>
<td>48%</td>
<td>48%</td>
<td>46%</td>
<td>76%</td>
</tr>
<tr>
<td>30</td>
<td>NN</td>
<td>40%</td>
<td>33%</td>
<td>46%</td>
<td>64%</td>
<td>73%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>DCNLP</td>
<td>39%</td>
<td>37%</td>
<td>50%</td>
<td>69%</td>
<td>76%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Pseudo.</td>
<td>42%</td>
<td>42%</td>
<td>50%</td>
<td>65%</td>
<td>67%</td>
<td>100%</td>
</tr>
</tbody>
</table>
Chapter 5

Real time implementation and Flight Test

Simulation of a trajectory optimization method is quite different than actual implementation. For example, simulation bypasses any requirements for real time operation and does not require consideration of issues related to communicating with hardware or other software. This chapter looks at the implementation of the direct collocation method for use in real-time aboard the Applied Research Laboratory/Penn State (ARL/PSU) testbed UAV. Results from flight testing are presented. These results were originally published in Reference [77] which was combined from two earlier conference papers ([37, 80]).

5.1 UAV Testbed Description

The ARL/PSU testbed UAV is a modified Sig Kadet Senior. The Kadet is a basic balsa-construction .60-size trainer aircraft that comes in almost-ready-to-fly (ARF) kit form. The ARF kit allows for easy modification and replacement parts if needed. The modified aircraft has a payload of about five pounds, gross weight of 14 pounds, and an 80 inch wingspan. The payload includes a CloudCapTech Piccolo Plus autopilot, Ampro ReadyBoard 800 single board computer equipped with a 1.4 GHz processor and 1 GB memory, a USB webcam or a digital camera, and 4000 mAh lithium polymer battery. Figure 5.1 shows external and internal views of the aircraft. Visible in Figure 5.1(b) are the Ampro ReadyBoard 800
towards the left, the autopilot in the middle, and the white fuel tank in the back-
ground on the right. The Kadet is heavily modified from its kit form. Heavy duty
landing gear/mounting hardware, .90-size 4-stroke engine, and rearrangement of
internal servo mounts are necessary for maximum payload volume and weight.

Figure 5.2(a) shows the modified main landing gear along with the payload bays.

Starting from the left: (1) servo battery and nosegear servo housing, (2) camera,
webcam, or mission specific payload bay, (3) fuel tank. The space behind the main
landing gear mount is empty. Figure 5.2(b) shows the empennage servos’ loca-
tions. Figure 5.1(c) shows the cowling, the modified nosegear, and webcam/digital
camera mount. Behind the cowling, a larger engine mount is installed and the
throttle servo is relocated to the outside of the firewall (normally inside the air-
craft). Additional modifications include the mounting of GPS, 900 MHz, and
802.11b antenna ground planes. As shown in Figure 5.1(a), these can be seen on
the on right side of horizontal stabilizer, large antenna just behind the wing, front
tip of the vertical tail, for the GPS, 900 MHz, and 802.11b antennae respectively.
The pitot-static system, provided by CloudCap Technologies, consists of a carbon
fiber pitot-static probe with corresponding mounting hardware installed on the
right wing.

During the assembly of the UAVs, the onboard computer was found to interfere
with GPS reception. GPS satellites broadcast on two frequencies: L1 at 1.57542
GHz and L2 at 1.2276 GHz. Because the processor operates at 1.4 GHz, inter-
ference is likely (and did occur) if the GPS antenna is too close to the computer.
The problem was solved by putting the GPS antenna on the horizontal tail of the
Kadet. This setup has been operated without additional interference problems and
generally receives signals from between 6 & 10 GPS satellites. Note that 4 satel-
lites are required for a 3-dimensional position fix, and the GPS unit in the Piccolo
Plus autopilot is capable of tracking 12 satellites simultaneously (a Motorola M12
Oncore).

Onboard power is supplied by a 12V, 4000 mAh lithium-polymer (LiPo) battery,
providing approximately 1.5 to 2 hours of system operating time, depending on
processor load. The Ampro single board computer is powered by a V-Infinity 12V-
to-5V DC power converter, which is connected to the LiPo battery. The power
converter can be remotely powered on or off from the Piccolo Command Center
Figure 5.1. The Applied Research Lab/Penn State UAV Testbed
to conserve power or in a situation where the computer is interfering with the safe operation of the aircraft. The Ampro board is equipped with 1GB RAM and a 4GB CompactFlash memory card which serves as main storage. Windows XP is installed and ground operations are accomplished using Remote Desktop or via SSH. The computer has two serial ports connected to the two serial ports on the Piccolo to enable custom code interaction with the autopilot. Servos are powered separately by a 6V, 2700 mAh nickel-metal-hydride battery. Figure 5.3 shows a schematic of the overall system.

The limiting factor for endurance is fuel load. With a 24 ounce fuel tank, the UAV can be flown at cruise speeds (30 knots) for approximately 1 hour. The longest flight completed to date is 50 minutes. A 32 ounce fuel tank can be fitted for increased flight time with reduced payload capacity. With the larger fuel tank, battery capacity becomes the limiting endurance factor.
The onboard camera is hard mounted to the airframe and points downward. A Logitech Ultra Vision USB webcam with a $63^\circ$ horizontal and $50^\circ$ vertical field of view is used. Video output has a resolution of 640 by 480 pixels, and is captured using OpenCV\cite{81}, an open source computer vision library created by Intel in C++. The UAV also is able to accept a 4 megapixel Canon digital camera.

The ARL/PSU UAV team has three Piccolo equipped UAVs, two with on-board single board computers. The Sig Kadet platform has proved itself to be a reliable platform for research. Over the years of 2006-2008, the team has logged nearly 50 flights with the Kadet UAVs, including simultaneous operation of two UAVs to facilitate path planner testing and future collaborative research efforts\cite{36}. Though not flown yet, the ARL/PSU group is building two larger UAVs based on the Super Flyin’ King kit from Bruce Tharpe. These aircraft will support expansion of the Intelligent Controller work done by Miller \cite{35}.

All flight operations take place at the Centre Airpark airfield, a private grass strip located near Old Fort, Pennsylvania. The runway is 3100 ft long and 210 feet wide. Large red barrels line the run way boundary. These barrels are used as observation targets in some tests because they are painted red and can easily been seen by the camera onboard the UAV. Figure 5.4 shows an aerial photo of Centre Airpark, the target exercise balls, and a runway boundary barrel.
Figure 5.4. Airfield Photos and test target photos
5.2 Implementation Issues & Initial Flight Tests

Initially, a separate path navigator program was written. It read the path output from the optimization and sent turn rate commands to the autopilot. The reason for this was to have greater control over how the path is followed. Only limited control is available through the gains on the autopilot. However, the onboard computer must run the path planner, the path navigator, and any image capture software. In light of the requirement to run the path planner in real time, it would be best to offload as many tasks from the onboard computer as possible. Additionally, there would be increased programming complexity as communications between the path navigator and planner would be required. The autopilot is provided with a software development kit which makes communications with external devices easy to implement. Therefore, only the North and East coordinates of the path nodes output from the path planner are sent to the autopilot. The autopilot is then commanded to fly this path.

The connection between the path planning algorithm and the Piccolo autopilot is then fairly simple. Once a path is computed, the $x, y$ coordinates of each node are converted to latitude and longitude. The string of points is converted to the proper format and sent to the Piccolo (using the Cloud Cap Software Development Kit). Since there is a time lag between receiving the current UAV position, computing the path, and sending the waypoint list, the path planner chooses the correct waypoint on the list to command the autopilot to fly to. The chosen waypoint is one not yet reached on the newly updated path. The autopilot will then follow this path until the next update. Speed commands are sent directly by periodically checking where the UAV is on the computed path and sending the appropriate speed command.

5.2.1 Real-Time Implementation

A C++ version of the path planning algorithm capable of running in real time has been developed and tested. As discussed in Section 4.6, the SNOPT[13] package is used as the nonlinear solver based on personal experience and recommendations of others who have used it. Since SNOPT has a C++ and MATLAB interface, it can be integrated with both the C++ path planner implementation and with the
existing MATLAB simulation. Verifying the correctness of the C++ implementation compared to the MATLAB implementation is aided by the ability to use the same solver in both languages.

A second change was the use of the analytical derivatives of the objective function. Initially, numerical derivatives were used. However, numerical derivatives are slow compared to analytical derivatives, and the path planner is intended to operate online. Of course, analytical derivatives can be very complicated and tedious to obtain. Computer algebra packages such as Mathematica or Maxima can be used to calculate the full derivative, but the resulting expression is very long and most likely contains redundant calculations. If the derivatives of smaller sections of the objective function are taken, the final objective and constraint Jacobian can be obtained using the chain and product rules of differentiation. This results in a longer but more manageable analytical derivative.

Several compromises must be made for the DCNLP method to run in real time. Processing time is essentially driven by the complexity of the objective function. The most complex part of the current objective function is the target-in-view cost which requires calculation of the pixel coordinates given the real world coordinates of the target and position and orientation of the UAV. Furthermore, an analytical expression for the integral over the collocation points cannot be written unlike the distance-to-target and control effort costs. Numerical integration is used, which adds more computation time. Several options present themselves for real-time operation. The first is a reduction in the number of nodes. This would result in fewer calculations at the expense of reduced accuracy in the interpolating polynomials used to approximate the UAV and target equations of motion. In turn, this means that the generated path would have a greater probability to exceed the physical limitations of the UAV (such as turn rate). However, by reducing the horizon time, this effect can be mitigated. A second option is to increase the horizon update interval. This means that the UAV would follow a generated path for a longer amount of time, without the benefit of an update of the world state.

5.2.1.1 Hardware-in-the-Loop Simulation

Before flight tests were performed with the path planner operating in real time onboard the UAV, hardware-in-the-loop (HIL) simulation was performed with the
same computer that is flown on the UAV. Several combinations of horizon time, number of nodes, and horizon update interval were tested to find settings that would be best to flight test. A stationary target was used in the simulation. These HIL simulations test the ability of the path planner to generate a viable path and to do so within the horizon update interval. The standard setup used in simulation was 11 nodes, a 30 second horizon time, and a horizon update every 1.5 seconds. Using 11 nodes, the algorithm takes around 3 seconds on average and up to 6 seconds to generate a new path. This produced a very smooth cloverleaf pattern around a stationary target, however it was too slow to use in real-time operation because of the very short path update interval. Increasing the update interval to 6 seconds allowed the path planner to generate a new path before another update was required. While this was satisfactory, the path generation was still fairly slow at 3 seconds per path. To obtain a shorter path generation time, the path planner was tested using 7 nodes and a 20 second horizon time. On average, the path planner requires around 1 second to generate a new path in this configuration. Using an update interval of 4 seconds with these settings generates a viable path while maintaining a buffer between the time required to generate a path and the path update interval. The two configurations tested in flight tests are given in Table 5.1. The results section discusses the advantages and disadvantages of these configurations.

### 5.3 Flight Test Results

The limits shown in Table 5.2 were used in the following flight test results. They are based on the estimated performance limitations of the Sig Kadet Senior. As a safety margin, the bank angle limit is lower than that which the UAV is capable.

The path planner was flight tested while operating in real time onboard the UAV. The test scenarios performed include a stationary target in calm and steady
<table>
<thead>
<tr>
<th>Constraint</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stall Speed</td>
<td>$V_{\text{min}} = 22 \text{kts}$</td>
</tr>
<tr>
<td>Maximum Speed</td>
<td>$V_{\text{max}} = 50 \text{kts}$</td>
</tr>
<tr>
<td>Maximum Longitudinal Acceleration</td>
<td>$u_{l_{\text{max}}} = 10 \text{ft/s}^2$</td>
</tr>
<tr>
<td></td>
<td>$u_{l_{\text{min}}} = -10 \text{ft/s}^2$</td>
</tr>
<tr>
<td>Maximum Bank Angle</td>
<td>$\phi = \pm 30^\circ$</td>
</tr>
</tbody>
</table>

Table 5.2. UAV performance constraints

wind conditions, a walking person, a moving vehicle, and a second UAV flying at a lower altitude. The results of these tests are discussed below. Figures that show the ground track of the UAV use a greyscale aerial photo of the flying field as the background. The end of the ground track is indicated by a triangle. Also visible are 5 dots which mark the locations of barrels used for targets. If they are relevant to the results, they are circled and labeled. For cases with two UAVs where one UAV acts as a target, the tracker UAV is shown as a solid line and the target UAV is shown as a dashed line.

For these results, the path planner was given the target’s location through a ground station uplink. In Section 5.4, results using integrated path planning and target geolocation are presented. In order to relay the target location from the ground to the UAV above, a second Piccolo autopilot was used as the target, be it handheld, in a truck, or on a second UAV. The ground station communicates with both Piccolos simultaneously. It also has the ability to run a sever on the local wireless network which serves the data it receives from the autopilots. The tracking UAV connects to the ground station server over its onboard 802.11b connection, and requests all data specific to the Piccolo being used as a target. This data includes GPS location and ground speed.

5.3.1 Stationary Target

For this test, an airfield barrel was chosen as the target to observe. Its latitude and longitude was input to the path planner prior to starting the test. Figure 5.5(a) shows the ground track of the UAV while under the direction of the path planner. The familiar clover leaf pattern seen in simulation is readily apparent. Figure 5.5(b) is a histogram showing the time required to calculate each new path update. A
majority of the path updates require under one second and no path update took longer than the update interval (4 seconds for this flight), showing that the planner is operating in real time. For both configurations, the camera has a view of the

Figure 5.5. Stationary target observation in calm winds

target 41% of the time. This percentage is measured by watching the recorded onboard video after the flight and measuring the length of each time the target is
in view and dividing by the length of time from the first view of the target to the
end of the test.

5.3.2 Stationary Target in steady winds

Figure 5.6(a) shows the results of observing a stationary target in a 5 knot wind
coming from the west (the left side of the figure). Note that the Kalman filter
on Piccolo autopilot is able to estimate wind speed using airspeed and GPS mea-
surements. The wind estimate is then used in the optimization. Previously, flight
tests were conducted without accounting for steady winds resulting in very poor
target observation time. Using wind estimates, the figure-8 pattern seen in sim-
ulation emerges here in the flight test. Also note that all of the turns are made
into the wind; this minimizes the ground track turn radius. Comparing the path
update time histogram (Figure 5.6(b)) with the calm wind case shows that comp-
ensating for wind requires slightly more processing time, which is to be expected.
Configuration 2 was used in both of the results shown for the stationary target
tests (7 nodes, 20 second horizon time, 4 second update interval). Configuration 1
produced similar results and was able to operate in real time. The only significant
difference was the amount of processing time required. For configuration 1, the
onboard camera has a view of the target 41% of the time, while for configuration
2 achieves 40% time coverage.

5.3.3 Tracking a moving ground target

For the second set of flight tests, a spare Piccolo autopilot was used as a mobile
target. The laptop connected to the ground station runs a server on the 802.11
wireless network that mirrors the data received from the autopilots. The computer
onboard the tracker UAV connects to this server over the 802.11 network and is
able to receive all data from the target autopilot.

Both a person walking with the target Piccolo and a truck driving down the
adjacent road were used to test the mobile target tracking ability of the path
planner. We found that a person walking was not fast enough to produce much
variation in the path (other than the motion of the path center) of the UAV
compared to a stationary target. Figure 5.7(a) shows a typical result. The person,
Figure 5.6. Stationary target observation with 5 knots wind from the west represented by the dashed line, is walking along the edge of the field heading toward the Northeast. The UAV, shown as a solid line, repeatedly circles over the person as he moves down the field. The settings used for this test are a 30 second horizon time, 10 segments, and 6 second update interval. For this case, the real-time performance is bordering on exceeding the time requested for the update interval. When this happens, the path planner continues to operate, but the UAV traverses a longer portion of the path before it is updated. In extreme cases, the UAV may...
reach the end of the path before a new update is available and turn around to the beginning. This would mean that real-time operation is not possible. Figure 5.7(b) shows average time to generate a new path is approaching the update interval. This is likely due to the inclusion of target motion; the optimization requires more time to converge with more variables in play. The path planner achieved 38% time coverage for the walking person, and 40% coverage with the person running.

Figure 5.7. Observation of a walking person

To test the performance of the path planner with a faster moving ground target,
a pickup truck driving down a road adjacent to the airfield was tracked. The target Piccolo autopilot was placed in the truck. Due to the hilly terrain, there was a limit on how far away the truck could be before the signal to the target autopilot was lost. Therefore, only a limited amount of road was available for the tracking test. The truck drove at 20 miles per hour. Figure 5.8(a) shows the ground track of the truck as a dashed line and the UAV as a solid line. This test mainly shows the behavior of the path planner when the target is stopping and starting, as there was not enough space to safely track the truck with the UAV. There are still some real-time implementation problems as shown by the timing plot in Figure 5.8(b). The cause of the long processing times was the truck stopping completely to turn around. This means the initial guess at the correct path would be fairly bad and the optimization will take longer to converge. A shorter update interval may be advantageous when tracking a mobile target as it would allow for more timely updates when the target changes direction. As the path planner is currently configured, it assumes the target will always continue following its current heading. Using a predictive target model similar to the road data model discussed in Section 4.4.1 would bring improvement.

5.3.4 Tracking a flying UAV from above

Because ground space with which to test the tracking of a vehicle is limited surrounding the airfield, a ground vehicle is ‘simulated’ by using a UAV flying at a lower altitude than the tracking UAV. Using the same method discussed in Section 5.3.3 to upload the target’s GPS position to the tracker UAV, the target UAV was flown on various flight plans 300 feet below the tracker UAV. This way, we have much greater freedom to test the performance of the path planner when tracking a moving target.

The results shown in this section required a change in the implementation compared to the previous results. Speed commands were not sent in the above stationary and slow moving target cases because the optimal observation airspeed is the minimum safe airspeed. For the following cases with a fast target, speed commands in addition to the path waypoints were sent to the autopilot. We found from an initial test that fixed speeds wouldn’t work even when the observation
Figure 5.8. Observation of a moving truck (20 mph); Time required to generate a new path

and target UAVs’ speeds were the same. Note that speed commands cannot be updated when the path is being calculated. Since the target’s speed does not vary quickly, this limitation did not cause a problem. The limitation could be overcome by using an intermediary program to send commands to the autopilot.

The initial test tracked a UAV around a simple rectangular pattern. For this test, the tracker UAV was allowed to match the speed of the target UAV. Figure 5.9 shows the start of the maneuver. The tracker UAV is shown by the solid line and
starts from its parking orbit. The tracker smoothly intercepts the target UAV and assumes position above it. The path planner generates a path that commands the UAV to speed up to close the distance to the target. This can be seen at 1400 seconds in Figure 5.9(b). As the tracker UAV approaches the target, it is commanded to slow down as seen at 1420 seconds. When the target UAV turns the corner of its flight plan, the path planner commands the UAV to speed up again. When the target UAV turns the corner of its flight plan, the tracker UAV

Figure 5.9. Initial acquisition of the target UAV from a parking orbit continues on for a short while until the path planner updates the path with the new
heading of the target. Figure 5.10 shows the UAVs on their second time around the pattern. The tracker UAV overshoots the target when the target turns a corner, but accelerates to make up ground and decelerates to reacquire the target. This behavior can be clearly seen in Figure 5.10(b). Onboard video was not available for this test because the UAV lost engine power and had a hard landing, causing the video to be lost. However, based on the ground track, the target is estimated to be covered 35% to 45% of the time.

A following flight retested this scenario with a slightly different rectangular pattern. When the UAV was allowed to match the target’s speed, the coverage time was measured to be 33%. When the UAV is limited to 10 knots faster than
the target, the coverage time is 25%. However, it was clear from both videos that for a significant amount of time, the target was just out of view of the bottom of the frame. The optimization and equations of motion assume that the UAV does not pitch. However, in flight the UAV is pitched up some actual amount, which would point the camera forward. We may see better results by accounting for even a constant amount of pitch angle based on the trim attitude of the UAV at some nominal airspeed. However, new analytical derivatives would need to be calculated to include pitch attitude.

These next results show flights test with the path planner using road data. In this case, the ‘road’ is simply the flight plan of the target UAV. The current position along the flight plan and speed of the target UAV is sent to the tracker UAV. The path planner then uses this information in predicting the future path of the target when generating a new trajectory. A figure-8 flight plan was used for the target UAV. The first scenario has the tracker UAV limited to 15.4 meters/second (30 knots) minimum airspeed, while the target can travel at 10.3 m/s (20 knots). Figure 5.11(a) shows the ground track and Figure 5.11(b) shows that the majority of the time, the tracker UAV is at its minimum allowed airspeed. For this scenario, the target coverage time was 27% of the total time.

The second scenario tested with the figure-8 ‘road’ was the case when both target and tracker UAV can fly at the same speed. The ground track of the UAVs is shown in Figure 5.12(a) and is continued in Figure 5.12(b). The speed of the target UAV was set to 15.4 m/s (30 knots) to match the minimum allowed airspeed of the tracker from the previous scenario. The average path calculation time for this and the previous scenario is 1.5 seconds. While the target is tracked closely, the tracker UAV is never able to maintain position directly over the target UAV. It seems to consistently lag approximately 2-3 seconds behind the target. This may be due to the latency involved in downlinking the target’s position to the ground station and then resending it to the tracker UAV. This apparent lag could be corrected by applying a correction to the position of the target UAV on its path proportional to its speed and the estimated latency. In this scenario, the target coverage time was 69% of the total time.

The following figures show a frame capture from the onboard video. Figure 5.13(a) shows a person walking dragging two balls (for increased visibility
Figure 5.11. Tracking of the target UAV around a figure-8 pattern with a 5 m/s (10 knot) speed difference in the video. Figure 5.13(b) shows a target UAV flying 300 ft below. The person and UAV are circled in white.

5.4 Integrated Target Tracking and Path Planning

The previous flight test results are run with the target location being sent to the UAV in flight. An integrated implementation is now discussed that is able to
Figure 5.12. Tracking of the target UAV around a figure-8 pattern where tracker can match target speed

operate in a self-sufficient fashion. Using an image processing routine coupled with a Kalman filter, a twin-red ball target can be identified and tracked without external communication [39]. The red ball target serves only as a test target for the image processing algorithm. Any image processing/target localizer could be coupled with the trajectory optimization. As such, only the software components involved in tying all systems together are discussed here along with some hardware-in-the-loop (HIL) setup and HIL and flight test results.
To allow both the path planner and target localizer to run simultaneously, several issues must be addressed. Both programs require UAV telemetry data, and both need to communicate with each other. The path planner needs to send commands back to the autopilot as well. Figure 5.14 shows the basic layout of the
required connections. Timely communication between the processes is paramount,

![Diagram of hardware-in-the-loop simulation setup]

Figure 5.14. Schematic of the hardware-in-the-loop simulation

so a shared memory scheme was chosen. The host controller receives telemetry from
the autopilot over a serial connection and places the data structure in a shared
memory location so the path planner and target geolocation processes can access
it. By continually updating the telemetry, each process always has the most recent
available data without the need for polling or requesting it. The target location
shared memory is continually updated with estimated target location from the
Kalman filter. The path planner reads the target location information and creates
a path. When path generation is finished, the path planner loads the path into
shared memory, and then sets the shared update flag. The host controller checks
this flag each cycle; if set, the shared memory containing the path is read and
sent to the autopilot. To avoid deadlock issues, no process depends on another to
continue running. Timing data is recorded for all processes to measure how the
integrated setup performs.

5.4.1 Hardware-in-the-loop simulation description

In the hardware-in-the-loop (HIL) setup, the autopilot is connected to a computer
that simulates flight dynamics. The Ampro ReadyBoard is connected to the au-
topilot and runs the image processing and path planning software. The camera is connected to the USB port of the ReadyBoard and is pointed at a LCD which displays aerial images generated by Flightgear[82], thus simulating what it would actually see in flight. Figure 5.15(a) shows the setup. Figure 5.15(b) shows an example of alignment calculation using the camera calibration toolbox.

Several problems present themselves with this setup. The most basic is the choice of screen. A CRT screen cannot be used as the webcam will pick up flickering as the screen is redrawn. While tuning a camera to prevent flickering is possible, most cheap webcams do not have this option. Therefore, an LCD is used to avoid flickering. The second problem to solve is correct positioning of the camera; it should be aligned with the normal vector of the LCD screen’s surface. To compute the orientation of the camera with respect to the screen’s surface, the camera calibration toolbox is used with the calibration pattern (shown in Figure 5.15(a)). The required roll, pitch, and yaw corrections for the camera of the camera is easily computed. An alignment to within 0.3 degrees in all axes was attained. The field of view in Flightgear is then set to match the field of view of the camera.

The final component of the HIL system is the actual visualization of the terrain. Flightgear comes with default world scenery that is fairly sparse. It does have limited support for photorealistic scenery over small areas. Using 1 ft/pixel resolution aerial photos of the flying field from Pennsylvania Spatial Data Access

Figure 5.15. Webcam HIL rig and calibration result
(PASDA) [83], a 4096 $ft^2$ area centered at the runway was imported into Flightgear using the *photomodel* command (available when Flightgear is compiled from the source). Note that the area should be a power of two (in this case, 4096 pixels square) for the texture to be properly displayed in Flightgear. Shown in Figure 5.16 is a comparison between actual picture recorded in flight and visuals produced by Flightgear using the photorealistic scenery. The webcam reduces the contrast and detail of the Flightgear visuals, however it is still usable by the target recognition algorithm. The darkening in the top half of Figure 5.16(c) is due to the viewing angle limitations of the LCD screen. One problem with using photorealistic scenery in Flightgear is that it is difficult to apply the aerial photo over an elevation model. Therefore, the model used here is simply a flat plane. The focus of the HIL simulation is not to test absolute localization accuracy, but to provide a platform for debugging the system in a “realistic-enough” environment.

![Figure 5.16. Comparison of actual and simulated aerial views](image)

(a) Actual image captured in flight  
(b) Flightgear screen shot  
(c) Flightgear as seen through the webcam
5.4.2 Path Planning

The integrated path planner and target geolocation code results are presented here for the HIL simulation test. Figure 5.17(a) shows the ground path of the UAV; the path planner generates a clover leaf pattern similar to that seen in simulation and flight tests where the target location was known. Figure 5.17(b) shows individual target observations and the positions of the UAV when the observations were made.

![Ground track](image)

![Target observations](image)

(a) Ground track  (b) Target observations

Figure 5.17. Integrated path planner and target geolocation in HIL simulation

The path planner and target localizer work well together in the HIL simulation. Note that this exact hardware configuration would be flown. The only change is that the camera would be looking out from the bottom of the aircraft instead of at the LCD monitor. This particular test case ran the target localizer at 4 Hz and the path planner using 7 nodes, a 20 second horizon length, and a 2 second horizon update time. Figure 5.18 shows timing results for both the path planner and target localizer. Note that in only 3 out of 251 instances does the path generation time exceed the horizon update interval of 2 seconds. Additionally, these excursions are within 0.5 seconds of the update interval and have a negligible effect on the real-time performance of the path planner. The target geolocation algorithm showed an average processing time of 0.086 seconds per run but with several runs taking as long as 1 second (note it is expected to run at 4 Hz). These extended processing times are most likely due to the path planner running and
consuming all available processing resources. However, the accuracy of the target geolocation is not affected because the UAV state data is saved at the beginning of the run and thus does not change throughout (even as the shared memory is updated). The Kalman filter can be adversely affected by this if it is trying to estimate the target velocity. These timing results show that both the path planner and the target geolocation algorithm are able to run simultaneously at the expected update rates.

![Path planner processing time](image1)
![Target geolocation processing time](image2)

**Figure 5.18.** Histogram of processing times

### 5.5 Discussion

Of the two configurations flight tested (Table 5.1), Configuration 2 performed adequately in all cases. This configuration maintained a significant margin between path calculation time and the update interval. The average path calculation time for Configuration 2 was around 1.5-2 seconds using an update interval of 4 seconds, which makes available processor resources for other tasks such as image processing. Configuration 1, which used more nodes and a longer horizon time, worked in the simpler stationary target scenarios, however its performance suffered when a mobile target was used due to its longer update interval.

To achieve the computation speed necessary to use this method in real time on this particular processor, analytical derivatives for the objective and constraint gradients are required. This hampers making even small changes to the objective or constraint functions (useful in research) as the derivatives must be recalculated,
which is tedious and can introduce errors. However, writing analytical derivatives are not impossible, and only need be written once if the general mission scope of the UAV does not change.

While the method depends entirely on the convergence of the nonlinear solver, the UAV is not physically at risk from invalid path solutions. The system is set up to send only waypoint and speed commands to the autopilot, and simple checks can be implemented to make sure invalid commands are not used. Therefore, the UAV can be prevented from proceeding along a trajectory that would cause harm or cross boundaries.

Neglecting pitch angle does reduce observation performance. This could be corrected by including pitch in the actual observation, but a more useful fix may be to simply add pitch to the camera orientation matrix instead of adding to the equations of motion. This would avoid adding parameters to the optimization problem while at the same time accounting for aircraft pitch. The pitch angle could be scheduled with airspeed. If the target observation is assumed to take place at a constant airspeed, the pitch angle could be held constant over the horizon time.

Note that the algorithm is being run on a processor released in 2004. During flight tests, observed CPU usage averaged approximately 50%. Thus with proper setup, a simple image processing routine to track a specific target could be integrated. With the release of more powerful processors since 2004, including low power multi-core designs, it is reasonable to state that the method presented would be useful in a more complete airborne package.
Chapter 6

Conclusions and Future Work

6.1 Conclusions

6.1.1 Collocation Methods and Flight Demonstration

Simulation and convergence testing shows that the direct collocation method is suitable for use in a real-time surveillance time maximizing trajectory optimization problem. Additionally, a suitably small number of nodes can be used to enable real-time performance while maintaining an acceptable level of observational performance. Augmenting the trajectory optimization with road data allows the method to more accurately predict target motion, however, any type of predictive target motion model can be used. A perspective driven modification to allow specific view of the target to be captured is also presented and performs satisfactorily but has room for improvement.

Recommendations for horizon length and node spacing are as follows: in general, the horizon length should be small but long enough to predict one cycle of the path and to provide a satisfactory prediction window given target speed. The horizon length should be at least long enough for the aircraft to turn $180^\circ$. It is found that 2 second node spacing works well for the direct collocation method (and the equivalent for the pseudospectral method). However, a 4 second node spacing showed that the approximation accuracy of the cubic polynomials were approaching the limit for producing useful paths. Note that this is directly tied to the speed range and turning radius of the UAV.
In flight demonstrations, the direct collocation method is shown to be able to capture views of the targets in all flight test scenarios, and in most cases maintain a reasonable amount of sensor coverage given the limitations of the particular system tested (a single non-gimbaled camera). In addition, there was great similarity between the simulated and actual flight test results in these scenarios. This shows that despite the simplifying assumptions made (planar UAV motion model with zero pitch angle), the method performs well in actual flight.

Additional flight tests showed that a motion model that predicts future target motion greatly enhances path planning performance. However, the simple straight line target motion assumption did perform well for targets that are slow or not highly agile. Modeling wind effects is just as important as modeling target motion in any path planning application. A moving target and a steady wind have the same effect on the UAV’s path when transformed into the target’s local reference frame. In addition, a steady wind can even sometimes be advantageous when observing a target as it can be used to lessen the speed difference between the UAV and target. This advantage may only be useful for smaller, low flying UAVs such as the one used in this work.

In flight demonstrations, the nonlinear solver SNOPT is generally robust in terms of providing a valid solution. The number of nodes used in the optimization is critical for ensuring real-time operation of a path planner using direct transcription methods. Paths that use fewer nodes may be less optimal, but can be updated more frequently which is important for incorporating changing sensor data. However, reduction in the number of nodes must be accompanied by reduction in horizon time length to maintain accurate dynamics interpolation. This reduction of course reduces target motion anticipation. In addition, care must be taken to provide a smooth objective function in order to enable the nonlinear solver to find a stopping point in as few steps as possible. Switching from the “min-max” to “quadratic” target-in-view objectives provided greatly reduced computation times despite the two function being similar in shape. Although the method can be computationally expensive, especially for larger problem sets, the direct collocation method can be used in a real-time situation given a compromise between the number of optimization parameters and the real-time observation results achieved.

The pseudospectral method is newer than direct collocation and is more accu-
rate. However, it did not out-perform the direct collocation method in all cases. The pseudospectral method will be faster than the direct collocation method given analytical derivatives due to its use of efficient numerical quadrature. However, the direct collocation method could be modified to use a more efficient quadrature algorithm. Because the nodes of the pseudospectral discretization are uneven, the method may show increased accuracy problems in the middle of the path compared to the ends. In contrast, the equal node spacing of the direct collocation method does not have this behavior. Pseudospectral method can cause control limit excursions due to the polynomial interpolation. Therefore, an increased number of nodes may be required to avoid this problem. The linear interpolation scheme, while less accurate, does not have this problem.

6.1.2 Neural Network Approximation Method

The neural network approximation method is shown to be a viable real-time alternative to direct collocation and the pseudospectral methods. Its major advantage is that analytical derivatives are not required for satisfactory real-time performance given similar problem sizes compared to both direct collocation and pseudospectral methods. It is generally 2-5 times faster than the direct collocation method using analytical derivatives and 5+ times faster than the direct collocation and pseudospectral methods using complex-step or automatic derivatives. In addition, it can provide smoothing of discontinuous objective functions which results in additional speedup. Finally, the dynamic collocation requirement is essentially moved to the training process which reduces the number of varied parameters in the nonlinear programming problem.

6.2 Future Work

The C++ version of the path planning algorithm could be integrated with the ARL Intelligent Controller (IC) architecture [35]. This would provide the controller with an autonomous target surveillance behavior. In addition, the IC architecture provides communications between vehicles, making surveillance with multiple UAVs possible. The path planner would need to be modified to operate in a master/slave
fashion where one computer computes the path for all UAVs. Of course, this isn’t
the most efficient method; an alternative may be to have each UAV do it’s own
path planning while treating other UAVs as targets to be avoided (or some method
to produce the desired interaction behavior between the UAVs during target ob-
servation).

Similar applications of the “target-in-view” behavior could be interesting to
investigate. Work by Jones et al. [84] discusses creating a ground path for an on-
board camera’s line of sight by “scribbling” on a map. The path planning methods
presented in this dissertation would easily apply and may offer increased utility
with the ability to natively incorporate constraints other than vehicle performance
limits (such as radar signature, ground visibility, or threat constraints).

Different target-in-view objective functions may reduce computation time. The
currently used quadratic objective function (Equation 4.14) is discontinuous at the
image boundaries which can cause problems during the solution of the nonlinear
programming problem. A function that is first- or second-order continuous at the
image boundaries may reduce solution time. An example sigmoid based function
is given in Equation 6.1.

\[ f(q_x, q_y) = 1 - g(q_x)g(q_y) \]
\[ g(x) = \frac{h(xc + b) + h(-xc + b) - 1}{2h(b) - 1} \]
\[ h(x) = \frac{1}{1 + e^{-x}} \]

where \( c = b + \log(1/d - 1) \). The variable \( d \) moves the “cliff” edge and \( b \) controls the
bucket width. Values of \( d = 0.05 \) and \( b = 7 \) give a reasonable objective function.
Alternatively, a sixth order 2-d polynomial can be second-order continuous at the
image boundaries and the image midpoint.

\[ f(q_x, q_y) = 1 - g(q_x)g(q_y) \]
\[ g(x) = -2x^6 + 3x^4 - 1 \]

Analytical derivatives with the pseudospectral method should be investigated to
make use of the increased quadrature efficiency compared to the direct collocation
method. A Chebyshev or Gauss quadrature method could be applied to the direct
collocation method to produce a less biased comparison between the two methods. In addition, a constant pitch angle should be incorporated in to the target-in-view function to account for aircraft pitch in level flight. From here, a gimbaled camera (time dependent camera orientation) would be useful. Work by Rysdyk [85] on path following with a gimbaled camera may be useful. Additional constraints such as landing gear or empennage obstructions could be added to the optimization to automatically account for various airframes.

A disadvantage of the neural network approximation method is that it currently only works with problems where final time is fixed. In order to be more useful, the method needs to be expanded to accommodate problems with free final time. A method to do this is outlined below. Consider the brachistochrone problem. If the problem is discretized in terms of horizontal position (since both initial and final time are given), the neural network method can be applied. Two different approaches could be taken. The first is to train a network to output $\Delta V_i$, $\Delta y_i$, and $\Delta \theta_i$ that drives the bead from $x_i$ to $x_{i+1}$ given $V_i$, $\theta_i$, and a timestep $dt_i$. Then the varied parameters in the optimization problem are the timesteps required to arrive at the horizontal position at the end of each segment.

The second method is similar, but the network inputs are $V_i$, $\theta_i$, and $\theta_{i+1}$. The outputs are $\Delta V_i$, $\Delta y_i$, and the required timestep to arrive at $x_{i+1}$ from $x_i$. This formulation is closer to the original problem in that the varied parameters are only the controls.

These formulations suffer from the limitation in initial and final horizontal position. If a new final position is chosen, the network needs to be retrained. However, there may be leeway if the dynamics network is trained over a range of $dt$ and thus it could handle arbitrary changes in final positions provided an adequate number of nodes are still used. With this, the number of nodes required would depend on the range of $dt$ that the network is trained with.
Appendix A

Analytical Derivations for the Direct Collocation Method

The analytical derivatives and integrals for the direct collocation method are derived here. These derivatives include both the objective and constraint equations in terms of the hermite approximating polynomials and the equations of motion. The control and distance-to-target analytical integrations are also derived here.

Recall the Hermite interpolating polynomial equations (Equation 2.4)

\[
x_{pi}(s) = [2(x_i - x_{i+1}) + \dot{x}_i + \dot{x}_{i+1}]s^3 + [3(x_{i+1} - x_i) - 2\dot{x}_i - \dot{x}_{i+1}]s^2 + \dot{x}_is + x_i
\]

\[u_{pi}(s) = u_i + (u_{i+1} - u_i)s\]  

(A.2)

where \(s\) is nondimensional segment time, \(s = (t - t_i)/\tau, s \in [0, 1]\). The length of the segment in time is \(\tau\). Note that all dotted (\(\dot{\cdot}\)) variables are multiplied by \(\tau\).

The state equations are repeated below.

\[
\dot{x} = \dot{x}_{u1} = V_t \cos(\psi) - V_{windN}
\]

(A.3)

\[
\dot{y} = \dot{x}_{u2} = V_t \sin(\psi) - V_{windE}
\]

(A.4)

\[
\dot{V}_t = \dot{x}_{u3} = u_t
\]

(A.5)

\[
\dot{\psi} = \dot{x}_{u4} = g \tan(u_\phi)/V_t
\]

(A.6)
\begin{align*}
\dot{x}_t &= \dot{x}_{t_1} = V_{igtN} \tag{A.7} \\
\dot{y}_t &= \dot{x}_{t_2} = V_{igtE} \tag{A.8}
\end{align*}

\section*{A.1 Integrated control cost}

The squared control cost integration is straightforward due to the assumption of linear behavior over the segment. The basic equation for the $i_{th}$ segment is:

\[ \int_0^1 ((u_{i+1} - u_i) s \tau + u_0)^2 = \frac{\tau}{3} (u_i^2 + u_i u_{i+1} + u_{i+1}^2) \tag{A.9} \]

The partial with respect to $u_i$ is then

\[ \frac{\partial}{\partial u_i} = \frac{\tau}{3} (2u_i + u_{i+1}) \tag{A.10} \]

Note that when forming the derivatives over all the segments, each segment derivative depends on both the leading and trailing point (except the endpoints).

\section*{A.2 Integrated distance-to-target cost}

The integration of the distance between the UAV and target over the horizon time is derived here. The squared distance is used to avoid requiring a square root operation. Refer to Section 4.1 for variable names. Note that because a two-dimensional approximation is used, UAV altitude $h$ is constant.

\[ d^2 = d_{xy}^2 + h^2 = (x(t) - x_i(t))^2 + (y(t) - y_i(t))^2 + h^2 \tag{A.11} \]

Since the true states are only tracked at individual nodes, the Hermite interpolating polynomials are used as the continuous time history of the states for the integration. Note that this is the continuous time history over the $i^{th}$ segment only.

\[ x(s)_i = [2(x_i - x_{i+1}) + \dot{x}_i + \dot{x}_{i+1}]s^3 \\
+ [3(x_{i+1} - x_i) - 2\ddot{x}_i - \ddot{x}_{i+1}]s^2 \\
+ \dot{x}_i s + x_i \tag{A.12} \]
Similar equations are used for \( y_1, x_1, \) and \( y_t \). Several intermediate variables are now adopted to simplify notation. These conflict with previously defined notation and are only used in the context of the remainder of this section. For additional simplicity, the \( i \) notation is dropped and the derivation is given for the first segment: \( i = 0 \). Define \( a, b, c, d, e, f, g, h \):

\[
a = 2 \left( \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} - \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \right) + \begin{bmatrix} \dot{x}_0 \\ \dot{y}_0 \end{bmatrix} + \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \end{bmatrix} \tag{A.13}
\]

\[
b = 3 \left( \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} - \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \right) - 2 \begin{bmatrix} \dot{x}_0 \\ \dot{y}_0 \end{bmatrix} - \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \end{bmatrix} \tag{A.14}
\]

\[
c = \begin{bmatrix} \dot{x}_0 \\ \dot{y}_0 \end{bmatrix} \tag{A.15}
\]

\[
d = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{A.16}
\]

\[
e = 2 \left( \begin{bmatrix} x_{t0} \\ y_{t0} \end{bmatrix} - \begin{bmatrix} x_{t1} \\ y_{t1} \end{bmatrix} \right) + \begin{bmatrix} \dot{x}_{t0} \\ \dot{y}_{t0} \end{bmatrix} + \begin{bmatrix} \dot{x}_{t1} \\ \dot{y}_{t1} \end{bmatrix} \tag{A.17}
\]

\[
f = 3 \left( \begin{bmatrix} x_{t1} \\ y_{t1} \end{bmatrix} - \begin{bmatrix} x_{t0} \\ y_{t0} \end{bmatrix} \right) - 2 \begin{bmatrix} \dot{x}_{t0} \\ \dot{y}_{t0} \end{bmatrix} - \begin{bmatrix} \dot{x}_{t1} \\ \dot{y}_{t1} \end{bmatrix} \tag{A.18}
\]

\[
g = \begin{bmatrix} \dot{x}_{t0} \\ \dot{y}_{t0} \end{bmatrix} \tag{A.19}
\]

\[
h = \begin{bmatrix} x_{t0} \\ y_{t0} \end{bmatrix} \tag{A.20}
\]

Performing the appropriate substitutions, converting Equation A.11 to vector form, and integrating results in:

\[
\int_0^1 d^2_{xy} ds = \int_0^1 \left( (a - e)s^3 + (b - f)s^2 + (c - g)s + (d - h) \right)^2 ds \tag{A.21}
\]
After expanding the squared term, integrating over $s \in [0, 1]$, and simplifying the resulting 7th order polynomial, the result is

$$\int_0^1 d_{xy}^2 ds = \frac{1}{7} (a-e)^2 + \frac{1}{3} (a-e)(b-f)$$

$$+ \frac{1}{5} [(b-f)^2 + 2(a-e)(c-g)]$$

$$+ \frac{1}{2} [(b-f)(c-g) + (a-e)(d-h)]$$

$$+ \frac{1}{3} [(c-g)^2 + 2(b-f)(d-h)]$$

$$+ (d-h)(c-g) + (d-h)^2$$

(A.22)

And the final distance to target cost is (here $h$ is altitude):

$$J_{dist} = \sum_{i=0}^{n_{seg}} d_{xy_i}^2 + h^2$$

(A.23)

The gradient of the above integration is now derived. The gradient must be computed for the North and East directions. The following is the general derivative of $J_{dist}$ where $[.]$ is any variable.

$$\frac{\partial J_{dist}}{\partial [.] } = \frac{2}{7} \frac{\partial(a-e)}{\partial [.] } (a-e)$$

$$+ \frac{1}{5} \left[ (b-f) \frac{\partial(a-e)}{\partial [.] } + (a-e) \frac{\partial(b-f)}{\partial [.] } \right]$$

$$+ \frac{2}{3} \left[ (a-e) \frac{\partial(c-g)}{\partial [.] } + (c-g) \frac{\partial(a-e)}{\partial [.] } + (b-f) \frac{\partial(b-f)}{\partial [.] } \right]$$

$$+ \frac{1}{2} \left[ (b-f) \frac{\partial(c-g)}{\partial [.] } + (c-g) \frac{\partial(b-f)}{\partial [.] } + (a-e) \frac{\partial(d-h)}{\partial [.] } + (d-h) \frac{\partial(a-e)}{\partial [.] } \right]$$

$$+ \frac{2}{3} \left[ (c-g) \frac{\partial(c-g)}{\partial [.] } + (b-f) \frac{\partial(d-h)}{\partial [.] } + (d-h) \frac{\partial(b-f)}{\partial [.] } \right]$$

$$+ (d-h) \frac{\partial(c-g)}{\partial [.] } + (c-g) \frac{\partial(d-h)}{\partial [.] }$$

$$+ 2(d-h) \frac{\partial(d-h)}{\partial [.] }$$

(A.24)

The partials with respect to the states and controls are computed below and substituted into Equation A.24. The gradient for one segment is determined by the
\[
\frac{\partial (a-e)}{\partial x} \quad \frac{\partial (b-f)}{\partial v} \quad \frac{\partial (c-g)}{\partial x} \quad \frac{\partial (d-h)}{\partial y}
\]

\[
\begin{array}{cccc}
\frac{\partial a}{\partial x} & 2 & -3 & 0 & 1 \\
\frac{\partial a}{\partial y} & 0 & 0 & 0 & 0 \\
\frac{\partial a}{\partial v} & \tau \cos(\psi) & -2\tau \cos(\psi) & \tau \cos(\psi) & 0 \\
\frac{\partial a}{\partial \psi} & -\tau V_t \sin(\psi) & 2\tau V_t \sin(\psi) & -\tau V_t \sin(\psi) & 0 \\
\frac{\partial a}{\partial x_t} & -2 & 3 & 0 & -1 \\
\frac{\partial a}{\partial y_t} & 0 & 0 & 0 & 0 \\
\end{array}
\]

Table A.1. Leading-point partials for the North direction for \( J_{dist} \)

\[
\frac{\partial (a-e)}{\partial x} \quad \frac{\partial (b-f)}{\partial v} \quad \frac{\partial (c-g)}{\partial x} \quad \frac{\partial (d-h)}{\partial y}
\]

\[
\begin{array}{cccc}
\frac{\partial a}{\partial x} & -2 & 3 & 0 & 0 \\
\frac{\partial a}{\partial y} & 0 & 0 & 0 & 0 \\
\frac{\partial a}{\partial v} & \tau \cos(\psi) & -\tau \cos(\psi) & 0 & 0 \\
\frac{\partial a}{\partial \psi} & -\tau V_t \sin(\psi) & \tau V_t \sin(\psi) & 0 & 0 \\
\frac{\partial a}{\partial x_t} & 2 & -3 & 0 & 0 \\
\frac{\partial a}{\partial y_t} & 0 & 0 & 0 & 0 \\
\end{array}
\]

Table A.2. Trailing-point partials for the North direction for \( J_{dist} \)

sum of the partials at the leading and trailing nodes. Therefore, the following calculation is split into leading and trailing parts. The leading-point partials for the North direction are given in Table A.2 (the actual state and control variables are used here). The segment length in time \( \tau \) is multiplied by all cos and sin functions because the derivative is derived from the dimensional-time state equations. The trailing point partials for the North direction are given in Table A.2. The partials for the East direction \( (y - y_t) \) are given in Table A.2 for the leading nodes and in Table A.2 for the trailing nodes.

A.3 Integrated distance-to-target cost

The distance-to-target cost integral is computed by a simple trapezoidal integration and therefore is not listed in detail. Only the Jacobian of the integration is listed. Throughout this section, the assumption is made that the derivatives are computed
for a particular point along a segment. The chain rule is then used to apply the derivatives to the trapezoidal integration. At any point along the segment, the derivatives are affected by the state and control values and the leading and trailing nodes.

Referring to Equation A.2, we write $x_i = x_{p_{uav}}(s) - x_{p_{tgt}}(s)$ and $y_i = y_{p_{uav}}(s) - y_{p_{tgt}}(s)$. This is the relative North and East position $\mathbf{p}_i$ of the UAV with respect to the target in terms of the approximating polynomials evaluated at $s$. Note that the $z$ component of $\mathbf{p}_i$ is constant due to the assumption of planar UAV motion and can be neglected. Taking partial derivatives of $x_i$ and $y_i$ with respect to the states and controls, we come across several repeated constants. If we are computing the contribution to the derivatives of a point with respect to the leading node, the
Constants are

\[ t_a = (s - 1)^2(1 + 2s) \]
\[ t_b = (s - 1)^2s \] \hspace{1cm} (A.25)
\[ t_c = 1 - s \]

For the point’s trailing node, we have

\[ t_a = s^2(3 - 2s) \]
\[ t_b = s^2(s - 1) \] \hspace{1cm} (A.26)
\[ t_c = s \]

The direction cosine matrix at \( s \) is defined.

\[
R_i = \begin{bmatrix}
c\psi_i & s\psi_i & 0 \\
-cu_\phi_i s\psi_i & cu_\phi_i cu\psi_i & su_\phi_i \\
su_\phi_i s\psi_i & -su_\phi_i cu\psi_i & cu_\phi_i
\end{bmatrix}
\] \hspace{1cm} (A.27)

The gradient of \( p_i \) with respect to the states and controls is

\[
\frac{\partial p_i}{\partial x} = [t_a \ 0 \ 0] \] \hspace{1cm} (A.28)
\[
\frac{\partial p_i}{\partial y} = [0 \ t_a \ 0] \] \hspace{1cm} (A.29)
\[
\frac{\partial p_i}{\partial V_t} = \tau t_b[c\psi \ s\psi \ 0] \] \hspace{1cm} (A.30)
\[
\frac{\partial p_i}{\partial \psi} = \tau t_b V_t[-s\psi \ c\psi \ 0] \] \hspace{1cm} (A.31)
\[
\frac{\partial p_i}{\partial x_t} = [-t_a \ 0 \ 0] \] \hspace{1cm} (A.32)
\[
\frac{\partial p_i}{\partial y_t} = [0 \ -t_a \ 0] \] \hspace{1cm} (A.33)
\[
\frac{\partial p_i}{\partial u_a} = [0 \ 0 \ 0] \] \hspace{1cm} (A.34)
\[
\frac{\partial p_i}{\partial u_\phi} = [0 \ 0 \ 0] \] \hspace{1cm} (A.35)
\[
\frac{\partial p_i}{\partial u_\phi} = [0 \ 0 \ 0] \] \hspace{1cm} (A.36)
Next, the partials of the direction cosine matrix $R$ are given. Note that $R_i$ is computed from the interpolated $\psi_i$ and $u_{\phi_i}$, not the endpoint values. Recall the state equations (Eq. A.8) are present in the approximating polynomials (Eq. A.2) which are used to compute $\psi_i$ and $u_{\phi_i}$. All of these must be accounted for. The final result in matrix form is

$$\frac{\partial R_i}{\partial x} = 0$$  \hspace{1cm} (A.37)

$$\frac{\partial R_i}{\partial y} = 0$$  \hspace{1cm} (A.38)

$$\frac{\partial R_i}{\partial V_t} = -\tau b \frac{d\psi_i}{dR_i} g \tan(\phi)/V_t^2$$  \hspace{1cm} (A.39)

$$\frac{\partial R_i}{\partial \psi} = \frac{du_{\phi_i}}{dR_i} t_c + \frac{d\psi_i}{dR_i} g(1/\cos(\phi))^{2/V_t t_b}$$  \hspace{1cm} (A.40)

$$\frac{\partial R_i}{\partial x_t} = 0$$  \hspace{1cm} (A.41)

$$\frac{\partial R_i}{\partial y_t} = 0$$  \hspace{1cm} (A.42)

$$\frac{\partial R_i}{\partial u_a} = 0$$  \hspace{1cm} (A.43)

$$\frac{\partial R_i}{\partial u_{\phi}} = \frac{d\psi_i}{dR_i} t_a$$  \hspace{1cm} (A.44)

Next, the derivatives of $R_i p_i$ are computed with the chain rule. Take note of the zero valued partials for $R_i$ above.

$$\frac{\partial R_i p_i}{\partial x} = R_i \frac{\partial R_i}{\partial x}$$  \hspace{1cm} (A.46)

$$\frac{\partial R_i p_i}{\partial y} = R_i \frac{\partial R_i}{\partial y}$$  \hspace{1cm} (A.47)

$$\frac{\partial R_i p_i}{\partial V_t} = R_i \frac{\partial R_i}{\partial V_t} + \frac{\partial R_i}{\partial V_t} p_i$$  \hspace{1cm} (A.48)

$$\frac{\partial R_i p_i}{\partial \psi} = R_i \frac{\partial R_i}{\partial \psi} + \frac{\partial R_i}{\partial \psi} p_i$$  \hspace{1cm} (A.49)

$$\frac{\partial R_i p_i}{\partial x_t} = - \frac{\partial R_i p_i}{\partial x}$$  \hspace{1cm} (A.50)

$$\frac{\partial R_i p_i}{\partial y_t} = - \frac{\partial R_i p_i}{\partial y}$$  \hspace{1cm} (A.51)
\[
\frac{\partial R_i p_i}{\partial u_a} = 0 
\]  
(A.52)

\[
\frac{\partial R_i p_i}{\partial u_\phi} = R_i \frac{\partial p_i}{\partial u_\phi} + \frac{\partial R_i}{\partial u_\phi} p_i 
\]  
(A.53)

\[
\frac{\partial R_i p_i}{\partial u_\phi} = R_i \frac{\partial p_i}{\partial u_\phi} + \frac{\partial R_i}{\partial u_\phi} p_i 
\]  
(A.54)

Now the derivatives for \( T \) are computed (recall \( T \) is defined in Equation 4.9).

\[
\frac{\partial T_i}{\partial x} = \begin{bmatrix} 0_{3 \times 2} & -\frac{\partial R_i p_i}{\partial x} \end{bmatrix} 
\]  
(A.55)

\[
\frac{\partial T_i}{\partial y} = \begin{bmatrix} 0_{3 \times 2} & -\frac{\partial R_i p_i}{\partial y} \end{bmatrix} 
\]  
(A.56)

\[
\frac{\partial T_i}{\partial V_t} = \begin{bmatrix} \frac{\partial R_i}{\partial V_t} \left[ ; :1:2 \right] & -\frac{\partial R_i p_i}{\partial V_t} \end{bmatrix} 
\]  
(A.57)

\[
\frac{\partial T_i}{\partial \psi} = \begin{bmatrix} \frac{\partial R_i}{\partial \psi} \left[ ; :1:2 \right] & -\frac{\partial R_i p_i}{\partial \psi} \end{bmatrix} 
\]  
(A.58)

\[
\frac{\partial T_i}{\partial x_t} = \begin{bmatrix} 0_{3 \times 2} & -\frac{\partial R_i p_i}{\partial x_t} \end{bmatrix} 
\]  
(A.59)

\[
\frac{\partial T_i}{\partial y_t} = \begin{bmatrix} 0_{3 \times 2} & -\frac{\partial R_i p_i}{\partial y_t} \end{bmatrix} 
\]  
(A.60)

\[
\frac{\partial T_i}{\partial u_a} = 0 
\]  
(A.61)

\[
\frac{\partial T_i}{\partial u_\phi} = \begin{bmatrix} \frac{\partial R_i}{\partial u_\phi} \left[ ; :1:2 \right] & \frac{\partial R_i p_i}{\partial u_\phi} \end{bmatrix} 
\]  
(A.62)

(A.63)

Note that \( \frac{\partial R_i}{\partial V_t} \left[ ; :1:2 \right] \) means to extract all of the rows and the first two columns. Following the homography computation in Equation 4.10, the derivatives of \( K C_m T_i p_i \) are computed.

\[
\frac{\partial K C_m T_i p_i}{\partial x} = K C_m \left( T_i \frac{\partial p_i}{\partial x} + \frac{\partial T_i}{\partial x} p_i \right) 
\]  
(A.64)

\[
\frac{\partial K C_m T_i p_i}{\partial y} = K C_m \left( T_i \frac{\partial p_i}{\partial y} + \frac{\partial T_i}{\partial y} p_i \right) 
\]  
(A.65)

\[
\frac{\partial K C_m T_i p_i}{\partial V_t} = K C_m \left( T_i \frac{\partial p_i}{\partial V_t} + \frac{\partial T_i}{\partial V_t} p_i \right) 
\]  
(A.66)

\[
\frac{\partial K C_m T_i p_i}{\partial \psi} = K C_m \left( T_i \frac{\partial p_i}{\partial \psi} + \frac{\partial T_i}{\partial \psi} p_i \right) 
\]  
(A.67)
\[ \frac{\partial KCT_p_i}{\partial x_t} = KC_m \left( T_i \frac{\partial p_i}{\partial x_t} + \frac{\partial T_i}{\partial x_t} p_i \right) \] (A.68)

\[ \frac{\partial KCT_p_i}{\partial y_t} = KC_m \left( T_i \frac{\partial p_i}{\partial y_t} + \frac{\partial T_i}{\partial y_t} p_i \right) \] (A.69)

\[ \frac{\partial KCT_p_i}{\partial u_a} = 0 \] (A.70)

\[ \frac{\partial KCT_p_i}{\partial u_\phi} = KC_m \left( T_i \frac{\partial p_i}{\partial u_\phi} + \frac{\partial T_i}{\partial u_\phi} p_i \right) \] (A.71)

The gradient matrix for the unnormalized pixel location derivatives \( \nabla DP \), is a row vector assembled from above eight equations. Continuing with Equation 4.11 and using the quotient rule, the normalized pixel location derivatives are given by

\[ \nabla P = \nabla DP \frac{1}{\bar{q}_z} \left( \bar{q}_x \quad \bar{q}_y \right) \nabla DP \left( \bar{q}_z \right) \] (A.73)

where \( \bar{q} = [\bar{q}_x \quad \bar{q}_y \quad \bar{q}_z]^T \) is defined in Equation 4.10. Now the final gradient is assembled. This step applies the chain rule to compute the effect of the pixel location objective function on the pixel location derivatives \( J_{tiv} = min(q \cdot q, 1) \).

\[ \nabla J_{tiv} P = 2 \nabla P^T q \] (A.74)

This final gradient is multiplied by the target-in-view weight, \( \tau \), and a factor of 0.5 if it is computed directly at a node. It is then summed appropriately with the overall gradient.

### A.4 Constraint Derivatives

The collocation constraints for the direct collocation method are given by the difference between the state equations evaluated at the midpoint of a segment \( x_{ci} \) and the derivative of the Hermite cubic polynomial at the midpoint. The relevant equations are as follows (Refer to Section 2.2).

\[ x_{ci} = x_{pi}(0.5) = \frac{1}{2}(x_i + x_{i+1}) + \frac{\tau}{8}(\dot{x}_i - \dot{x}_{i+1}) \] (A.75)
The constraint derivatives at the $i^{th}$ segment are now listed. Several simplifying constants are given first.

\[
\begin{align*}
\frac{dt}{32Q} & = [1.5 \quad -1.5] \quad (A.78) \\
csQ & = \frac{1}{2}(\psi_i + \psi_{i+1}) + \frac{\tau}{8}(\psi_i - \psi_{i+1}) \quad (A.79) \\
c_{csQ} & = \cos(csQ) \\
s_{csQ} & = \sin(csQ) \\
vtQ & = \frac{1}{2}(V_{ti} + V_{ti+1}) + \frac{\tau}{8}(V_{ti} - V_{ti+1}) \quad (A.82) \\
t_{\phi V1} & = \tan(u_{\phi i})/V_i^2 \quad (A.83) \\
t_{\phi V2} & = \tan(u_{\phi i+1})/V_{i+1}^2 \quad (A.84) \\
s_{\phi V1} & = \sec(u_{\phi i+1})/V_{i+1} \quad (A.85) \\
s_{\phi V2} & = \sec(u_{\phi i+1})/V_{i+1} \quad (A.86)
\end{align*}
\]

And now the derivatives:

\[
\begin{align*}
X & = W_{def[1]} \begin{bmatrix}
\frac{c_{csQ}}{2} + \frac{\tau}{8}vtQs_{csQ} & \frac{dt32Q}{dt} \\
-\frac{1}{2}s_{csQ}vtQ & -\frac{1}{4}Vti\sin(\psi_i) - \frac{1}{4}V_{ti+1}\sin(\psi_{i+1})
\end{bmatrix} \quad (A.87) \\
X_u & = W_{def[1]} \begin{bmatrix}
\frac{\tau}{8}c_{csQ} & 1 & -1 \\
\frac{\tau}{8}s_{csQ}vtQ & -s_{\phi V1} & s_{\phi V2}
\end{bmatrix} \quad (A.88) \\
Y & = W_{def[2]} \begin{bmatrix}
\frac{s_{csQ}}{2} + \frac{\tau}{8}vtQc_{csQ} & -t_{\phi V1} & t_{\phi V2} + \frac{1}{4}[\sin(V_{ti}) \quad \sin(V_{ti+1})] \\
-\frac{1}{2}c_{csQ}vtQ & -\frac{1}{4}Vti\cos(\psi_i) - \frac{1}{4}V_{ti+1}\cos(\psi_{i+1})
\end{bmatrix} \quad (A.89) \\
Y_u & = W_{def[2]} \begin{bmatrix}
\frac{\tau}{8}c_{csQ} & 1 & -1 \\
\frac{\tau}{8}s_{csQ}vtQ & s_{\phi V1} & -s_{\phi V2}
\end{bmatrix} \quad (A.90) \\
V & = W_{def[3]} \begin{bmatrix}
\frac{dt32Q}{dt}
\end{bmatrix} \quad (A.91)
\end{align*}
\]
\[ V_u = W_{def}[3] \begin{bmatrix} \frac{3}{4} & \frac{3}{4} \end{bmatrix} \]  
(A.92)

\[ \Psi = W_{def}[4] \begin{bmatrix} -\frac{g}{2} \tan(u_{\phi_c})/vt_q^2 - \frac{g}{3} \begin{bmatrix} t_{\phi V_1} & t_{\phi V_2} \end{bmatrix} \end{bmatrix} dt3_{2Q} \]  
(A.93)

\[ \Psi_u = W_{def}[4] \begin{bmatrix} \frac{g^x}{8} \tan(u_{\phi_c})/vt_q^2 - \frac{1}{2} \begin{bmatrix} -1 & 1 \end{bmatrix} - \frac{g}{4} \begin{bmatrix} s_{\phi V_1} & s_{\phi V_2} \end{bmatrix} dt3_{2Q} \end{bmatrix} \]  
(A.94)

\[ X_t = W_{def}[5] dt3_{2Q} \]  
(A.95)

\[ Y_t = W_{def}[6] dt3_{2Q} \]  
(A.96)

The constraint derivatives for the \( i^{th} \) segment are then assembled columnwise in the order they appear above into the \( i^{th} \) column of the constraint Jacobian.
Appendix B

Neural Network Weights, Biases, and Scaling for the UAV Surveillance Problem

The weights and biases of the three networks used in the UAV surveillance problem are examined here. Their numerical values are tabulated and the corresponding Hinton graph is shown with each table. The Hinton graph shows the relative magnitudes of the weights and biases in a graphical format. Positive weights are green and negative weights are red. The bias column is highlighted in blue. The Hinton graph is created by calling the MATLAB command `hintonwb(W,b)` with the corresponding weight matrix and bias vector. Each network has three layers with 15 nodes in the input and hidden layers and the appropriate number of nodes in the output layer.

B.1 Dynamics Network

The weights and biases for the dynamics network are shown in Tables B.1, B.2, and B.3. The Hinton graph for the input layer, shown in Figure B.1 shows that the weight and bias magnitudes are relatively well distributed. Layer 2 and layer 3, shown in Figures B.2 and B.3 respectively, show similar distribution. However, looking more closely at layer 2, one can see that input 11 does not greatly contribute to the layer’s output. Thus, a neuron can probably be removed from the input
layer.

**Figure B.1.** Dynamics network, Layer 1

**Figure B.2.** Dynamics network, Layer 2
Figure B.3. Dynamics network, Layer 3
<table>
<thead>
<tr>
<th>Biases</th>
<th>Weights</th>
<th>Biases</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.4810</td>
<td>0.18845</td>
<td>-0.31462</td>
<td>0.19237</td>
</tr>
<tr>
<td>1.43986</td>
<td>0.18272</td>
<td>-0.55586</td>
<td>0.4708</td>
</tr>
<tr>
<td>-0.28889</td>
<td>0.19327</td>
<td>0.03227</td>
<td>0.02670</td>
</tr>
<tr>
<td>1.53709</td>
<td>0.21142</td>
<td>-1.39604</td>
<td>0.21491</td>
</tr>
<tr>
<td>-0.87871</td>
<td>-0.20908</td>
<td>-0.23787</td>
<td>0.34031</td>
</tr>
<tr>
<td>-0.31462</td>
<td>-0.88822</td>
<td>0.40643</td>
<td>-0.02323</td>
</tr>
<tr>
<td>-1.34134</td>
<td>-0.31077</td>
<td>-0.01513</td>
<td>-0.04232</td>
</tr>
<tr>
<td>-1.59353</td>
<td>-0.53869</td>
<td>-0.03927</td>
<td>-0.21229</td>
</tr>
<tr>
<td>-0.99142</td>
<td>-0.91243</td>
<td>1.17284</td>
<td>-0.03470</td>
</tr>
<tr>
<td>-1.57797</td>
<td>0.17731</td>
<td>0.72667</td>
<td>0.21258</td>
</tr>
<tr>
<td>-0.00044</td>
<td>0.25454</td>
<td>0.02667</td>
<td>0.15077</td>
</tr>
<tr>
<td>0.41819</td>
<td>0.08802</td>
<td>0.94363</td>
<td>0.02334</td>
</tr>
<tr>
<td>-0.00742</td>
<td>-0.07512</td>
<td>1.06206</td>
<td>-0.01815</td>
</tr>
<tr>
<td>-0.53649</td>
<td>-0.26647</td>
<td>0.10941</td>
<td>-0.03349</td>
</tr>
</tbody>
</table>

Table B.1. Dynamics network, Layer 1

<table>
<thead>
<tr>
<th>Biases</th>
<th>Weights</th>
<th>Biases</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.50102</td>
<td>0.00037</td>
<td>1.35753</td>
<td>-0.00327</td>
</tr>
<tr>
<td>3.03539</td>
<td>0.60207</td>
<td>0.04277</td>
<td>-0.07210</td>
</tr>
<tr>
<td>1.77550</td>
<td>1.33687</td>
<td>-0.04886</td>
<td>-0.70583</td>
</tr>
<tr>
<td>-0.03442</td>
<td>-2.23609</td>
<td>1.66097</td>
<td>-2.95015</td>
</tr>
<tr>
<td>-0.53254</td>
<td>-0.24158</td>
<td>0.39092</td>
<td>0.39196</td>
</tr>
<tr>
<td>1.79851</td>
<td>1.24523</td>
<td>0.02361</td>
<td>0.39985</td>
</tr>
<tr>
<td>-6.91408</td>
<td>-3.21429</td>
<td>1.71875</td>
<td>-0.95508</td>
</tr>
<tr>
<td>3.10569</td>
<td>0.88986</td>
<td>-0.11442</td>
<td>0.42518</td>
</tr>
<tr>
<td>1.37960</td>
<td>2.12087</td>
<td>-0.04022</td>
<td>0.12843</td>
</tr>
<tr>
<td>0.70476</td>
<td>0.55308</td>
<td>0.29114</td>
<td>-0.32320</td>
</tr>
<tr>
<td>-0.63388</td>
<td>0.73932</td>
<td>0.30276</td>
<td>-0.40271</td>
</tr>
<tr>
<td>1.65162</td>
<td>1.21048</td>
<td>1.54800</td>
<td>-0.01519</td>
</tr>
<tr>
<td>-1.49425</td>
<td>-0.75307</td>
<td>-0.21085</td>
<td>-1.00340</td>
</tr>
<tr>
<td>2.93380</td>
<td>3.80240</td>
<td>0.13950</td>
<td>0.52083</td>
</tr>
</tbody>
</table>

Table B.2. Dynamics network, Layer 2

<table>
<thead>
<tr>
<th>Biases</th>
<th>Weights</th>
<th>Biases</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.95767</td>
<td>-0.30102</td>
<td>-1.01927</td>
<td>2.13190</td>
</tr>
<tr>
<td>0.71020</td>
<td>0.25315</td>
<td>-1.97720</td>
<td>1.65693</td>
</tr>
<tr>
<td>0.86710</td>
<td>3.68301</td>
<td>0.01278</td>
<td>-0.09495</td>
</tr>
<tr>
<td>0.40336</td>
<td>0.17874</td>
<td>-0.22173</td>
<td>1.00707</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.01918</td>
<td>-0.03023</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.63627</td>
<td>-1.26650</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.31901</td>
<td>-0.03113</td>
</tr>
</tbody>
</table>

Table B.3. Dynamics network, Layer 3

---

123
B.2 Distance-to-target Network

The weights and biases for the distance-to-target objective function network are shown in Tables B.4, B.5, and B.6. The Hinton graphs for each layer are shown in Figures B.4, B.5, and B.6. The graphs show that the number of neurons for the distance-to-target objective function can likely be reduced. Looking at layers 2 and 3, the network can probably be reduced by 6 neurons in both layers.

Figure B.4. Distance-to-target network, Layer 1

Figure B.5. Distance-to-target network, Layer 2
Figure B.6. Distance-to-target network, Layer 3
### Table B.4. Distance-to-target network, Layer 1

<table>
<thead>
<tr>
<th>Biases</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.21064</td>
<td>0.01021</td>
</tr>
<tr>
<td>0.64745</td>
<td>-0.00489</td>
</tr>
<tr>
<td>-0.85100</td>
<td>0.01717</td>
</tr>
<tr>
<td>-0.82760</td>
<td>-0.01757</td>
</tr>
<tr>
<td>-0.80905</td>
<td>-0.07965</td>
</tr>
<tr>
<td>0.73015</td>
<td>0.93640</td>
</tr>
<tr>
<td>-0.79105</td>
<td>0.23476</td>
</tr>
<tr>
<td>-0.89702</td>
<td>0.10577</td>
</tr>
<tr>
<td>0.19690</td>
<td>-0.10993</td>
</tr>
<tr>
<td>0.98365</td>
<td>-0.12803</td>
</tr>
<tr>
<td>0.19887</td>
<td>-0.00482</td>
</tr>
<tr>
<td>-1.02782</td>
<td>0.01179</td>
</tr>
<tr>
<td>0.06302</td>
<td>0.07590</td>
</tr>
<tr>
<td>-3.07710</td>
<td>-0.18887</td>
</tr>
</tbody>
</table>

### Table B.5. Distance-to-target network, Layer 2

<table>
<thead>
<tr>
<th>Biases</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.02654</td>
<td>6.21419</td>
</tr>
<tr>
<td>0.00009</td>
<td>-0.53727</td>
</tr>
<tr>
<td>-2.10280</td>
<td>-0.60518</td>
</tr>
<tr>
<td>-2.03867</td>
<td>-1.08461</td>
</tr>
<tr>
<td>-3.10806</td>
<td>2.45724</td>
</tr>
<tr>
<td>-14.4555</td>
<td>3.35317</td>
</tr>
<tr>
<td>9.86047</td>
<td>-7.52877</td>
</tr>
<tr>
<td>-2.15152</td>
<td>0.81064</td>
</tr>
<tr>
<td>-2.19017</td>
<td>-0.98367</td>
</tr>
<tr>
<td>-2.54448</td>
<td>-0.91827</td>
</tr>
<tr>
<td>-9.42125</td>
<td>-9.28507</td>
</tr>
<tr>
<td>4.97833</td>
<td>-4.62784</td>
</tr>
<tr>
<td>-0.84836</td>
<td>-1.08546</td>
</tr>
<tr>
<td>-2.96803</td>
<td>3.31911</td>
</tr>
<tr>
<td>2.73213</td>
<td>3.38774</td>
</tr>
</tbody>
</table>

### Table B.6. Distance-to-target network, Layer 3

<table>
<thead>
<tr>
<th>Biases</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.85086</td>
<td>-0.00077</td>
</tr>
<tr>
<td>-0.76264</td>
<td>-3.13470</td>
</tr>
<tr>
<td>0.00021</td>
<td>0.00006</td>
</tr>
<tr>
<td>-0.23034</td>
<td>-2.78657</td>
</tr>
<tr>
<td>0.77351</td>
<td>-0.05751</td>
</tr>
<tr>
<td>0.00032</td>
<td>0.00020</td>
</tr>
<tr>
<td>-2.89532</td>
<td>-0.09012</td>
</tr>
<tr>
<td>0.60077</td>
<td>-0.23087</td>
</tr>
</tbody>
</table>

126
B.3 Target-in-view Network

The weights and biases for the target-in-view objective function network are shown in Tables B.7, B.8, and B.9. The Hinton graphs for each layer are shown in Figures B.7, B.8, and B.9. This network shows good distribution of weight and bias magnitudes except for inputs 1, 5, and 7, which correspond to $V_t$, $u_{a_0}$, and $u_{a_1}$. This suggests that these inputs could be removed from the network without affecting network performance. This seems to be an intuitive result, since airspeed and acceleration command would have a weak effect on the target-in-view objective function value given the short time step and relatively low speed of the UAV. Note that this only likely applies if the time step is kept low (this network is trained over 2 seconds) and the speed capabilities of the UAV are low.

![Figure B.7. Target-in-view network, Layer 1](image-url)
Figure B.8. Target-in-view network, Layer 2

Figure B.9. Target-in-view network, Layer 3
### Table B.7. Target-in-view network, Layer 1

<table>
<thead>
<tr>
<th>Biases</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2.9429</td>
<td>2.97483</td>
</tr>
<tr>
<td>-0.6235</td>
<td>2.26648</td>
</tr>
<tr>
<td>-0.1047</td>
<td>-0.95045</td>
</tr>
<tr>
<td>-1.07835</td>
<td>1.17543</td>
</tr>
<tr>
<td>-0.30691</td>
<td>-0.54470</td>
</tr>
<tr>
<td>0.47374</td>
<td>1.56403</td>
</tr>
<tr>
<td>-0.59330</td>
<td>-1.65817</td>
</tr>
<tr>
<td>0.23616</td>
<td>0.57911</td>
</tr>
<tr>
<td>0.34004</td>
<td>2.38151</td>
</tr>
<tr>
<td>-1.92013</td>
<td>-0.10430</td>
</tr>
<tr>
<td>0.69557</td>
<td>0.00385</td>
</tr>
<tr>
<td>0.30375</td>
<td>0.00930</td>
</tr>
<tr>
<td>0.45142</td>
<td>-0.26904</td>
</tr>
<tr>
<td>-1.23704</td>
<td>-0.00785</td>
</tr>
<tr>
<td>-0.82476</td>
<td>-0.85710</td>
</tr>
</tbody>
</table>

### Table B.8. Target-in-view network, Layer 2

<table>
<thead>
<tr>
<th>Biases</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.32225</td>
<td>-4.0886</td>
</tr>
<tr>
<td>-1.36883</td>
<td>2.39285</td>
</tr>
<tr>
<td>-0.21487</td>
<td>0.04106</td>
</tr>
<tr>
<td>-0.89811</td>
<td>-1.26327</td>
</tr>
<tr>
<td>3.72743</td>
<td>-0.89998</td>
</tr>
<tr>
<td>0.12494</td>
<td>0.80481</td>
</tr>
<tr>
<td>-2.24378</td>
<td>-0.12379</td>
</tr>
<tr>
<td>0.05799</td>
<td>-0.50340</td>
</tr>
<tr>
<td>0.87585</td>
<td>-0.90061</td>
</tr>
<tr>
<td>-3.90579</td>
<td>0.36791</td>
</tr>
<tr>
<td>1.92545</td>
<td>0.60379</td>
</tr>
<tr>
<td>0.57206</td>
<td>1.15862</td>
</tr>
<tr>
<td>-3.70679</td>
<td>0.77028</td>
</tr>
<tr>
<td>-0.17863</td>
<td>-0.39211</td>
</tr>
</tbody>
</table>

### Table B.9. Target-in-view network, Layer 3

<table>
<thead>
<tr>
<th>Biases</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.4634</td>
<td>-10.0144</td>
</tr>
<tr>
<td>-13.0326</td>
<td>-22.9967</td>
</tr>
<tr>
<td>-10.7190</td>
<td>5.51253</td>
</tr>
<tr>
<td>21.2717</td>
<td>45.52562</td>
</tr>
<tr>
<td>24.5277</td>
<td>-6.29022</td>
</tr>
<tr>
<td>18.6734</td>
<td>12.8679</td>
</tr>
<tr>
<td>3.76143</td>
<td>-22.5209</td>
</tr>
</tbody>
</table>
B.4 Scaling

The networks are scaled so that inputs and outputs range from -1 to 1. The scaling parameters are presented below. Inputs are scaled according to Equation B.1.

\[ y = m_i x + b_i \]  

(B.1)

Outputs are scaled according to Equation B.2.

\[ y = (x - b_o)/m_o \]  

(B.2)

Note that \( m_i x \) is an element-wise multiplication (the .\* operator in MATLAB).

The scaling parameters for the inputs and outputs for the dynamics network are given below.

\[
m_i = \begin{bmatrix} 0.03921569 \\ 0.31830989 \\ 0.1 \\ 1.90985932 \\ 0.1 \\ 1.90985932 \end{bmatrix}, \quad b_i = \begin{bmatrix} -2.4509804 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \]  

(B.3)

\[
m_o = \begin{bmatrix} 0.005540884 \\ 0.005550518 \\ 0.039221038 \\ 0.257599148 \end{bmatrix}, \quad b_o = \begin{bmatrix} 0.005350524 \\ -0.005409400 \\ -2.451184862 \\ 0.006443062 \end{bmatrix} \]  

(B.4)
For the distance-to-target network, the scaling parameters are given below.

\[
\begin{bmatrix}
0.039215686 \\
0.318309886 \\
0.001 \\
0.001 \\
0.1 \\
1.909859317 \\
0.1 \\
1.909859317 \\
\end{bmatrix}
\quad b_i =
\begin{bmatrix}
-2.4509804 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\]  
(B.5)

\[
m_o = \begin{bmatrix} 4.601430E - 007 \end{bmatrix} \quad b_0 = \begin{bmatrix} -1.1482193 \end{bmatrix}
\]  
(B.6)

For the target-in-view objective function network, the scaling parameters are given below. The input parameters are the same as the distance-to-target network.

\[
\begin{bmatrix}
0.039215686 \\
0.318309886 \\
0.001 \\
0.001 \\
0.1 \\
1.909859317 \\
0.1 \\
1.909859317 \\
\end{bmatrix}
\quad b_i =
\begin{bmatrix}
-2.4509804 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\]  
(B.7)

\[
m_o = \begin{bmatrix} 0.537292446 \end{bmatrix} \quad b_0 = \begin{bmatrix} -0.074584891 \end{bmatrix}
\]  
(B.8)
Bibliography


Vita
Brian Geiger

Brian Geiger was born on July 15, 1980 and was raised in Pass Christian, Mississippi. He started at Mississippi State University in 1998 first as an Electrical Engineering student but switched to Aerospace Engineering after a few short weeks. After his undergraduate studies were completed, he went to the Pennsylvania State University in 2003 to pursue graduate studies under Dr. Joseph Horn at the Rotorcraft Center of Excellence. He obtained his Master of Science (MS) degree in May of 2005 on redundant control optimization for compound helicopters. Continuing at Penn State, he completed his Ph.D. on trajectory optimization for unmanned aerial vehicles in 2009.