Real-Time Maneuvering Decisions for Autonomous Air Combat

by

James S. McGrew

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2008

Author .....
Department of Aeronautics and Astronautics
May 23, 2008

Certified by .....
Jonathan How
Professor
Thesis Supervisor

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

Real-Time Maneuvering Decisions for Autonomous Air Combat

by

James S. McGrew

Submitted to the Department of Aeronautics and Astronautics
on May 23, 2008, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Unmanned Aircraft Systems (UAS) have the potential to perform many of the complex and possibly dangerous missions currently flown by manned aircraft. Within visual range air combat is an extremely difficult and dynamic aerial task which presents many challenges for an autonomous UAS. An agile, unpredictable, and possibly human-piloted adversary, coupled with a complex and rapidly changing environment, creates a problem that is difficult to model and solve. This thesis presents a method for formulating and solving a function approximation dynamic program to provide maneuvering decisions for autonomous one-on-one air combat. Value iteration techniques are used to compute a function approximation representing the solution to the dynamic program. The function approximation is then used as a maneuvering policy for UAS autonomous air combat. The result is an algorithm capable of learning a maneuvering policy and utilizing this policy to make air combat decisions in real-time. Simulation results are presented which demonstrate the robustness of the method against an opponent beginning from both offensive and defensive situations. The results also demonstrate the ability of the algorithm to learn to exploit an opponent's maneuvering strategy. Flight results are presented from implementation on micro-UAS flown at MIT's Real-time indoor Autonomous Vehicle test ENvironment (RAVEN) demonstrating the practical application of the proposed solution in real-time flight with actual aircraft.

Thesis Supervisor: Jonathan How
Title: Professor

Acknowledgments

I would like to thank my advisor, Professor Jonathan How, for his support and guidance throughout this project. I am extremely grateful to have had the opportunity to pursue an area of research that I am most interested in. Thanks to my friend, Larry Bush, for his invaluable assistance with the project. The Boeing Company was generous enough to allow my use of their Vehicle Swarm Technology Lab in Seattle, WA for flight testing; I am thankful for this opportunity as it was very extremely beneficial to the project. I also would like to thank Professor Mark Drela for his assistance with airplane design, as well as Professor Missy Cummings for her helpful guidance. I appreciate the help of my colleagues, Dr. Mario Valenti, Brett Bethke, Dan Dale, Spencer Ahrens, Frank Fan, Buddy Michini, Brandon Luders, Ray He and Luc Brunet. It was a pleasure to work in the Aerospace Controls Laboratory with such talented individuals. I appreciate the assistance and hard work of Eli Cohen, Jason Wallace and Brittany Baker. Their help as UROPs allowed me to accomplish much more on this project than I could have alone. I gratefully acknowledge the American Society for Engineering Education for funding my education through the National Defense Science and Engineering Graduate Fellowship. AFOSR also supported this research in part with DURIP grant FA9550-07-1-0321. And, finally, I would like to thank my wife, Eden, for her patience and support during the past years as I put in the long days and nights required to complete this work.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Unmanned Aircraft Systems (UAS) have been successful in replacing manned aircraft in a variety of commercial and military aerial missions. However, due to the challenging and dynamic nature of air-to-air combat, these missions are still solely accomplished by manned platforms. An agile, unpredictable, and possibly human-piloted adversary, coupled with a complex and rapidly changing environment, creates a problem that is difficult to model and solve. One approach to using Unmanned Aircraft (UAs) for air combat is to pilot them remotely, as was first accomplished by an MQ-1 Predator UAS in 2002 [18]. This approach still requires a one-to-one pilot-to-aircraft ratio and would not necessarily leverage the particular strengths of a combat UAS. By automating some of the decisions required in air combat, an operator could potentially maximize vehicle performance and manage multiple UAs in combat simultaneously.

While weapon and missile technology improves, air combat continues to extend to longer and longer ranges. However, employment of long range and high off bore-sight weapons is not always practical or possible depending on the situation. Consequently, the most modern fighter aircraft (*e.g.*, F/A-22, F-35, and F-15) are still built with a gun designed for close combat and military pilots are trained to effectively use in one-on-one air combat, or basic fighter maneuvering (BFM). If a UAS is ever going to fulfill the air combat missions performed by these manned aircraft it is my opinion

that the the ability to effectively fly BFM will be a requirement. [1]

The purpose of this research is to develop a method that allows for on-line computation of near-optimal maneuvering decisions for a UAS engaged in BFM. As is common in human flight training, an incremental approach is taken to develop this BFM method. This work focuses on solving a simplified air combat problem. The velocity for both aircraft is set to a constant value, and both aircraft are constrained to maintain level fight. These restrictions emphasize the need to use maneuvering geometry to solve the BFM problems and help to reduce the search space. As such, the proposed planning methods optimize a 2-dimensional (horizontal plane) flight trajectory with fixed aircraft velocities. Combat simulation results are presented showing the capabilities of the planning techniques. Additionally, actual micro-UAS flight results are presented using Real-time indoor Autonomous Vehicle test ENvironment (RAVEN) [24]. The next step in the development of a more capable planning algorithm would incorporate velocity control and 3-dimensional trajectories using similar methods. This extension is beyond the scope of this thesis, and will be explored in future works.

1.1 Literature Review

Modeling and solving air combat games has been the subject of previous research utilizing various techniques. The use of dynamic programming, as applied to air combat, however, has been limited due to the computational complexity of formulating and solving such an approach. The optimal solution to the pursuer-evader model was first defined in [10]. The defining conditions and approach were subsequently named the principle of optimality and dynamic programming respectively [4]. Yet methods found in the literature tend to ignore these principles for computational expediency.

Burgin and Sidor developed an Adaptive Maneuvering Logic Program in [7]. The authors developed a rule-based air combat simulation. This method requires hard

[1] The author is a former U.S. Air Force F-15C Eagle and MQ-1B Predator UAS pilot with training and experience in air-to-air and UAS combat missions.

coding the preferences of experienced pilots into a maneuver selection algorithm. The authors noted that while this method was capable of operating successfully in simulation with human pilot adversaries, it was extremely time consuming to improve the tactical performance. They commented on the extreme complexity of real-life air-to-air combat and the importance of algorithm evaluation with highly-skilled human pilots. The difficulty with such a rule based approach is the effort and time required to manually evaluate and adjust the maneuver selection parameters. Of course, the development process would need to be repeated for application on any vehicle with different performance characteristics than those originally considered.

Austin et al. suggest using a game theoretic approach in [2, 3]. This method uses a recursive search over discrete maneuver choices to maximize a heuristic scoring function with a fixed planning horizon. Using this approach, they demonstrate the feasibility of real-time autonomous combat in simulation. However, as presented, the method is computationally complex and can only be extended to relatively short planning horizons in real-time (0.5 to 3.0 s). They demonstrated that the method was capable of producing some on-line maneuvering decisions which are similar to those made by experienced human pilots; note this was accomplished without specifically coding such maneuvers. However, the authors state that the selected maneuver is optimal only in the short-term sense and may not be optimal if the game is played out to its end. Additionally, this method is only optimal in relation to the chosen heuristic scoring function. A nonlinear model predictive tracking controller (NMPTC) is demonstrated in [12, 22], which presents a real-time implementation of a evasion controller game involving fixed wing aircraft. The demonstrated algorithm did not have the ability to switch between pursuit and evasion roles. The authors also comment on the need to encode proven aircraft maneuvering tactics from [21] into the cost functions used for the optimization, in order to encourage these behaviors. The requirement of such input in the development phase indicates the need to hard code desired maneuvers. The method presented is not shown to have the ability compute such desirable maneuvers. Virtanen et al. utilize an influence diagram game to model air combat [13]. The authors mention the combinatorial explosion in computation

attributed to solving the proposed game using dynamic programming. They circumvent this problem by using a moving horizon control approach. The planning horizon is limited by computational complexity, but they demonstrate plausible and desirable control choices with the relatively short planning horizons required for real-time computation.

1.1.1 Objectives

This thesis describes an algorithm which attempts to utilize the benefits of dynamic programming (DP) introduced in [4] to solve an air combat game. The goal is to develop a method that has the following desirable qualities: Is Capable of making maneuvering decisions on-line in real-time, incorporates a long planning horizon, has the ability to learn control sequences of desirable maneuvers without expert pilot inputs, and allows switching from pursuit to evasion rolls during an engagement. DP has the potential to produce such maneuvering policies. While an exact DP solution is intractable for a complex game such as air combat, an approximate architecture is capable of producing good results in a finite time. A neuro-dynamic programming approach [6] is used to learn a value function which represents an approximation of the true DP solution. This technique requires some computation time prior to the engagement in which to learn the maneuvering policy based on the adversary aircraft's capabilities and assumed adversary maneuvering strategy. The resulting policy can then be used on-line to make real-time decision making during combat. An additional goal of this research is to demonstrate successful planning algorithm implementation using fixed wing UAS flying in MIT's RAVEN.

1.2 Overview of MIT RAVEN Research Platform

The MIT Real-time indoor Autonomous Vehicle test ENvironment (RAVEN) was used as the hardware testbed for this research. A brief overview of RAVEN is presented here; for more details see [15, 16].

The core of the testbed consists of a Vicon motion capture system [25] that is

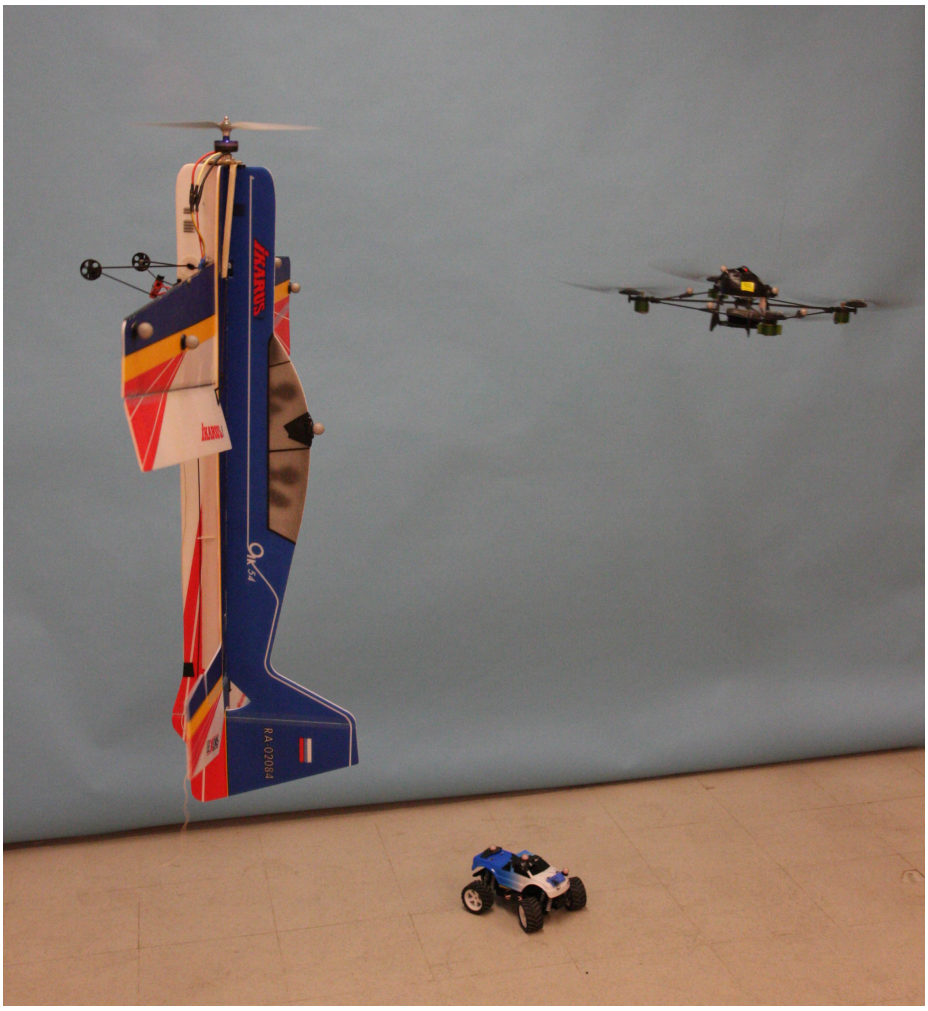


Figure 1-1: Vehicles under autonomous control in MIT's RAVEN

used to provide highly accurate measurements of the positions of numerous ground and aerial vehicles within a flight space measuring approximately $6 \times 8 \times 5$ m. The array of Vicon cameras can be seen in Figure 1-3. This positioning information is distributed in real-time to a number of processing computers that run the controllers for each vehicle in the system. The control commands are then sent to the vehicles over a R/C wireless link, closing the control loop and stabilizing the vehicles.

The system allows multiple vehicles to be flown and tested in a controlled environment simultaneously. To date, a number of different vehicle types have been used in the system to including helicopters, quadrotors, fixed wing aircraft, and ground vehicles. Figure 1-1 shows a variety of vehicles under autonomous control in the RAVEN.

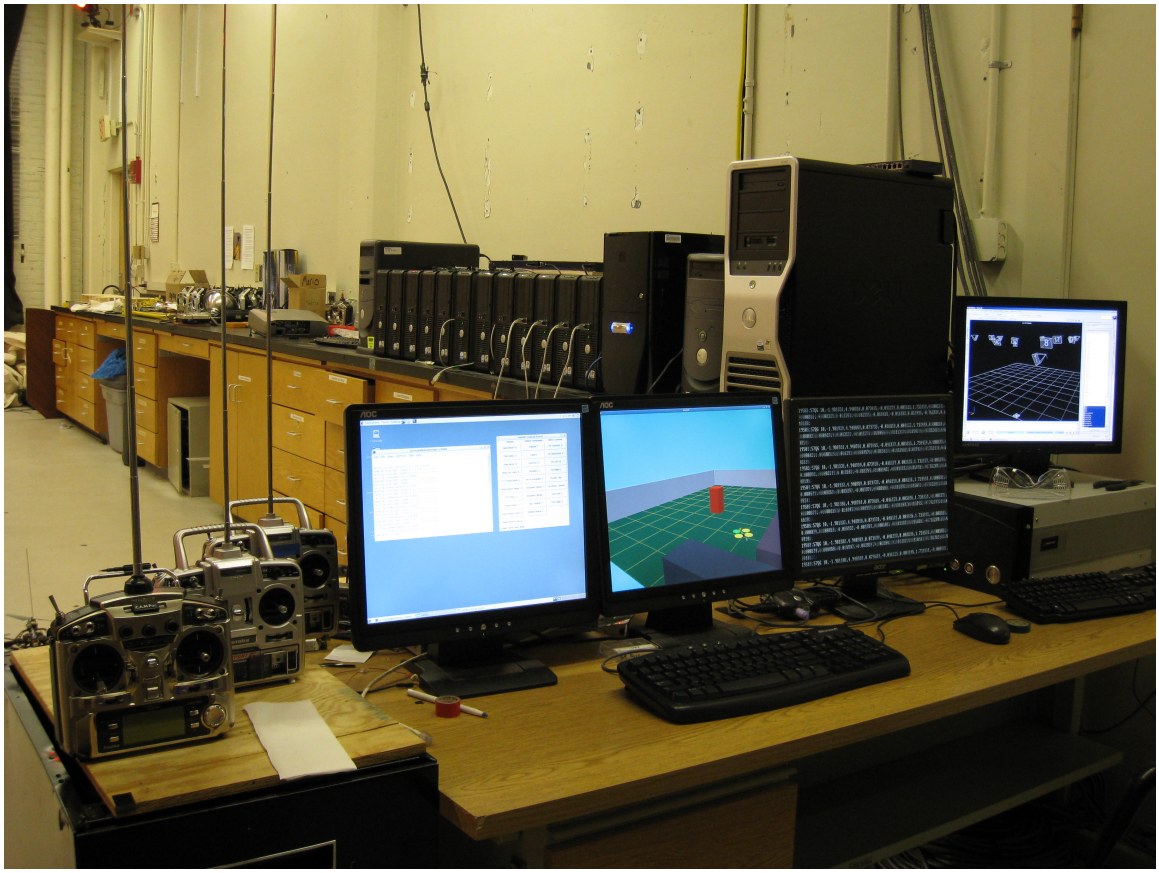


Figure 1-2: Operator station and vehicle control computers

Figure 1-3: MIT's Real-time indoor Autonomous Vehicle test ENvironment (RAVEN) showing the Vicon motion capture camera system.

This research used fixed wing balsa/mylar airplanes designed and built specifically for this research, see Chapter 3 for more information.

1.3 Approach

This thesis presents methods for computing maneuvering decisions for one-on-one air combat, also referred to as basic fighter maneuvering (BFM). The primary contribution of this work is the application of a neuro-dynamic programming learning architecture to the air combat problem. The result is an algorithm capable of learning a maneuvering policy and utilizing this policy to make air combat decisions in real-time. Leading up to the work with dynamic programming, different formulations for a scoring function to quantify the desirability of given combat states were considered. In Chapter 2, the thesis describes the development of two such scoring functions for use in air combat decision making. The utility of these functions is tested using a search tree algorithm in an air combat simulation. In Chapter 3 the scoring functions are used to develop a path planner capable of operating in real-time. This planner is implemented in the RAVEN where actual unmanned aircraft (UAs) are flown in air combat. Additionally, a discussion is included in Chapter 3 on the development of the flight vehicles and flight controllers used in this demonstration. Results from the flight tests are presented. The scoring function discussed in Chapters 2 and 3 is used in further development in the following chapters. Chapter 4 discusses the methods used to formulate the problem as an neuro-dynamic program. This approach defines BFM as a discrete time approximate dynamic programming problem. The resulting policies are easily evaluated in real-time. Chapter 5 describes the calibration of this method and presents the resulting simulation performance data and flight results. Chapter 6 concludes the thesis with a summary, a discussion of the thesis objectives, and suggested research directions for the continuation of this work.

Chapter 2

Scoring Function Development

The first step taken to develop an air combat trajectory planner was to develop a way to quantify the desirability of a given combat situation. This chapter describes the development of two methods for evaluating the desirability of any given orientation of the two aircraft relative to each other. The utility of these scoring functions is tested with a receding horizon control trajectory planner. The planner that makes maneuvering decisions in a simplified air combat game by attempting to maximize the value returned from the scoring functions.

The combat game involves two aircraft engaged in BFM (Basic Fighter Maneuvering). The threat or adversary aircraft will be referred to as the *red* aircraft. The friendly interceptor aircraft will be referred to as the *blue* aircraft. The methods presented in this paper will always be applied to the blue aircraft. Both aircraft have a similar, yet conflicting, goal: To establish flight in a specified region behind the other aircraft. The complexity of air combat is simplified to some degree to make formulating and evaluating new methods simpler and more intuitive.

The major simplifications of the air combat game used in this project are limiting both aircraft to level flight, and fixing the velocity of the aircraft to a constant value. These simplifications remove some of the dynamic complexity of air combat, while keeping the basic maneuvering problems which are critical to BFM. This simplified problem allows a human observer to easily assess the desirability of a particular maneuver. Weapons employment will also be ignored in this game. Both aircraft

will continue to maneuver until one establishes flight behind the other for a specified period of time. However, the ability of the aircraft to avoid threat zones (i.e. area in front of the nose of the red aircraft) will be measured along with time required to complete an intercept. Another technical issue that will be ignored is that of the required sensor technology. It will be assumed that both aircraft have perfect information about the other aircraft's location and orientation. In reality, the position and orientation information could be derived from a number of sources with various degrees of accuracy. This thesis will focus on using that information, not how it is collected.

The first of two scoring functions (the *baseline scoring function*) used to evaluate air combat situations is based on basic geometry between the two aircraft engaged in combat. This method is described in Sectionr̃efsec:bl. The resulting performance of this scoring function, when used to guide an aircraft in simulated combat, is presented in Section 2.2. In Section 2.4 the baseline scoring function is used as a basis for the development of the *expert modified scoring function*. This function includes heuristic components which encourage behaviors desired by experience combat pilots during simulation. The expert modified scoring function is also testing using simulation to compare with the baseline. In the following chapter, both of these scoring functions are tested in actual flight using micro-UAS. The scoring functions and other methods presented in this chapter are also used in the development of the dynamic programming solution.

2.1 Baseline Scoring Function

The relative merit of every possible state has been reasonably captured by a scoring function. The scoring function is intended to guide an aircraft to a position of advantage and avoid threatening defensive situations. The scoring function considers relative aircraft orientation and range. The *baseline scoring function* is computed in a similar fashion to methods described in [2, 3] and provides an initial basis for quantifying the value of a given combat situation. Figure 2-1 shows a representation

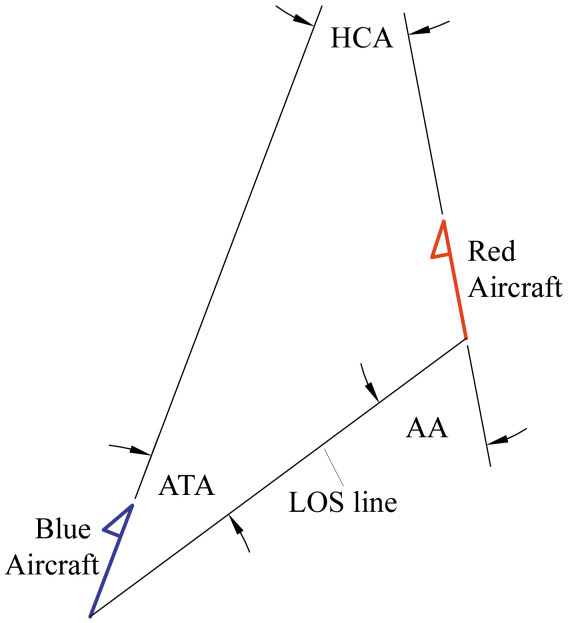


Figure 2-1: Aircraft relative geometry showing Aspect Angle (AA), Antenna Train Angle (ATA) and Heading Crossing Angle (HCA).

of relative aircraft geometry. The angles are shown from the point of view of the blue aircraft. The red and blue aircraft are represented by velocity vectors rooted at the aircraft center of mass. The aircraft centers of mass are connected by the line of sight (LOS) line. The aspect angle (AA) is the angle between the tail of the red aircraft and line of sight line. Aspect angle is a value monitored by pilots of fighter aircraft (both visually and electronically) to help make maneuvering decisions. The antenna train angle (ATA) is measured from the nose of the blue aircraft to the LOS line. The three simple terms AA, ATA, and range are sufficient to describe the basic geometry of a BFM situation to a pilot: ATA tells the blue pilot where to look, range indicates how far to look and AA represents which direction the red aircraft is heading. Another number used to extract more details about the current situation is the heading crossing angle (HCA), which is simply the difference in heading between the two aircraft. AA, ATA and HCA are all limited to a maximum magnitude of 180° by definition. As a convention, angles to the right side of the aircraft are considered as positive, angles to the left as negative.

The orientation and range contributions were developed using the approach in [2].

In order to maneuver to a position of advantage the blue aircraft wishes to minimize both AA and ATA. This is summarized by the function:

$$S_A = 1 - \left[\left(1 - \frac{\texttt{AA}}{180^\circ}\right) + \left(1 - \frac{\texttt{ATA}}{180^\circ}\right)\right] \tag{2.1}$$

The goal is to minimize S_A. The best case is $S_A = -1$ for the offensive position represented by $\texttt{AA} = \texttt{ATA} = 0^\circ$. The worst case is $S_A = 1$ for the defensive position of $\texttt{AA} = \texttt{ATA} = \pm 180^\circ$.

A range contribution returns a score relative to a desired range (R_d).

$$S_R = e^{-\left(\frac{|R - R_d|}{180^\circ k}\right)} \tag{2.2}$$

where R is the current range from the red aircraft in meters. The constant k has units of meters/degree and is used to adjust the relative effect of range and angle. A value of 0.1 was found to be effective for k. The orientation and range terms are combined to create the baseline scoring function using the following equation:

$$S_{bl} = S_A S_R \tag{2.3}$$

Further discussion of the development of Equation 2.3 is available in [2, 3]. These papers use four different scores that contribute to the cost function: orientation, range, velocity, and terrain. Since velocity changes and terrain are not considered in the air combat game, Equation 2.3 is used as a basis for comparison. This function accepts a state with 8 components that represent the current combat situation. This state is composed of x–position, y–position, heading, and bank angle for both blue and red aircraft. The state vector,x, will be defined as: $x = \{x_b, y_b, \psi_b, \phi_b, x_r, y_r, \psi_r, \phi_r\}$. The function returns a 0 for neutral positions, a negative number for offensive positions and positive number for defensive positions. Thus, in combat the goal is to minimize the value of the scoring function.

A graphical depiction of the baseline scoring function can be seen in Figure 2-8.

These plots are a cross section of what is actually a 6-dimensional function (S_{bl} does not use aircraft bank angle in computing the score, and thus ignores two components of the state). This particular representation shows the values as a function of blue aircraft location relative to the red aircraft. Red aircraft location and heading is fixed. In each plot the blue aircraft heading is fixed to the indicated value. Blue color on the plot represents blue advantage, while red represents red advantage. The red aircraft's current position and flight history is represented by the red asterisk and line, respectively. The red aircraft 3 o'clock - 9 o'clock line (3/9 line) is also plotted. When red and blue headings are aligned, the 3/9 line is the dividing line between offensive and defensive positions for the blue aircraft. However, note how the blue aircraft heading effects the zones deemed offensive. When headings are opposed, as in Figure 2-8(c), neither aircraft has an advantage. Finally, note that the minimum score of -1.0 is only possible when the headings are aligned with the blue aircraft centered in the goal zone, see Figure 2-8(a).

2.2 Combat Simulation using the Baseline Scoring Function

The simulation uses a limited look-ahead search which evaluates a heuristic scoring function and predefined adversary policy. During execution, the algorithm conducts a depth first search starting from the current two aircraft states. The depth first search considers each control action $u_b \in \{left,\ straight,\ right\}$ of the pursuing aircraft and the adversary aircraft. At each time-step both aircraft have three available control actions: roll left, roll right, or maintain the current bank angle. The adversary action selection is dictated by a predefined human constructed policy. The algorithm searches a fixed number of time- steps into the future and then computes the values of the trajectories using a heuristic scoring function. The first action of the best trajectory is executed. The process then repeats.

The baseline scoring function was used to make decisions in a simulated air combat

game. A simulation was developed using simplified vehicle dynamics based on the micro-UAS shown in Figure 3-2. The state of the game is defined for each aircraft by the state x as described above. The flight dynamics and limitations are summarized by the state transition matrix, $f(x, u_b, u_r)$, described in Algorithm 1. The function $f(x, u_b, u_r)$ is the heart of the simulation algorithm. The roll rate for the aircraft is fixed at $45°/s$, this represents the maximum roll rate possible with full rudder deflection. The velocity was fixed at 2.5 m/s, which is a good maneuvering speed for the aircraft. Both aircraft are restricted to level flight, so a given bank angle corresponds to a particular lateral acceleration by the following relationship:

$$A_{lat} = g \tan(\Phi) \tag{2.4}$$

where $g \approx 9.81 m/s^2$. The blue aircraft was given a performance advantage over the red aircraft by having a larger maximum bank angle. For the blue aircraft $\phi_{\max}^{blue} = 23°$ and for red $\phi_{\max}^{red} = 18°$.

A performance advantage is a common technique used in actual BFM training to assess a student's improvement from engagement to engagement. In the simulation, we wish to assess the blue aircraft's performance using various maneuvering policies. It is difficult to assess the performance of a particular policy if the two aircraft continue to maneuver indefinitely (as would be the case with equivalent maneuvering policies and equivalent performance). The performance advantage allows the use of time to intercept (TTI) as the primary measure of the effectiveness of a particular maneuvering policy. The metrics used for comparison are: time to intercept (TTI), defensive time and threat time. TTI was measured from the beginning of the simulation until an aircraft was established in a position of advantage. This position was defined as an area $\pm 30^o$ off the tail of the adversary in a range of 1 to 3 m, see Figure 2-2. The aircraft needed to maintain this position continuously for 3 s to ensure it was truly established in position. It is assumed that once blue is established, there is little chance the red aircraft will manage to escape. This particular region was selected based on the turn performance of the aircraft and the reaction time of the

Algorithm 1 State Transition Function $f(x_i, u_b, u_r)$ (Simulates 0.25 s, given red and blue actions)

Input: $\{x, u_b, u_r\}$
$\Delta t = 0.05$, $v = 2.5\ m/s$, $\dot{\phi}_{\max} = 45°/\text{s}$
$\phi^{red}_{\max} = 18°$, $\phi^{blue}_{\max} = 23°$
for i=1:5 (once for each 0.05 s time increment) **do**
  for both red and blue aircraft **do**
    if $u = left$ **then**
      $\phi = \phi - \dot{\phi}\Delta t$
      if $\phi < -\phi_{\max}$ **then**
        $\phi = -\phi_{\max}$
      end if
    else if $u = right$ **then**
      $\phi = \phi + \dot{\phi}\Delta t$
      if $\phi > \phi_{\max}$ **then**
        $\phi = \phi_{\max}$
      end if
    end if
    $\dot{\psi} = \frac{g}{v}\tan(\phi)$
    $\psi = \psi + \dot{\psi}\Delta t$
    $x^{pos} = x^{pos} + \Delta t \sin(\psi)$
    $y^{pos} = y^{pos} + \Delta t \cos(\psi)$
  end for
end for
Output: (x_{i+1})

path planner. If the blue aircraft follows closer than approximately 1 m behind the red aircraft, the planner may not have enough time to react if the adversary makes an unexpected move. Further back than approximately 3 m, the red aircraft has the ability to make a tight turn and meet the blue aircraft with a high aspect angle. The minimum value of the baseline scoring function is centered at 2 m to encourage flight in this region. Defensive time was accumulated any time $|\texttt{AA}| > 90^o$. Threat time is similarly defined as time the red aircraft could potentially employ ordnance ($|\texttt{AA}| < 150^o$).

The control actions available to the aircraft were also limited as a further simplification. Both aircraft have three actuators: elevator, rudder and throttle (see Chapter 3). Due to the limitation of level flight and constant velocity, the elevator control and throttle control are scheduled to maintain these parameters. The remain-

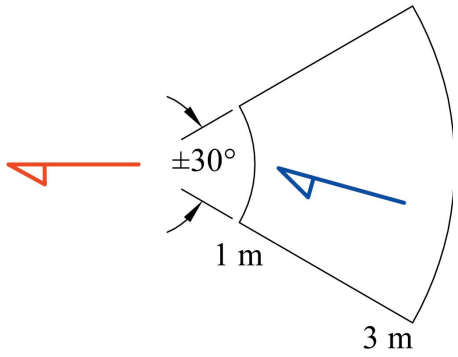


Figure 2-2: Defined position of advantage behind adversary.

ing control action is aircraft roll, which is discretized to minimize the search space. At each planning time-step ($\Delta t = 0.25$ s) the planner has the option to (1) roll left, (2) maintain the current ψ, or (3) roll right, so $u \in \{left, no-roll, right\} \equiv \{1, 2, 3\}$. This simplifies the aircraft control, but still allows for a wide range of bank angles. The application of these control actions can be seen in Algorithm 1.

A search was in a simulation algorithm which examined all possible combinations of blue maneuvering. Depth first search was used to examine each combination of blue aircraft control actions. For example, when using a search simulating 8 time-steps into the future, there are 3^8 or 6561 possible sequences of control actions ($\bar{u}$). The set of all possible sequences will be referred to as U, where U_i refers to the i^{th} sequence, $\bar{u}_b^i$. The n^{th} action in the i^{th} sequence would be $\bar{u}_b^i(n)$. In the 8 step example, the set of all possible control sequences would be:

$$U = \begin{cases} \{11111111\} \\ \{21111111\} \\ \{31111111\} \\ \{12111111\} \\ \quad \vdots \\ \{23333333\} \\ \{33333333\} \end{cases} \tag{2.5}$$

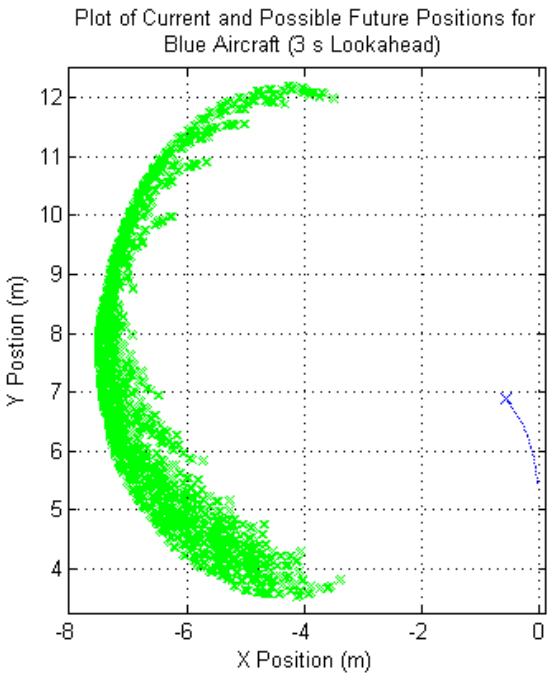


Figure 2-3: The blue line and X represents the blue aircraft current position and recent flight history. The green X's represent all possible future blue aircraft positions following 3 s look-ahead. The resulting positions and headings from all possible trajectories are compared using the scoring functions.

By evaluating each sequence of control actions in order with $f(x, u_b, u_r)$, the resulting blue and red locations and orientations can be computed. An example result from the evaluation of each sequence in U with an 8 step look-ahead and $\Delta t = 0.375$ s is shown in Figure 2-3. Most of the states appear to be to the left of blues initial position. This is because in initial state the blue aircraft is banked full left ($\phi = -23°$) with a compass heading of 315° (equivalent to $\psi = -45°$). Even with $u_b = \{3\ 3\ 3\ 3\ 3\ 3\ 3\ 3\}$, the blue aircraft would continue turning left for some time before coming back to the right. In order to find a value for each one of these states (from the baseline scoring function), an assumption must be made regarding the red aircraft maneuvering policy (π_r). For this simulation three different policies were assumed for the red aircraft: *left turning*, *pure pursuit* and *multi-policy*. The left turning policy assumes the red aircraft will always make a maximum performance left hand turn, so $u_r = 1$ at all time-steps. The pure pursuit policy assumes that the red aircraft will make a turn in the shortest direction to obtain pure pursuit

Algorithm 2 Control Planning Algorithm

Input: Current Game State {x}
Initialize U: set of all possible $\bar{u}_b$ sequences
Initialize L: number of time-steps to look ahead
Initialize $J_{Best} = \infty$
for $i = 1 : L^3$ (L^3 is the number of sequences) **do**
  $x_{temp} = x$
  $\bar{u}_b^i = U_i$
  for $j = 1 : L$ **do**
    $x_{temp} = f(x_{temp}, \bar{u}_b^i(j), \pi_r(x_{temp}))$
  end for
  if $S_{bl} < J_{Best}$ **then**
    $J_{Best} = S_{bl}$
    $\bar{u}_{best} = \bar{u}_b^i$
  end if
end for
$\bar{u}_b = \bar{u}_{best}$
Output: {u_b} sequence

($\texttt{AA} = 180°$). The multi-policy considered three possible red policies simultaneously: Left turning, right turning and pure pursuit.

The planning algorithm is depicted in Algorithm 2 The planner generates a sequence of actions for both aircraft, then predicts what the state will be at the end of the planning horizon. Once the resulting aircraft states are determined, they are evaluated by the baseline scoring function ($S_{bl}(x)$ from Equation 2.3). The control sequence resulting in the minimum score is chosen as the best choice. When using the multi-policy, the depth first search algorithm selected the blue maneuvering choice that provided the lowest average score over each red policy. The chosen path is then sent to the simulation or flight controller for execution. The approximate computation time was 0.25 s. So, although 3 s of commands were generated by the search, only the first 0.25 s are executed before a new plan is computed. This allows for rapid re-planning when the red aircraft policy is not exactly as expected.

Using the simulator, a variety of specific starting positions were evaluated to assess BFM performance. The simulation allows for consistent starting conditions from one set to the next. Five setups were used in the initial simulations for comparison of the two scoring functions. Setups 1 through 5 are depicted in Figure 2-4. They are

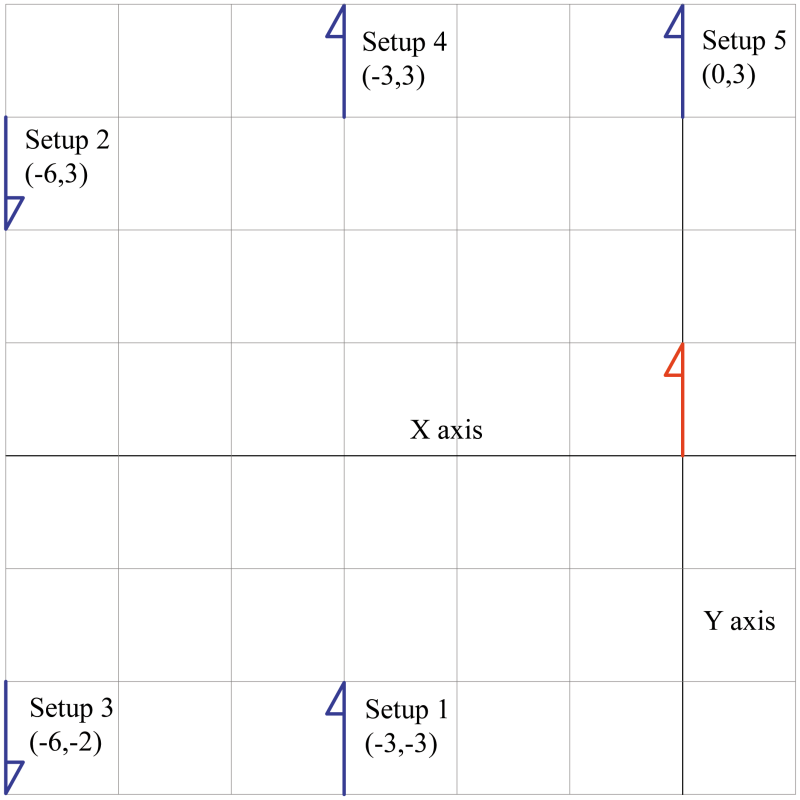


Figure 2-4: Starting geometry for simulation runs. The red aircraft is always at the origin heading North. The units of the coordinates on the graph are in meters.

in order of increasing difficulty for the blue aircraft. Setup 1 is an offensive starting position. Setups 2 and 3 are neutral starting positions. Setups 4 and 5 are both defensive, decreasing in range.

2.3 Simulation Results using Baseline Scoring Function

As mentioned above, several different methods were considered for the *assumed* red maneuvering policy, π_r. Clearly, it is impossible to know what the red aircraft is going to do, so some assumption must be made. In order to evaluate the effectiveness of the various assumptions, each assumed red policy {left turning, pure pursuit, multi-policy} was tested against three *actual* red policies, {left turning, pure pursuit, random}. This was done to determine which assumptions about red would produce

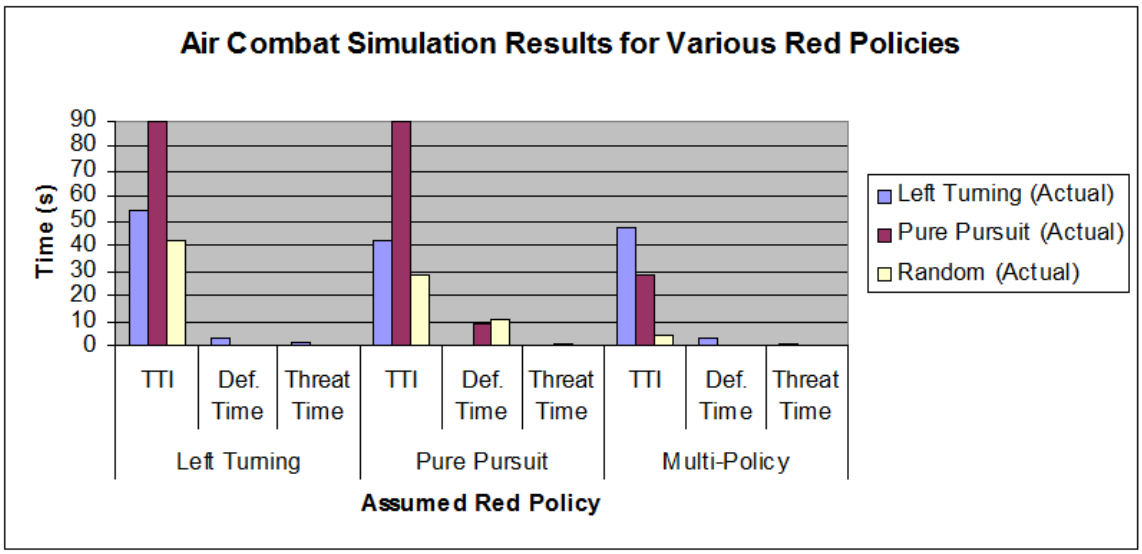


Figure 2-5: Results from combat simulation using various red policies. All simulations were accomplished using Setup 3. A TTI of 90 s represents failure to complete the intercept. Better overall performance was enjoyed by the multi-policy algorithm.

the best overall planning algorithm. The results can be seen in Figure 2-5. Overall, the assumed left turning policy had the highest TTI against each of the actual red maneuvering policies. The 90 s TTI against the pure pursuit policy indicates a failure to win in 90 s. This is not surprising since assuming the red aircraft will continue left is a poor assumption. The planner using the assumed pure pursuit policy did not perform much better. Again, the planner failed to win against the pure pursuit bandit in the maximum allowed time of 90 s. The multi-policy had the overall lowest average TTI. The multi-policy method was thus chosen for future testing and implementation against an unpredictable adversary. The strength of the multi-policy is that it selects the maneuver that produces the best average score, over three worst case red maneuvering strategies scenarios. The multi-policy proved to have the best performance over a variety of red behaviors.

During simulation testing of the baseline scoring function it became clear that some sub-optimal decisions were being generated by the path planner. A representative maneuvering error can be seen in Figure 2-6. This is referred to as a flight path overshoot. The numbers along the flight path represent time in seconds. In this situation the blue aircraft takes an overly aggressive bid toward the red aircraft.

This maneuver is similar to "pointing" the aircraft at the opponent; it is a natural mistake for humans and a common error among novice pilots. The blue aircraft ends up crossing the red aircraft's 6 o'clock with a relatively high `HCA` at approximately 4.5 s. Once the blue aircraft drifts outside the red aircraft flight path, the red aircraft is able to increase range and `ATA` by continuing the left turn, or force another overshoot by a quick reversal to the right. These reactions can result in a larger TTI or even a role reversal from offensive to defensive for the blue aircraft. The root cause of this error is the limited look ahead available to this search algorithm. While the search horizon increases linearly, the computation time increases exponentially. The search algorithm uses a prediction of the red aircraft's future position to optimize the blue aircraft's actions. Due to the large search space, the search algorithm is limited to an approximately 3 s look-ahead. This short search horizon leads to relatively greedy maneuver selection in simulation and flight implementation. This observation is consistent with those made by Austin et al. [2] when dealing with short search horizons using game theory.

In the next section, improvements are made to the baseline scoring function to encourage better maneuvering without increasing the search time. There are many techniques that have been developed by pilots of combat aircraft which help to improve tactical performance. The modifications to the baseline scoring function are designed to encourage some of these techniques. A discussion of the results of simulation testing and a comparison of the scoring functions is included in section 2.4.1.

2.4 Expert-modified Scoring Function

The baseline scoring function was modified to encourage good decisions despite the limited look-ahead. Maneuvering policies, as described in [21], are rules utilized by human pilots which approximate optimum solutions. Algorithm performance improvements were realized by modifying the scoring function to encourage usage of these simple BFM concepts.

The basic cost function was extended by adding an extra term that quantifies

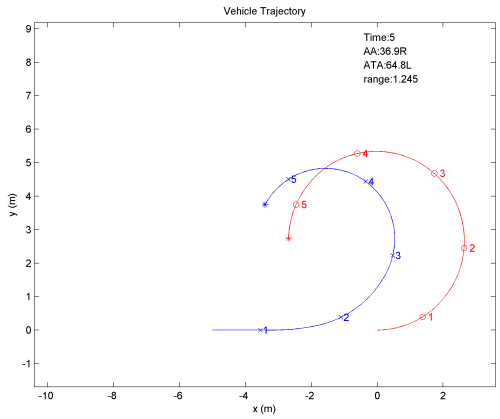


Figure 2-6: A poor decision made when using the baseline scoring function.

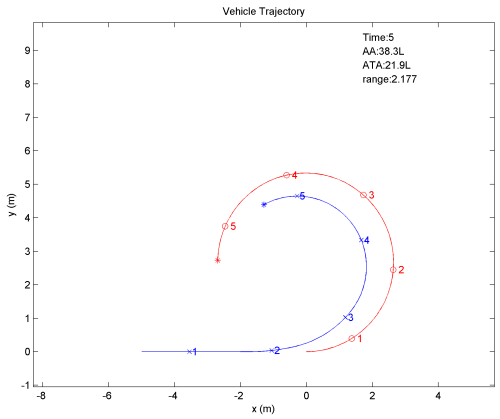


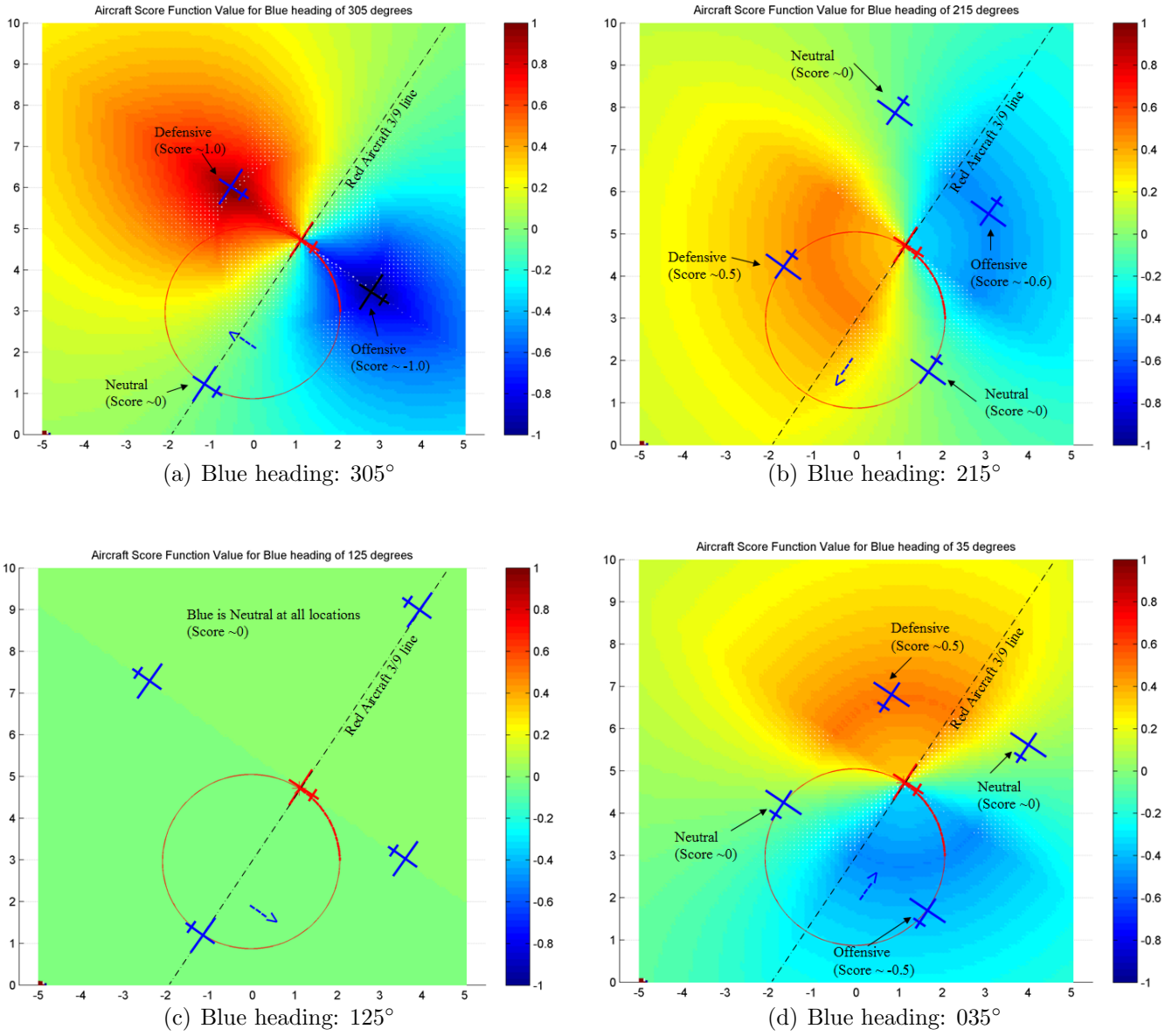
Figure 2-7: The expert-modified scoring function makes a better maneuvering decision.

the effect of the red aircraft turn circle. While the basic function returns a value representing the current orientation of the two aircraft, it provides no information regarding maneuvering choices made by the red aircraft. The turn circle of the red aircraft is a way of representing its current maneuvering and possible future maneuvering. It is a tool used by human pilots to visualize the current state and make maneuvering decisions [17]. The simplest form of BFM is a defensive aircraft making a continuous maximum performance turn to keep an offender from lining up behind while the offensive aircraft turns to establish a firing position. A combination of turning rate and geometry can be utilized by both aircraft to improve their individual situations. An increased turning rate by one aircraft will, over time, improve that aircraft's position in relation to the other aircraft. Consequently there is the necessity to maximize aircraft performance. Likewise, appropriate use of lead, pure and lag pursuit can be used to affect the flight geometry to appropriately increase or decrease the range between the aircraft [21]. The turn circle provides a basis with which to make these decisions.

The red aircraft turn circle (TC) is shown in Figure 2-8. Assuming similar aircraft with equal energy, the blue aircraft is in a relatively stable position when established in flight on the red turn circle. From this stable position, a turn to the inside of the circle will reduce the range between the aircraft (as in a low yo-yo maneuver [21]).

Similarly, a turn to the outside of the turn circle will bring about an increase in range (as in a high yo-yo). For an offensive aircraft, consideration of the adversary's turn circle helps to avoid overshoots (passing the tail of the enemy aircraft with high heading crossing angle). Overshoots can lead to a long TTI or even a role swap (becoming defensive). The turn circle can also help choose the appropriate lead turn for high aspect situations. Simply put, the goal of the offender is to align the turn circles, then reduce the range. In a defensive role, consideration of the offender's turn circle helps make good evasive decisions. The goal of the defender is to mis-align turn circles. By incorporating the turn circle into the scoring function, it was found that better maneuvering decisions were made with minimal additional computational complexity. Figure 2-7 shows how consideration of the turn circle helps to improve intercept performance. The simulations represented by Figures 2-6 and 2-7 began with the same starting conditions and ran for the same amount of time. The blue aircraft using S_{bl} in Figure 2-6 greedily begins to point at the red aircraft too early because this maneuver minimizes the return from the scoring function. However, this early turn leads to a flight path overshoot as the blue aircraft crosses the red aircraft turn circle with a high `HCA`. This overshoot will lengthen the amount of time it will take for the blue aircraft to establish flight in the goal region. The blue aircraft in Figure 2-7 delays making the turn until a more appropriate time due to the modifications incorporated into the *expert modified* scoring function (S_{em}). The development of this function is detailed below.

The turn circle portion of the scoring function keeps a history of the red aircraft position in memory. A least squares circle fit is used to approximate the current red aircraft turn circle. Based on the 100 Hz data available from RAVEN it was found that computing the turn circle using the 5 previous aircraft recorded positions produced a reasonable turn circle approximation. Because only a few points were used the turn circle very closely approximated the instantaneous turn circle. The simulation used the same technique. The curve fit returns the center point location (x_c, y_c) and a radius r_c. The distance of the blue aircraft current position from (x_c, y_c) is computed as r_b. The turn circle contribution to the function is then computed



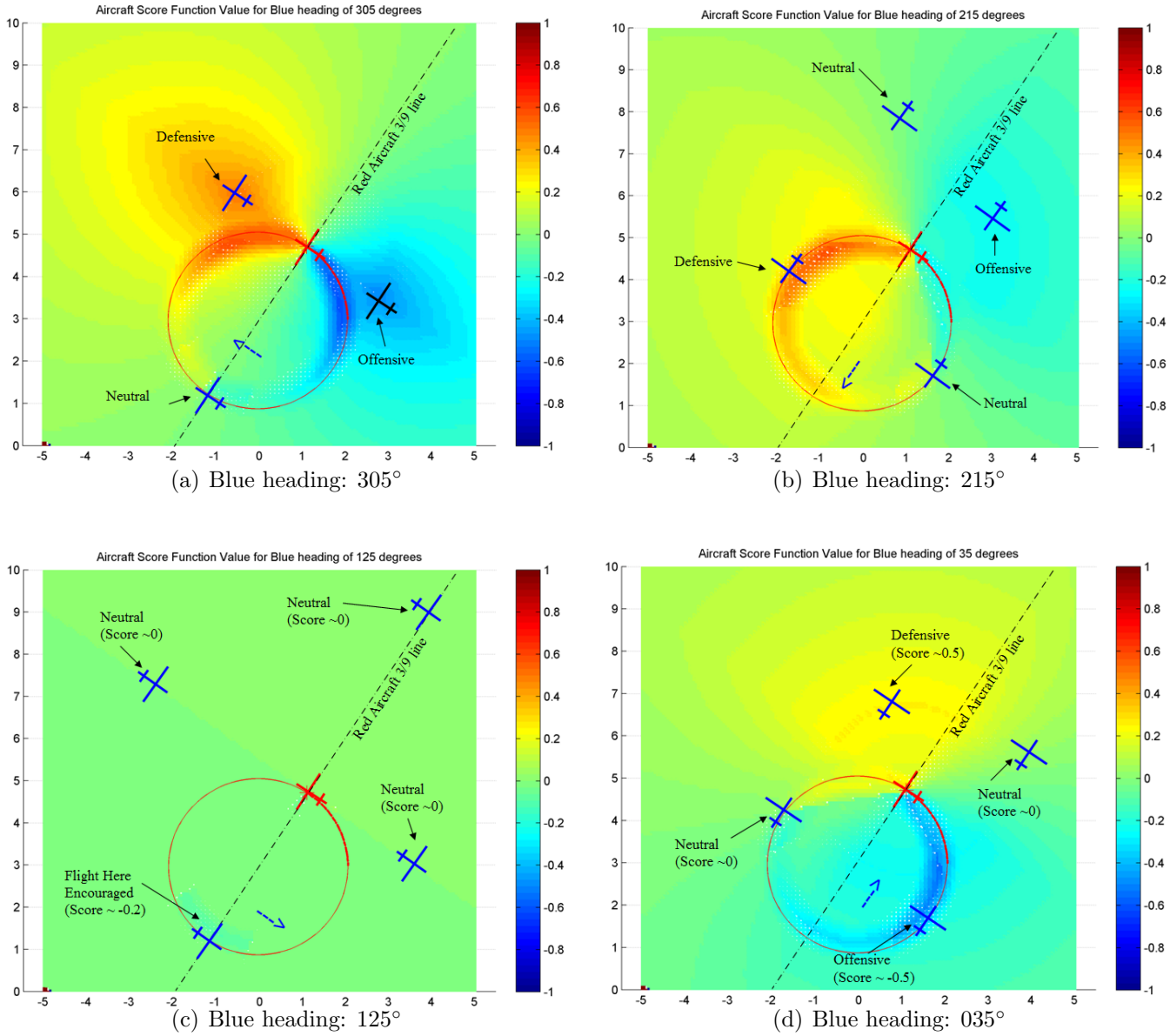
(a) Blue heading: 305°

(b) Blue heading: 215°

(c) Blue heading: 125°

(d) Blue heading: 035°

Figure 2-8: Visualizations of the *baseline scoring function*. The blue areas represent negative values of S_{bl}, these are areas of blue advantage. The red areas represent red advantage. In this plot the red aircraft position and heading are fixed as depicted. The blue aircraft's heading is also fixed. The colors represent the score the blue aircraft would receive for being in various locations relative to the red aircraft. The red aircraft 3/9 line and instantaneous turn circle are shown for reference. The units of: x–y position are in meters.



(a) Blue heading: 305°

(b) Blue heading: 215°

(c) Blue heading: 125°

(d) Blue heading: 035°

Figure 2-9: Visualizations of the *expert-modified scoring function*. The blue areas represent negative values of S_{bl}, these are areas of blue advantage. The red areas represent red advantage. In this plot the red aircraft position and heading are fixed as depicted. The blue aircraft's heading is also fixed. The colors represent the score the blue aircraft would receive for being in various locations relative to the red aircraft. The red aircraft 3/9 line and instantaneous turn circle are shown for reference. The effect the red aircraft's turn circle has on the function can be seen. The units of x–y position are in meters.

as:

$$S_{TC} = 2 - 2\,|r_c - r_b| \tag{2.6}$$

By multiplying S_{TC} by S_{RA}, both positive and negative values of S_{RA} are scaled up. In other words, when the blue aircraft is defensive, the scoring function will encourage flight away from the turn circle, thus foiling the red aircraft's attempt to align the turn circles. Conversely, when the blue aircraft is offensive, the search algorithm will chose maneuvers which align the turn circles.

The final term added to the basic cost function encourages the blue aircraft to establish a flight path that is tangent to the red aircraft's turn circle. This term further rewards establishing flight on the turn circle. It is especially helpful in guiding the aircraft to establish the correct flight-path separation in a high aspect merge ($\mathtt{AA} \sim \mathtt{180}^\circ$) as suggested by Shaw [21]. This term is computed as:

$$S_C = \frac{|\mathtt{AA}| + |\mathtt{ATA}|}{180} \tag{2.7}$$

This term is only computed for locations on the red aircraft turn circle for which the blue aircraft heading is oriented within 30^o of the circle tangent. This encourages turn circle alignment in a blue offensive situation and turn circle misalignment in a blue defensive situation. The final formulation for the new scoring function is then calculated as:

$$S_{em} = \frac{S_{RA} S_{TC}}{2} + S_{TC} S_C [\mathrm{sgn}\,(S_{RA})] \tag{2.8}$$

A graphical representation of eq. (2.8) is seen in Figure 2-9. As in Figure 2-8, this is a cross section of the expert-modified scoring function. This function depends on six components of the state vector, $\{x_b, y_b, \psi_b, x_r, y_r, \psi_r\}$, as well as the recent flight history of the red aircraft. In this visualization, the red aircraft position, flight history and heading are fixed. The blue heading is also fixed in each of the subplots. The values of the function vary for different positions of the blue aircraft in relation to the red aircraft. The addition of the turn circle term clearly encourages flight along the red aircraft turn circle. The benefits of the function modifications are discussed

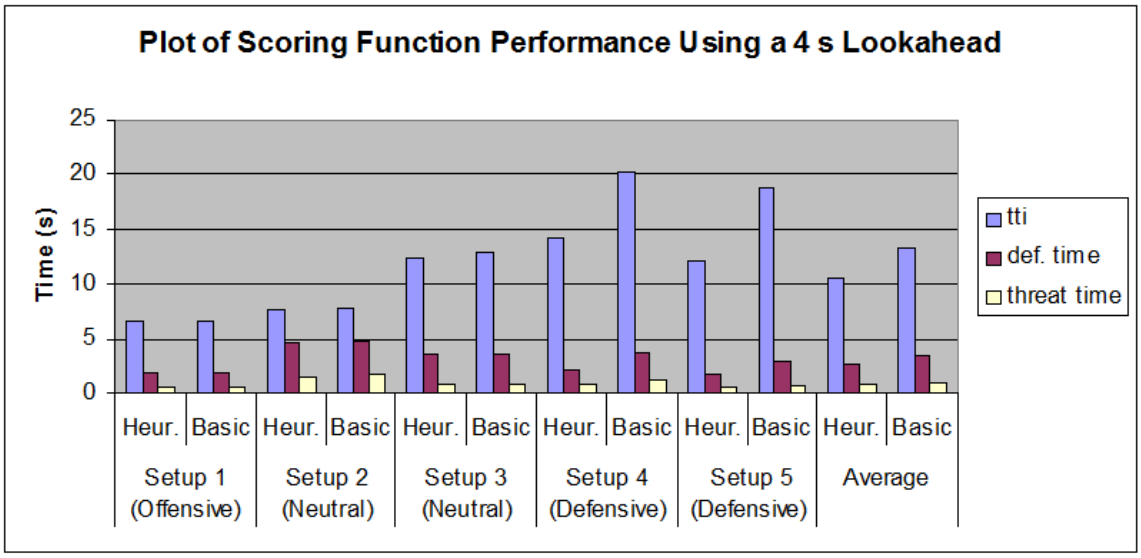


Figure 2-10: Results from combat simulation using 4 s look-ahead.

in the next section.

2.4.1 Simulation Comparison

To determine the tactical advantages of using the heuristic function over the basic function a number of simulations were run. Both functions were tested using the set of initial conditions shown in Figure 2-4. The setups increase in difficulty from setup 1 to setup 5, with setup 5 being the most difficult and most defensive setup. In general the results show a greater advantage for the heuristic function for the more difficult setups. It appears, the longer the engagement, the more gain. The expert-modified scoring function showed an overall average lower TTI, defensive time, and threat time. The times were equal or better in each of the individual setups. This suggests that the expert preference additions to the baseline scoring function had the desired result of encouraging better overall tactical behavior from the planner without causing any situation specific decrease in performance. The algorithms used in the simulations were recoded into C++ and implemented in the RAVEN for actual flight testing. These results are discussed in next chapter.

Chapter 3

Combat Implementation Using Heuristic Value Function

Following successful testing in simulation, the next step was to implement the combat planner using actual UAs flying in RAVEN. In order to accomplish this task, the aircraft themselves had to be designed, built and flight tested. A brief description of this process follows. Subsequently, the author designed and tested a low level flight controller and implemented a trajectory follower algorithm to achieve autonomous flight. Finally, the combat planner software was integrated into RAVEN to complete actual air combat experiments. This chapter concludes with a discussion of air combat flight test results.

3.1 Fixed Wing Aircraft Development

The flight vehicles used for air combat flight testing were designed and built in-house. Previously, the majority of flight testing accomplished in the RAVEN was done with quad-rotor aircraft [15]. The hovering capabilities of these vehicles are very well suited to the space limitations in RAVEN. An autonomous fixed wing aircraft was successfully flown within RAVEN in [8]. The airplane was capable of both horizontal and vertical takeoffs, hovering flight, transition to forward flight, forward flight, transition back to hover, and perch landings. While this was a very capable aircraft and

autonomous control system, the approximately 7 m/s forward speed of the airplane made it unusable for air combat maneuvering within the confines of the RAVEN flight space. To accomplish fixed wing combat flight testing in the confined flight space required, an airplane with a slow flight speed (approximately 2 m/s) and the ability to make tight turns (turn radius of approximately 1 m) needed to be developed.

Ten different model airplane configurations were designed and built by the author and assistants. Some aircraft were built with balsa with mylar covering and others with foam. The designs included conventional mono-wing and biplane airplanes, as well as flying wing designs. Due the desire to have a maneuverable airplane many attempts were made to have functional ailerons. This proved to be challenging as the aileron effectiveness was very low at such slow flight velocities. Increasing the size of the ailerons failed to succeed due to the limited torque available from the extremely lightweight micro-radio controlled actuators and servos.

The airplane that met the requirements was a design loosely based on the Cloud II airplane [1] and had only rudder and elevator flight controls. After approximately five design iterations including considerable flight testing and modifications at each step, the aircraft shown in Figure 3-1 was chosen for use in combat flight testing. A photograph of the test aircraft can be seen in Figure 3-2. The aircraft weighs approximately 20 g. It has an airframe constructed of balsa and is covered with RA Microlite available from [9]; this is a self adhesive iron-on mylar film with a weight of 0.8 g/m^2. The polyhedral wing approximates a circular arc to maximize the dihedral effect. To achieve this, the panels of the wing have the following angles in relation to level: $\{-15°, -5°, 5°, 15°\}$; this can be seen in the front view in Figure 3-1.

The ribs of the wing are cut from balsa in the shape of circular arcs. The wing is covered only on the top to reduce weight and to produce a wing with high camber. This was found to be the best approach as the mylar film does not adhere well to concave surfaces. As such, covering the bottom of the wings produces an airfoil with a flat bottom. We determined through flight testing that a flat bottomed airfoil fails to produce sufficient lift at the slow speeds desired. Experimentation also demonstrated that a wing located above the aircraft center of gravity produced the

best stability and turning ability with the polyhedral wing. Additionally, the high wing allows the propeller wash to flow directly over the vertical tail and rudder, thus providing sufficient rudder authority. Both rudder and elevator control surfaces were mass balanced and aerodynamically balanced to minimize the torque required from the micro-actuators. Lightweight Blenderm hinge tape was also used to keep the torque requirements low. The airplanes use MicroInvent MCA3 and MCA3A actuators for the elevator and rudder. The other equipment included a GB45S7O motor, MicroInvent MCF5030 prop, MINOR receiver/ESC, 15 mm fiberboard wheels, and a LIPO85-P battery. All of the micro-radio control hardware listed was obtained from [1]. The battery and electronics weigh a total of approximately 10 g, i.e. about half the weight of the airplane.

The flight performance of the vehicles used in this testing could be described as relatively poor in comparison with actual fighter aircraft. However, they appear to be a good platform for testing the viability of combat planning methods as it would apply to larger combat UAS. Despite the small size and slow flight speeds of the models, the rate at which interaction happens between aircraft is considerably higher for these model aircraft than for full sized fighter aircraft engaged in combat. Essentially, real-time planning for the micro-UAS requires faster computation than may be required with a larger combat UAS. To consider the time scales that full size aircraft operate at in combat, we can use the flight performance data of an F-4U aircraft, as it is readily available at [11]. This gives the sustained turn rate of an F-4U at a medium altitude of 20,000 ft MSL to be 14°/s. This equates to a complete 360° turn in 25 s, suggesting that during a typical 2-circle fight the aircraft would meet with high `AA` every 25 s. It can reasonably be argued that the UAS of the future may significantly out maneuver the outdated F-4, thus requiring a control algorithm to perform at a higher rate. However, the time scale of the micro-UA's is approximately an order of magnitude faster. Considering the limit on bank angle of $\phi_{\max} = 23°$ placed on the blue aircraft, this produces a turn rate of:

$$\dot{\psi} = \frac{g\ \tan(\phi_{\max})}{V} = \frac{9.81\ m/s^2\ \tan(23°)}{2.5m/s} = 95°/s \tag{3.1}$$

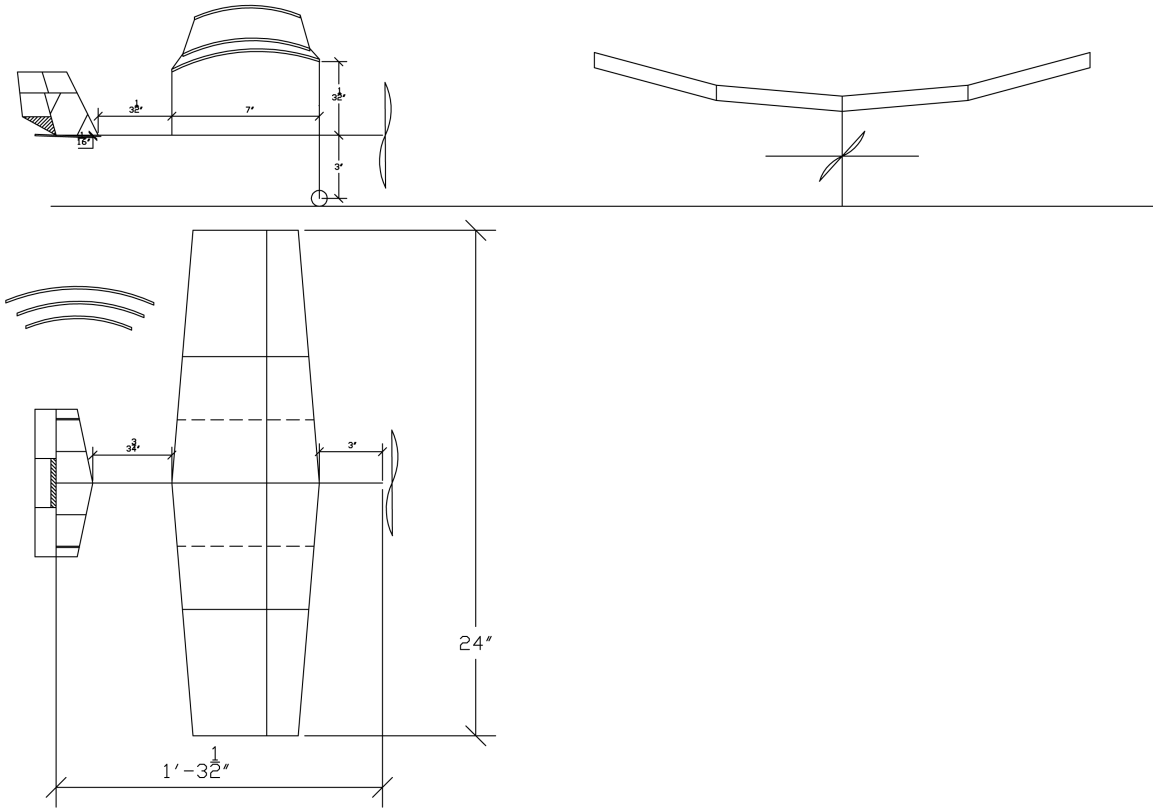


Figure 3-1: 3-View drawing of Micro-UAS design used in flight testing.

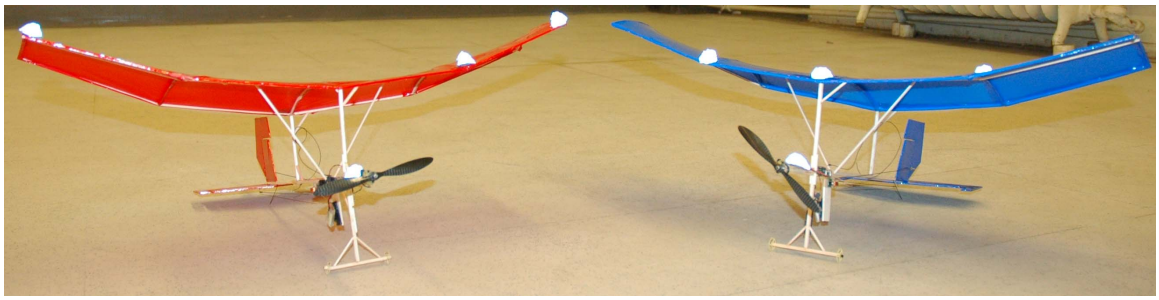


Figure 3-2: Micro-UAS designed for Real-time indoor Autonomous Vehicle test Environment (RAVEN).

At 95°/s a complete 360° requires only 3.8 s, suggesting that the vehicles maneuvering within the RAVEN will be operating at much faster time scales than what might be required in actual combat situations. The combat flight results in Section 3.3 confirm these results. There are, of course, many differences between the test platform used here and actual combat aircraft, particularly, in the precision of maneuvering and agility. However, the micro-UAS should provide a sufficient platform to prove the planning methods are capable of being scaled to real-world situations.

3.2 Flight Control and Trajectory Following

To enable the micro-UAs to fly autonomously within RAVEN, both low-level controllers and a waypoint tracking algorithm needed to be implemented. This section describes the process taken to develop these systems.

3.2.1 Flight Controller

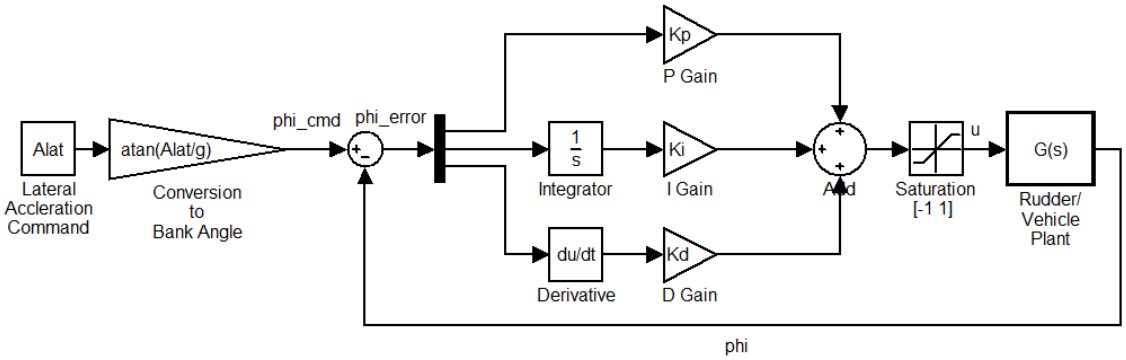
The aircraft is easily capable of flight within the confines of RAVEN. Initial flight testing was done with a human pilot at the controls, and the data was recorded while the aircraft flight characteristics were evaluated by hand control flight. The RAVEN data included control inputs, position and orientation data, and associated rates sampled at 100 Hz. The roll and pitch response of the aircraft was assumed to be decoupled. Based on the slow speeds and slow rotation rates of the aircraft, these proved to be acceptable assumptions. Additionally, the velocity response to throttle input was also assumed to be independent. Each was modeled as a linear system with a time delay. Comparing the control input with vehicle response showed an overall time delay of $\tau = 0.14$ s. This delay takes into account: computer processing, transmitter, receiver, servo mechanical, and aerodynamic delays. Each system was then modeled as $G(s) = e^{-s\tau}C$; where C is a constant chosen to match the model output with the experimental data. These simple models were used to determine initial gains for the aircraft control system.

A Matlab® Simulink representation of the control system used is presented in

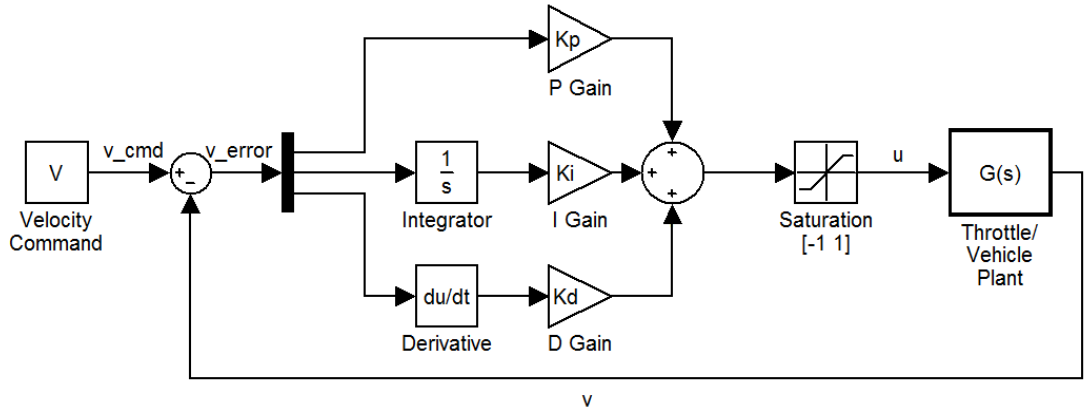
Figure 3-3. Aircraft control was divided into three separate systems to enable altitude, lateral acceleration (A_{lat}), and velocity reference signal tracking. Each system was assumed to be decoupled from the others. The velocity controller uses a proportional–integral–derivative (PID) controller to track the commanded velocity. The A_{lat} controller (see Figure 3-3(a)), is essentially a bank angle (ϕ) controller. The trajectory planner will be designed to output the current reference A_{lat}. Due to the assumption of level, coordinated flight, A_{lat} can be easily converted into a ϕ using Equation 2.4. The controller is then a PID designed to track this reference ϕ by producing a rudder control (u_{rud}) output.

The altitude controller is designed as a PD pitch controller inside a PID altitude controller (see Figure 3-3(c)). The reference signal from the planner is an altitude. A PID structure is used to determine a required pitch to maintain this altitude. A PD structure is then used to command the elevator deflection (u_{elev}) required to achieve the pitch command. For the relatively constant altitude commands required for most flight testing, this structure was successful in providing sufficient altitude tracking. Figure 3-4 shows the altitude plot of an aircraft tracking 1.7 m altitude. Based on the sensitivity of these extremely light planes to gusts (including their own propeller wash on subsequent circles), this is deemed to be adequate performance.

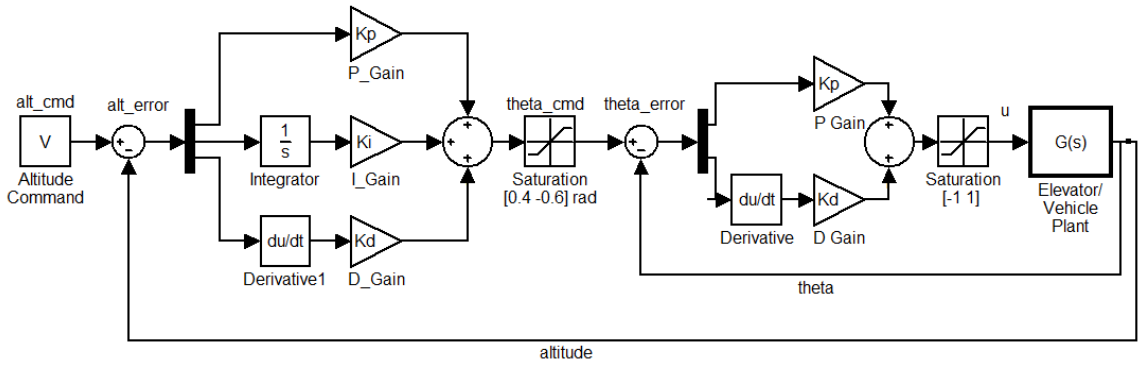
The high fidelity RAVEN platform makes implementing a vehicle controller exceptionally easy. The aircraft flight data, which is returned from the Vicon system at 100 Hz, is simply fed into the controller to produce the control signals. The computed gains were programmed into the flight computer initially; some adjustments were made experimentally during flight testing until acceptable tracking performance was achieved. The gains used are listed in Table 3.1. The control signal sent to the vehicle was normalized to the range $[-1, 1]$, with the extremes representing the limits of travel of the aircraft servos and speed control. A saturation function was included in each controller to ensure this limit was not exceeded. The following section discusses the flight performance results using the autonomously controlled micro-UAS aircraft.



(a) Aircraft roll controller.



(b) Aircraft velocity controller.



(c) Aircraft altitude controller.

Figure 3-3: Controllers used for reference signal tracking on the micro-UAs. The bank angle, altitude, and speed controllers were assumed to be decoupled systems.

Table 3.1: Micro-UAS Controller Gains.

Roll		Altitude		Pitch		Velocity	
K_P^ϕ	1.50 $\frac{u}{rad}$	K_P^{alt}	0.30 $\frac{u}{m}$	K_P^θ	0.70 $\frac{u}{rad}$	K_P^ϕ	0.7 $\frac{u\ s}{m}$
K_I^ϕ	0.05 $\frac{u}{rad\ s}$	K_I^{alt}	0.05 $\frac{u}{m\ s}$	K_I^θ	0.00 $\frac{u}{rad\ s}$	K_I^ϕ	0.15 $\frac{u}{m}$
K_D^ϕ	0.02 $\frac{u\ s}{rad}$	K_D^{alt}	0.05 $\frac{u\ s}{m}$	K_D^θ	0.20 $\frac{u\ s}{rad}$	K_D^ϕ	0.3 $\frac{u\ s^2}{m}$

3.2.2 Trajectory Follower

A waypoint follower, as described in [8], was used to produce the A_{lat} guidance for the micro-UA. This algorithm generates a continuous trajectory between waypoints. It uses the trajectory tracking algorithm presented in [20] to follow this trajectory. During flight testing the L_1 length used to track the trajectories was varied. Shorter values of L_1 allow for more aggressive trajectories to be followed. However, due to the inherent delay in the system it was discovered that too short of a value creates a tracking instability. The value of L_1 found to be most effective was 1.6 m.

Initially, the aircraft was commanded to track a circular trajectory with a radius of 1.8 m. This is a relatively simple tracking task, as the required A_{lat} becomes relatively constant once flight is established on the circle. A plot of tracking over 160 s of flight is shown in Figure 3-5(a). Slight deviations left and right of the course can be seen from lap to lap. As mentioned previously, the aircraft is susceptible to very small gusts. Although the air conditioning units in the RAVEN are powered down during the flight tests, the air disturbed on previous laps by the airplane are enough to cause a noticeable flight disturbance the next time by. Overall, good tracking of the circular trajectory was observed. The aircraft in Figure 3-5(a) was tracking a 3.67 m diameter circle. Over the course of the 165 s flight, the aircraft tracked the circle with a standard deviation of 0.22 m.

During air combat the aircraft needs to be able to roll quickly and reverse the turn. The ability of the aircraft to track a more complicated trajectory, requiring both left and right turns, is shown in Figure 3-5(b). This shows the aircraft tracking a figure-8 shaped trajectory. The aircraft is flying in a counter- clockwise direction on the lower portion, and clockwise around the top portion of the figure-8. Some deviation

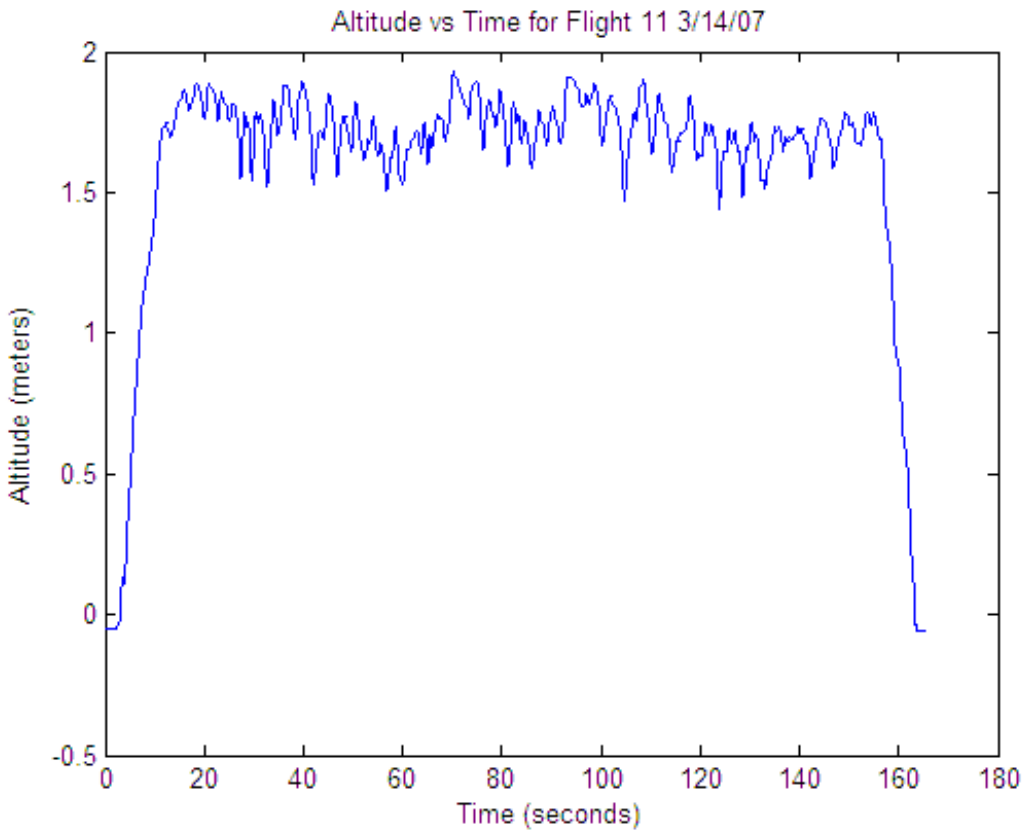
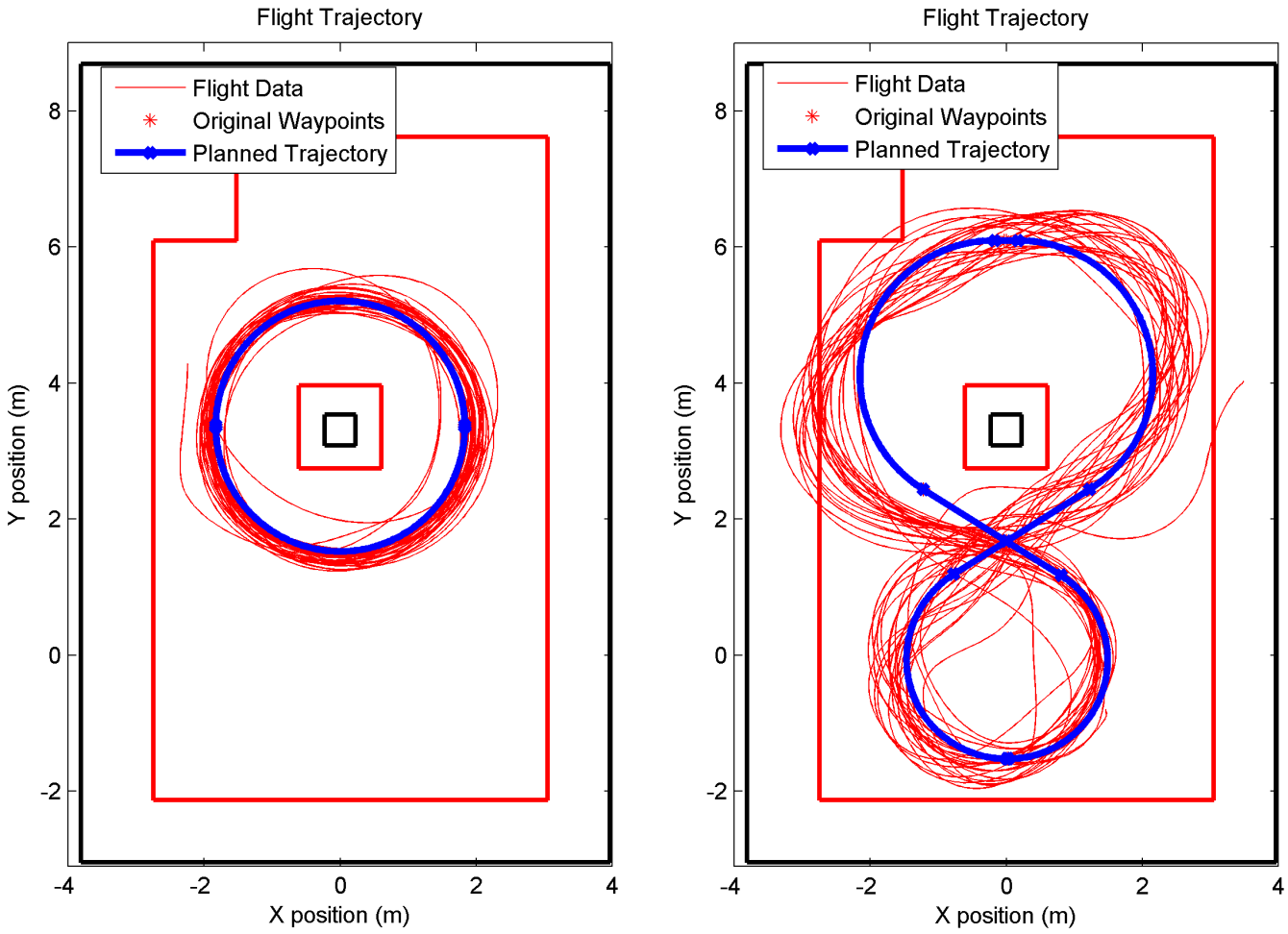


Figure 3-4: Plot of altitude controller performance while tracking a reference altitude of 1.7 m.

is seen, particularly in the upper left-hand side of the plot. This is primarily due the limited roll rate of the vehicle (approximately 45°/s). After completing the tight left-hand turn at the bottom of the plot, the aircraft must quickly roll out and initiate a right-hand turn to complete the top part of the figure-8. The inherent delay in the trajectory tracking algorithm with a $L_1 = 1.6$, combined with the limited roll rate of the aircraft, creates this deviation. The aircraft maneuvering performance, however, proved sufficient to proceed with air combat flights.

3.3 Air Combat Flight Testing

Implementing the planning algorithm presented in section 2.2 required coding the algorithm in C++ and running it on the vehicle computers along with the vehicle



(a) Circular Trajectory Tracking.

(b) Figure-8 Trajectory Tracking.

Figure 3-5: Flight path of micro-UA in left hand circular orbit. This stable platform was used as a target aircraft during flight tests. Also shown is a flight path of micro-UA tracking a figure-8 pattern, demonstrating the aircraft's ability to follow an arbitrary flight path within the confines of RAVEN.

controllers. The time-step utilized for the flight testing ranged from 0.25 s to 0.5 s. This limited the look ahead search to 6 steps and 8 steps, respectively, based on the available computation time. This produced total look ahead times and control sequences for future actions between 1.5 and 4 s in length.

One important change to the planning architecture needed to be made due to the realities of real-time implementation. The flight simulations performed in Chapter 2 assumed that planning could be performed and executed instantly; the aircraft would wait in the current state for the next control action to be determined by the planner, then execute said action. During real-time flight, the aircraft continue to maneuver during the time required to produce the next control action. Typically, the algorithm would take 0.25 s to produce a maneuvering set, $\bar{u}_b$. The aircraft response delay is $\tau = 0.14$ s. This means that the decision made by the planner does not actually have any effect on the system until 0.39 s after the current state is sent to the planner. To compensate for this, the current state, x, is passed to Algorithm 1 to create a the predicted future state 0.39 seconds in the future. This future state is then passed to Algorithm {alg:basicsim. This allows the decision made to take effect in approximately the state it was computed for. Without this compensation, the returned control action would be executed more then one time-step too late.

The ability to avoid colliding with walls and the other aircraft was also incorporated into the real-time planning software. During the planning algorithm search, the planner is attempting to optimize the scoring function at the end of the planning horizon. A large penalty (100) was added to the value returned by the scoring function if the predicted path intersected a wall or the other aircraft. Based on the limited roll rate of the aircraft, a 4 s look ahead was required to significantly reduce collision avoidance between aircraft. Of course, due to the stochastic nature of actual model aircraft, gusts and other disturbances ensure that collisions are still possible.

Using Algorithm 3 the current control action was converted into a bank angle. A_{lat} was computed using Equation 2.4 and passed along to the roll controller (Figure 3-3(a) as a reference signal for the blue aircraft. At each controller time-step ($\Delta t = 0.02$ s), the commaned bank angle is computed per Algorithm 3. For the air combat flight

Algorithm 3 Bank angle command determination given u

Input: u
$\Delta t = 0.02$, $\dot{\phi}_{\max} = 45°/\text{s}$
$\phi^{red}_{\max} = 18°$, $\phi^{blue}_{\max} = 23°$
if $u = left$ **then**
  $\phi_{cmd} = \phi_{cmd} - \dot{\phi}\Delta t$
  if $\phi_{cmd} < -\phi_{\max}$ **then**
    $\phi_{cmd} = -\phi_{\max}$
  end if
else if $u = right$ **then**
  $\phi_{cmd} = \phi_{cmd} + \dot{\phi}\Delta t$
  if $\phi_{cmd} > \phi_{\max}$ **then**
    $\phi_{cmd} = \phi_{\max}$
  end if
end if
Output: ϕ_{cmd}

tests, the red aircraft was commanded to take off and fly in a continuous left hand circle, maintaining approximately $\phi_{\max} = 18°$ while tracking a circular trajectory. The blue aircraft then took off and was required to maneuver to the position of advantage behind the red aircraft. This simple form of air combat is used in the initial phase of training for human pilots. While the target aircraft maintains a constant turn, the student pilot is required achieve a position of advantage using pursuit curves and basic maneuvers such as high and low yo-yos [21]. Using this simple exercise for evaluation, the flight tests demonstrated that the blue aircraft was capable of making good maneuvering decisions and achieving and maintaining an offensive stance. A photograph of the micro-UAs engaged in combat can be seen in Figure 3-6 in MIT's RAVEN.

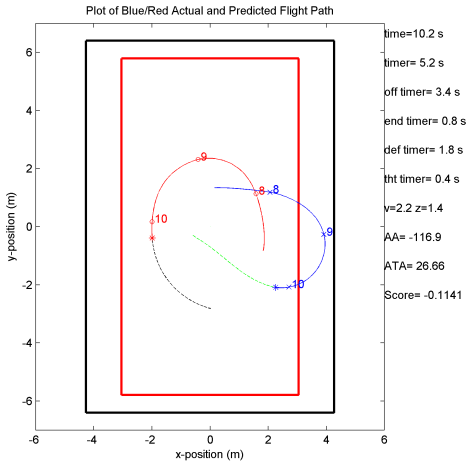
The algorithm development and initial testing was completed in RAVEN. However, combat flight tests were performed at Boeing's Vehicle SWARM Technology Lab in Seattle, WA. The larger facility allowed the aircraft slightly more room to maneuver. Flight tests were performed using both the baseline and expert-modified scoring functions. The actual flight testing demonstrated both scoring functions were capable of generating useful maneuvering decisions in real time. However, the results indicate that there is little difference between the scoring functions during actual im-

Figure 3-6: Micro-UAs engaged in Basic Fighter Maneuvering (BFM) during flight test.

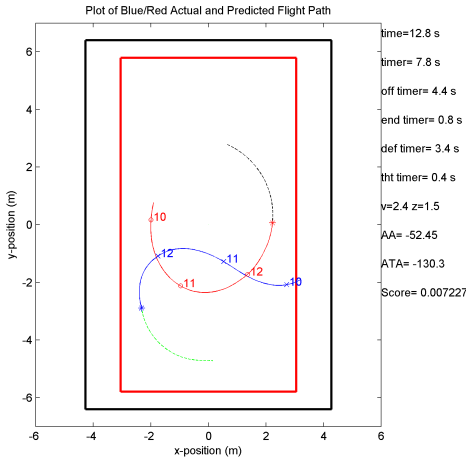
plementation. The distinction between the different scoring functions was obscured by the uncertainties involved with actual flying vehicles.

Despite the difficulties that actual flight tests present good combat maneuvering behavior was observed. For example, Figure 3-7 shows a representative flight sequence in which the blue and red aircraft have a relatively neutral merge in Figure 3-7(a). This is a typical *1-circle* fight, where both aircraft are essentially on opposite sides of the same turn circle. The blue aircraft manages to outperform the red aircraft and gain the opportunity to reverse the turn in Figure 3-7(b). This produces what can be described as a *2-circle* fight, in which the red and blue aircraft are each flying on their own turn circle. Typically these circles are offset as in Figures 3-7(c)–3-7(e). The blue aircraft used the available performance advantage to align the turn circles and has achieved a very offensive position in Figure 3-7(f).

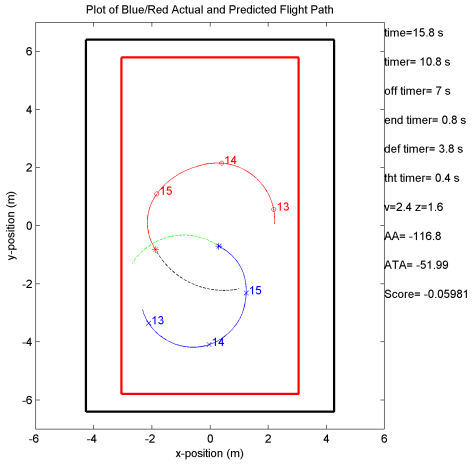
However, as the overall summary values suggest, not all of the behavior seen in



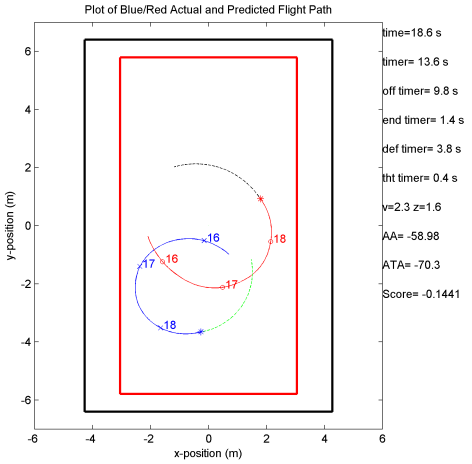
(a) History trails at 10.2 s



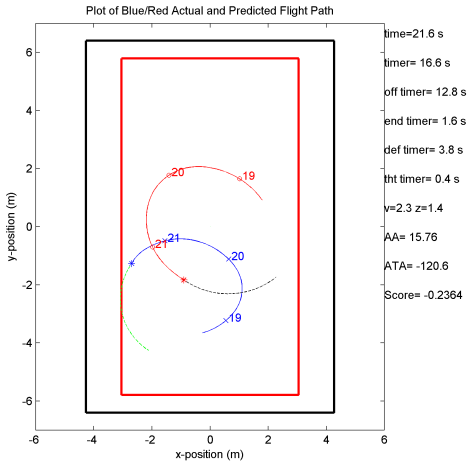
(b) History trails at 12.8 s



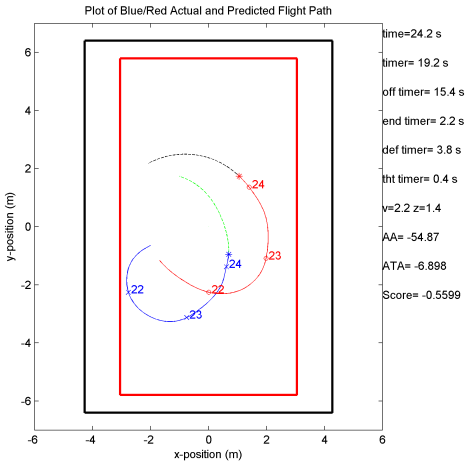
(c) History trails at 15.8



(d) History trails at 18.6

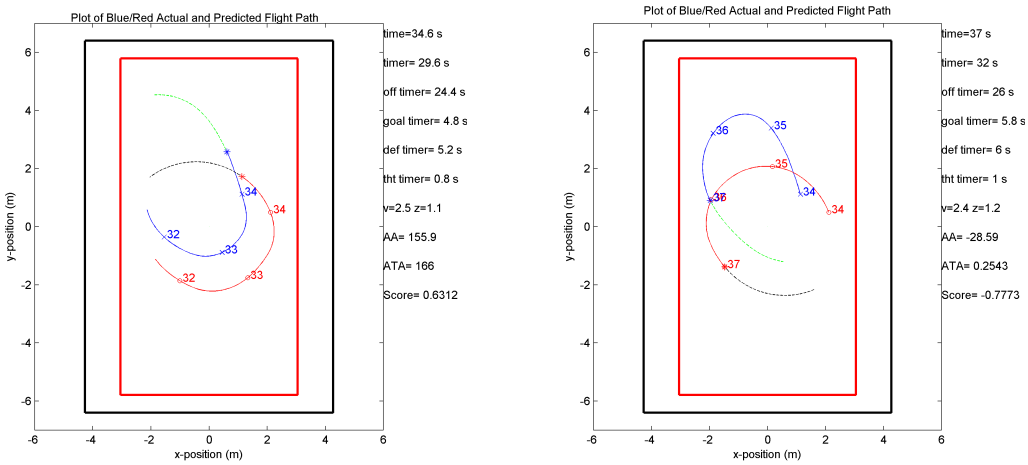


(e) History trails at 21.6



(f) History trails at 24.2

Figure 3-7: Basic Fighter Maneuvering flight test reconstruction. The red and blue lines represent history trails marked with 1 s intervals for the red and blue aircraft, respectively. The green and black dashed lines represent the current planned path for the associated aircraft.



(a) History trails at 34.6 s (b) History trails at 37 s

Figure 3-8: Flight path history trails from flight testing. This is representative of an observed maneuvering error which is attributed to using an *assumed* red maneuvering policy of left hand turns in the predictive algorithm.

flight tests was good. The blue aircraft performed one notably bad maneuver several times throughout the flight testing. The search algorithm was using a left turning assumed red policy. This was selected since it was known that the red aircraft would be continuously turning left for the flight. However, this had a negative effect on the maneuvers chosen by the blue aircraft, as shown in Figure 3-8. This sequence of maneuvering shows the blue aircraft starting in a defensive position in front of the red aircraft. The appropriate response to this situation would be turn make a sharp defensive turn to minimize exposure to the threat zone ($|\mathtt{AA}| > 150°$). Instead, the blue aircraft extends out in front of the red aircraft, then turns hard left rolling out in the position of advantage. While this is certainly not an advisable technique when faced with an enemy behind, the blue aircraft found a maneuver that was best based on the information it was given. The lesson learned here, is that a different assumed red policy (such as the multi-policy) would have been more effective in encouraging realistic behavior, since the blue aircraft would have expected more aggressive behavior from the adversary.

A summary of the performance of the baseline and expert-modified scoring functions can be seen in Figure 3-9. The algorithms are compared using the total accumulated time in four different zones: offensive, defensive, goal and threat. Offensive

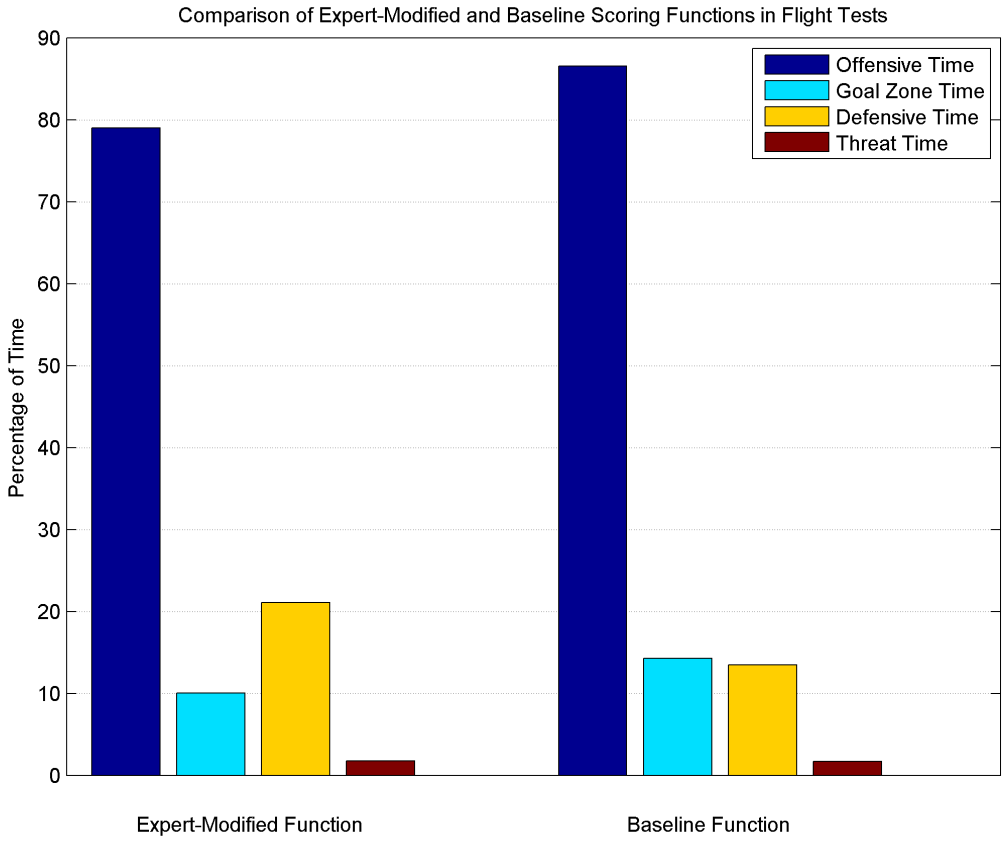


Figure 3-9: Results from flight tests. Bars represent percentage of total engagement time spent in those zones. Offensive time is accumulated when $S_A < 0$ (see Equation 2.1). Conversely, defensive time is accumulated when $S_A > 0$. Goal zone time is accumulated when the aircraft is in the goal zone shown in Figure 2-2. Threat time is accumulated when the red aircraft is essentially pointing at the blue aircraft $|\texttt{AA}| > 150°$.

time is accumulated when $S_A < 0$ (see eq. 2.1). Conversely, defensive time is accumulated when $S_A > 0$. Goal zone time is accumulated when the aircraft is in the goal zone shown in Figure 2-2. Threat time is accumulated when the red aircraft is essentially pointing at the blue aircraft $|\texttt{AA}| > 150°$. The information was compiled from approximately 600 s of combat data. There is relatively little difference between the algorithms. The majority of time was spent offensive, with approximately 10% of the total time spent in the goal zone. Approximately 15% of the time was spent defensive, with only 2% of the time spent in the threat zone.

3.4 Chapter Summary

The results in this chapter demonstrate the validity of the scoring functions at representing the value of a particular combat state. The functions were used with an algorithm that searched up to 4 s into the future to maximize the scoring function return. Computational limitations restricted using this particular method for look-ahead times much larger than those presented. The next chapter describes how the problem is reformulated as a neuro-dynamic program, thus, allowing for much larger effective search look-ahead times. The scoring functions described in this chapter are still necessary; their use is explained in the next chapter as well. However, the search method implemented in this chapter is not used further in this thesis.

Chapter 4

Neuro-Dynamic Programming Method

Dynamic programming (DP) provides the means to precisely compute the optimal maneuvering strategy for the blue aircraft in the proposed air combat game. The resulting strategy or *policy* provides the best course of action given any game state, eliminating the need for extensive on-line computation. While DP has many good qualities, it quickly becomes computationally prohibitive as the problem size increases. This problem arises from the need to solve a DP over a very large set of discrete states. This chapter introduces DP with an example problem and motivates the need for an approximate solution. A method of computing an approximate solution is also introduced, followed by a detailed explanation of how it is applied to the air combat problem.

4.1 Example Problem and Nomenclature

The shortest path DP problem shown in Figure 4-1 will be used to define some of the terminology and methods used in the following sections. For reference, the terminology is summarized in Table 4.1. In the example problem, a robot is capable making a one step move within the 4×4 grid at each time-step, i. The robot is allowed actions $u \in \{up, down, left, right\}$. The location of the robot is defined by

Table 4.1: Symbols used for approximate DP architecture.

Nomenclature	
x	state vector
x_i	state vector at time or time-step t
x^n	n^{th} state vector in X
x_{term}	special terminal state
x_b^{pos}	x coordinate in $x-y$ plane for blue aircraft
y_b^{pos}	y coordinate in $x-y$ plane for blue aircraft
X	vector of states $[x^1, x^2, \ldots, x^n]^T$
$f(x,u)$	state transition function
$\pi(x)$	maneuvering policy
$\pi^*(x)$	optimal maneuvering policy
$\bar{\pi}(x)$	maneuvering policy generated via rollout
$J(x)$	future reward value of state x
$J^k(x)$	k^{th} iteration of $J(x)$
$J^*(x)$	optimal value of $J(x)$
$J_{approx}(x)$	function approximation form of $J(x)$
$\hat{J}(x)$	scalar result of Bellman backup on state x
$\hat{J}(X)$	$[\hat{J}(x^1), \hat{J}(x^2) \ldots \hat{J}(x^n)]^T$
T	Bellman backup operator
γ	future reward discount factor
u	control or movement action
$\phi(x)$	feature vector of state x
Φ	$[\phi(x^1), \phi(x^2), \ldots, \phi(x^n)]^T$
β	function parameters vector
$g(x)$	goal reward function
$g_{pa}(x)$	position of advantage goal reward function
p_t	probability of termination function
$S_{bl}(x_b)$	baseline scoring function evaluated for blue

Algorithm 4 State Transition Function $f(x_i, u_i)$

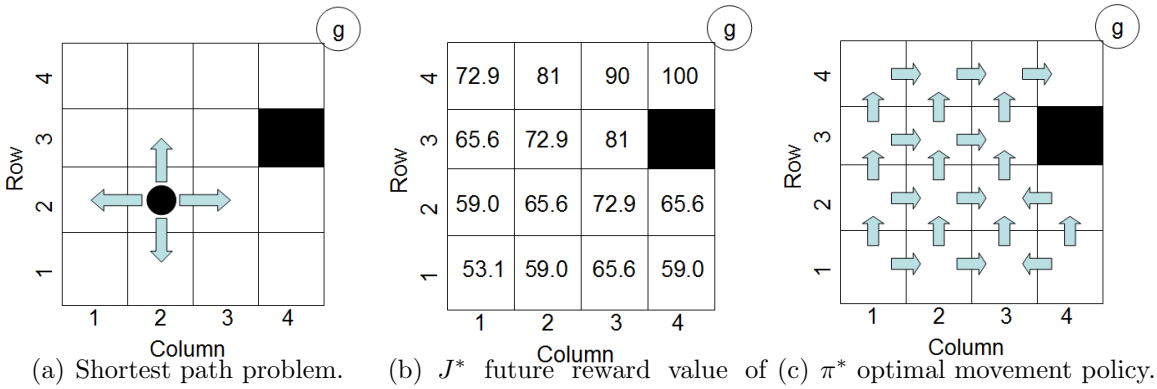
Input: $\{x, u\}$
$[row, col] = x$
if $u = (up)$ and $row \neq 4$ **then**
  $x_{t+1} = x_t + [1, 0]$
else if $u = (down)$ and $row \neq 1$ **then**
  $x_{t+1} = x_t + [-1, 0]$
else if $u = (left)$ and $row \neq 1$ **then**
  $x_{t+1} = x_t + [0, -1]$
else if $u = (right)$ and $row \neq 4$ **then**
  $x_{t+1} = x_t + [0, 1]$
else
  $x_{t+1} = x_t$ (to enforce walls)
end if
if $x_{x+1} = [3, 4]$ **then**
  $x_{t+1} = x_t$ (to enforce blocked square)
end if
Output: (x_{i+1})

the $[row,\ column]$ coordinates in the state vector $x_i = [row_i,\ col_i]$. Additionally, x^n refers to the n^{th} state vector when referring to a set of states $X = [x^1, x^2, \ldots, x^n]^T$. A state transition function $f(x, u)$ is defined which computes the next state of the game given a certain control action. The state transition function executes the dynamics of movement and enforces the limitations of the game (i.e., the robot can not move outside the grid or to the blocked square), see Algorithm 4.

The objective is to compute a movement strategy that results in the optimal path to reach the goal. The goal, which is accessible only from square (4,4), provides a reward for success and is defined by the function $g(x)$:

$$g(x) = \begin{cases} \text{if } x = [4, 4], & g = 10 \\ \text{else} & g = 0 \end{cases} \tag{4.1}$$

A function $J(x)$ is defined at each state representing the future reward value of that state. The initial value of $J(x)$ is set to zero, such that $J^0(x) = 0$ for all x. The optimal future reward function, $J^*(x)$ can be computed by repeatedly performing a Bellman backup [14] on each state. This optimal *value* will be referred to as $J^*(x)$.



(a) Shortest path problem. (b) J^* future reward value of each state for $g = 0$ and $\gamma = 0.9$. (c) π^* optimal movement policy.

Figure 4-1: Example shortest path problem solved using dynamic programming.

An optimal policy, π^* can then be computed from $J^*(x)$ using Equation 4.5. The Bellman backup operator, T is defined as:

$$J^{k+1}(x) = TJ^k(x) = \max_u [\gamma J^k(f(x,u)) + g(x)] \tag{4.2}$$

where $\gamma < 1$ is the discount factor. The discount factor means that future rewards mean less to us than current rewards [6]. This has the effect of encouraging the robot to move toward the reward as quickly as possible. In this example $\gamma = 0.9$ will be used. The vector x can also be replaced by a set of states, X, to accomplish Bellman backup operations on a number of states simultaneously.

After performing multiple Bellman backup operations, $J^k(x)$ converges to the optimal discounted infinite horizon value $J^*(x)$. The number of iterations required depends on the problem design and complexity. The values computed for the example are shown in Figure 4-1(b). Note the future reward of being in the goal square is 100. As an example, the first Bellman backup for state [4,4] is computed as follows:

$$J^o([4,4]) = 0$$
$$J^1([4,4]) = \max_u [\gamma J^o(f([4,4],u)) + g([4,4])]$$

In this case, the u or control action that provides the maximum return from the Bellman backup is either *up* or *right*. This will keep the robot in location [4,4] due to

the wall limitation. State [4,4] is unique that it continues to receive the goal reward at each time-step, thus:

$$\begin{aligned} J^1 &= 0.9 \cdot 10 + 10 = 19.0 \\ J^2 &= 0.9 \cdot 19 + 10 = 27.1 \\ &\vdots \\ J^\infty &= 0.9 \cdot 90 + 10 = 100 \end{aligned}$$

The square to the immediate left of the goal, [4,3], has an optimal discounted future reward of 90. This is because the square is one step away from the goal region and $g(x \neq [4,4]) = 0$. A robot located in this square will not receive a reward until it moves one square right in the first time-step. The future reward for location [4,3] becomes:

$$\begin{aligned} J^\infty([4,3]) &= \max_u [\gamma J^\infty(f([4,3],u)) + g([4,3])] \\ &= \gamma J^\infty([4,4]) + g([4,3]) = 0.9 \cdot 100 + 0 = 90 \end{aligned} \tag{4.3}$$

For the next state to the left, [4, 2]:

$$J^\infty([4,\ 2]) = \max_u [\gamma J^o(f([4,2],u)) + g(x)] = 0.9 \cdot 90 + 0 = 0.9^2 \cdot 100 + 0 = 81 \tag{4.4}$$

The process continues for all states with the future reward decreasing for each step away from the goal. The optimal future reward function J^* can then be used to derive the optimal policy π^*. Where the optimal action at time-step i is defined as:

$$\begin{aligned} u_i &= \pi^*(x_i) = u \in \{up,\ down,\ left,\ right\} \\ &= \arg\max_u [g(x_i) + \gamma J^*(f(x_i,u))] \end{aligned} \tag{4.5}$$

The policy π provides the shortest path move from any given state. The computed $J^*(x)$ for the example problem is shown in Figure 4-1(b) and Figure 4-1(c) shows the resulting optimal policy π^*. Note that while $x = [2,4]$ has only one optimal action,

all other states have multiple optimal actions. Due to the absence of obstacles in most of the grid, many squares have two possible actions that lead to a shortest path to the goal.

This discrete two dimensional path planning problem has very few states. Unfortunately, the The required number of discrete states for typical real-world problems make exact DP impractical. Consider the simplified air combat problem described in chapter 2. We desire to define a given state of the game with the following vector

$$x = [x_b^{pos}, y_b^{pos}, \psi_b, \phi_b, x_r^{pos}, y_r^{pos}, \psi_r, \phi_r]^T \tag{4.6}$$

where the components x^{pos} and y^{pos} are coordinates expressed in meters; ψ and ϕ are the aircraft heading and bank angle, respectively. Subscript b indicates the blue aircraft, subscript r represents the red aircraft.

In order to solve this problem as a DP, the entire state space needs to be discretized into unique points within an 8-dimensional space. Assume the flight area is defined as a 10×10 m space, the bank angle is limited to $\pm 25°$ and the heading is between 0° and 360°. Dividing each component into 10 discrete intervals results in the following increments: 1 m for position, 5° for bank angle and 36° for aircraft heading. This is arguably a coarse resolution that would most likely not be capable of representing the problem sufficiently to produce useful results. Even so, this simple problem representation would have $100,000,000$ discrete states. A finer resolution would have many more states, thus requiring an unacceptably long time to complete the required number of Bellman backup operations. This "curse of dimensionality" [5] motivates the use of an approximate technique.

4.1.1 Approximate Dynamic Programming Example

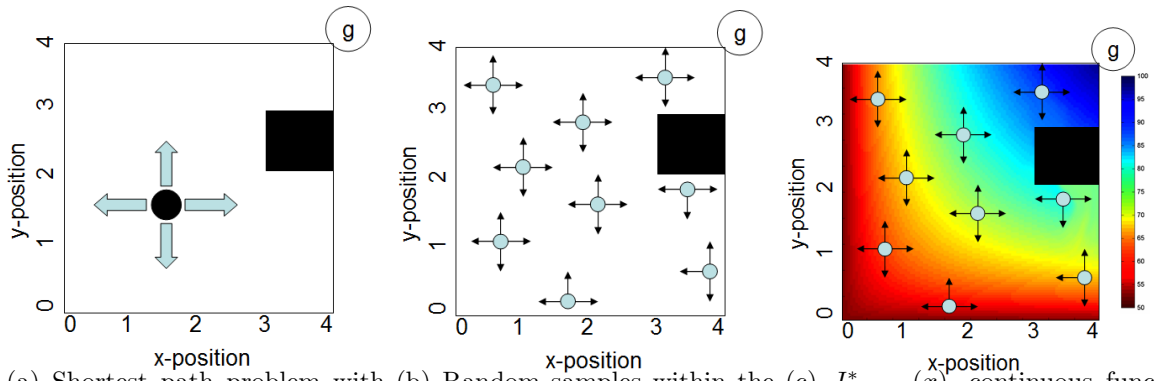
Neuro-dynamic programming is a general term for approximate dynamic programming [6]. A continuous function approximator eliminates the need to represent and compute the future reward for every discrete state. The simple shortest path problem can be redefined with continuous values for the coordinates (see Figure 4-2). The

components of x can now take any value between 0 and 4. $J(x)$, which is essentially a look-up table of values at discrete points, is replaced by $J_{approx}(x)$, a continuous function that can approximate the future reward of a given state. The state transition function $f(x, u)$ is redefined to allow movements from any arbitrary point. To accomplish this, the velocity of the robot, v, is used. The distance traveled by the robot is computed as $v\Delta t$ at each time-step, Δt. $J_{approx}(x)$ is initialized to be 0 at all locations. The state space is sampled with some manageable number of sample states; 9 were selected as shown in Figure 4-2(b). We refer to the set of state samples as X. A Bellman backup operator (T) is applied to each state sample as in Equation 4.2. The resulting values are stored in a target vector, $\hat{J}(X)$.

$$\hat{J}^{k+1}(X) = TJ^{k}_{approx}(X) \tag{4.7}$$

where $\hat{J}^{k+1}(X)$ refers to the set of values produced by a Bellman backup on X. The target vector $\hat{J}^{k+1}(X)$ is used by a linear estimator to produce the future reward function $J^{k+1}_{approx}(X)$. J^{k+1}_{approx} is a linear estimation of the values contained in the target vector $\hat{J}^{k+1}(X)$, and can be evaluated at any arbitrary state. $J^{k+1}(X)$ will be used in the next Bellman backup operation. The linear estimator uses a set of descriptive features to estimate the $\hat{J}^{k+1}(X)$.

A descriptive feature set $\phi(x)$ is computed for each state in X. $\phi(x)$ can contain any number of features. Typically, more than one is required to produce an accurate function approximation, and too large of a number can become computationally expensive to handle. The development and selection of features is discussed in Section 4.6. Each feature contained in $\phi(x)$ produces a scalar value for a given state. Therefore, with n features the $\phi(x)$ vector will contain n scalar values. Features are selected, which contain information related to the future reward of a given state. For example, a reasonable feature for the example problem is the Euclidean distance from the robot to the goal location. The feature sets are computed for all state samples



(a) Shortest path problem with continuous states.

(b) Random samples within the state space. Four actions are possible at each step.

(c) $J^*_{approx}(x)$, continuous function approximation of the future reward value of all states.

Figure 4-2: Example shortest path problem solved using approximate dynamic programming. Once found $J^*_{approx}(x)$ can be used to compute the optimal policy, $\pi^*(x)$.

$x^i \in X$ and stored in Φ so that:

$$\Phi(X) = \begin{bmatrix} \phi(x^1) \\ \phi(x^2) \\ \vdots \phi(x^n) \end{bmatrix} \tag{4.8}$$

The new $J_{approx}(x)$ can now be computed using standard least squares estimation as follows [6]:

$$\beta^{k+1} = (\Phi^T \Phi)^{-1} \Phi^T \hat{J}^{k+1}(X) \tag{4.9}$$

J_{approx} is computed as:

$$J^{k+1}_{approx}(x) \equiv \phi(x)\beta \tag{4.10}$$

Where β are the value function parameters computed in Equation 4.9. The function J^{k+1}_{approx} can now be used to evaluate the future reward of any state x. Additional discussion on this function approximation method can be found in [6].

The resulting function J^{k+1}_{approx} is a continuous function approximating the $\hat{J}^{k+1}(x)$ values An approximation of the true $J^*(x)$ can be generated through repeated Bellman backup operations. Figure 4-2(c) provides a visualization of $J^*_{approx}(x)$ for this example problem. The approximate policy π can then be computed from the resulting $J^*_{approx}(x)$ using Equation 4.5.

This method for solving an approximate DP can be extended to problems with much larger state spaces than that of the example problem. The architecture relieves some of the difficulty that the curse of dimensionality causes in classical DP techniques. The remainder of this chapter explains how neuro-dynamic programming was applied to the air combat game.

4.2 States

The air combat system state x is defined by the position, heading and bank angle as shown in Equation 4.6. There is also a special terminal state x_{term} from which no future rewards can be obtained. The terminal state is used as a way of reinforcing specific air combat behavior (i.e. not flying in front of the adversary), as discussed in Section 4.4. The components of x are continuous. The position of the aircraft x^{pos} and y^{pos} have no limits, thus allowing for flight in any portion of the x–y plane. The aircraft bank angle and heading are allowed to take any value between the specified limits.

As in the shortest path problem example, the air combat game state space was sampled to produce representative states. A higher density sampling produces a better approximation to the optimal solution than a lower density sampling. The limit on the number of points selected was based on the computation time. The amount of time required to execute Bellman backup operations on all points and approximate the results to produce the next $J_{approx}(x)$ increases linearly with the number of states chosen. A sample set, X, of 10^5 points proved to be a reasonable number to use during development and testing. One DP iteration using this set required approximately 60 s.

Due to the limit on the number sampled points, it was important to choose samples wisely. Areas of the state space with a higher density sampling would have a higher fidelity function approximation, $J_{approx}(x)$, and therefore a policy more closely resembling $\pi^*(x)$. To ensure the areas most likely to be seen during combat were sampled sufficiently, points were selected using a trajectory sampling technique. Red

and blue starting positions were selected from a Gaussian distribution with $\sigma = 7$ m. The initial aircraft headings and bank angles were selected from a uniform distribution. From this beginning state a combat simulation was run using the simulation described in chapter 2 and the state of the game was recorded every 0.25 s. The simulation terminated when the blue aircraft reached the goal zone behind the red aircraft. The simulation was initialized again at a randomly generated state. This process continued until all 10^5 points were generated.

A representation of this set of states, X, is shown in Figures 4-3 and 4-4. Each state, x^n, consists of the location and orientation of both aircraft so it is difficult to visualize all of the information in a 2D plot. Figure 4-3 is a plot of all states with the blue and red aircraft positions plotted on the x–y plane. The initial positions of the individual aircraft can be seen at the edges before they turn toward their adversary and begin turning in an engagement. Some of the circles flown during combat can also be distinguished at the edges. Note that the highest density of states is near the origin, which is where most maneuvering takes place.

Figure 4-4 shows two plots of the same set X. In these plots the red aircraft is shown at the origin, the relative blue aircraft position is then plotted. In Figure 4-4(a) the blue location appears to be similar to a Gaussian distribution relative to the red aircraft, as the center region has the highest density and tapers off as distance from the red aircraft increases. However, Figure 4-4(b) shows a densely sampled region surrounding the red aircraft. The pattern is a product of the relative aircraft capabilities and maneuvering strategies. However, it is not particularly obvious that this pattern would occur. Furthermore, the likely headings and bank angles of the samples in this dense region would be difficult to produce except through the trajectory sampling method used here.

The data set produced also contains information regarding the assumed red aircraft maneuvering policy. Some maneuvering strategy must be assumed for the red aircraft initially. This can be updated later in future iterations or when a better model of the red maneuvering strategy is determined. The technique developed in [2] was successful at producing realistic maneuvers for adversaries that were part of a

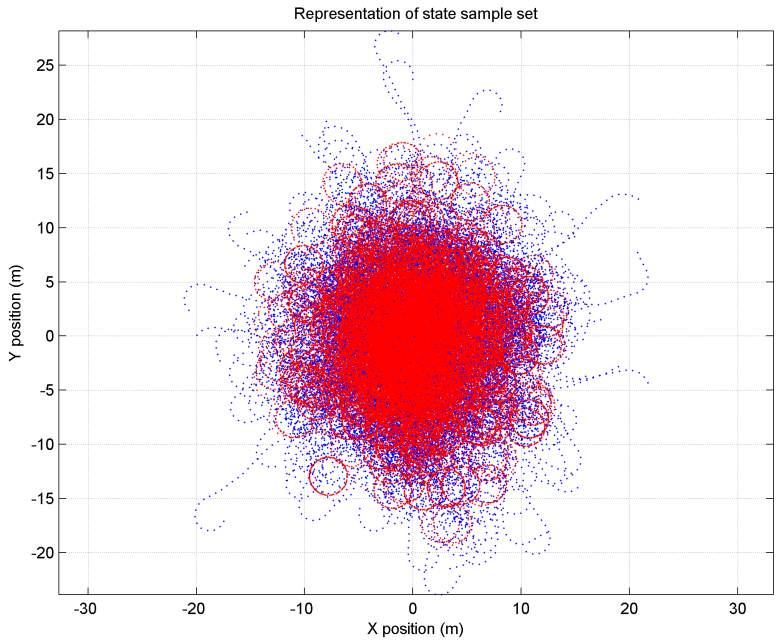
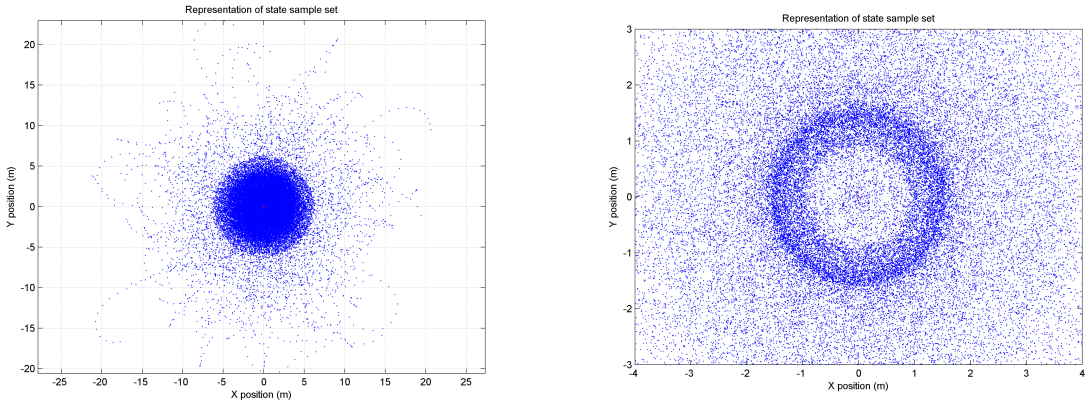


Figure 4-3: Data set of 10^5 state space samples used for approximate dynamic programming. Points represent blue/red aircraft location in the x–y plane. Each sample consists of a pair of aircraft and all the information contained in the state X. The points were generated using a combat simulation from randomly generated initial starting conditions.

flight simulation. A similar strategy is used to produce the maneuvers for the red aircraft in the simulation used in this thesis. This technique computes a $u_r(x)$ at each state using a limited look-ahead minimax search. The minimax search uses the baseline scoring function ($S_{bl}(x)$ from Equation 2.3) to determine the score of some future state. The specific search algorithm used is minimax with alpha–beta pruning as outlined in [19]. The recursive minimax algorithm returns the u_r that maximizes the baseline scoring function $S_{bl}(x)$ at each time-step under the assumption that the blue aircraft will select a u_b that minimizes $S_{bl}(x)$. The minimax search was performed over a 0.75 s receding search horizon, thus giving the red aircraft a relatively short look ahead. The algorithm manages to produce a π_r that was challenging to fly against and allowed the red aircraft to act as a good training platform. The precomputed $u_r(x)$ are used by the following DP formulation to train against and learn an appropriate counter policy, π_b.



(a) Full sample set. (b) Magnified view of center area.

Figure 4-4: An alternative representation of the data points shown in Figure 4-3. The red aircraft is used as the point of reference and is plotted at the origin. The relative location of the blue aircraft is plotted for each point. A dense ring of data points can be seen encircling the red aircraft; this is an apparent common relative orientation of the aircraft. The advantage of sampling states from combat simulation is that it produces a set of points with states likely to be seen in air combat.

4.3 Goal (reward)

The true goal of the blue aircraft is to attain and maintain an offensive position behind the red aircraft. A combination of two goal functions were used to reinforce this behavior. The baseline scoring function, S_{bl} from Equation 2.3 was utilized again. It proved to be a useful representation of the relative desirability of a given state during flight testing and has the added advantage of being a smooth, monotonic function. S_{bl} was modified to return values in the range (0,1), with 1 representing the best possible score for an aircraft in an offensive position. The modified function, S_{bl}^{mod} was computed as follows:

$$S_{bl}^{mod} = \frac{-S_{bl} + 1}{2} \tag{4.11}$$

Additionally, a more specific goal zone was defined as the area between 0.1 and 3 m behind the red aircraft. A position of advantage reward function is defined as $g_{pa}(x)$ as shown in Algorithm 5.

This goal zone, depicted in Figure 4-5, represents the location that a stabilized

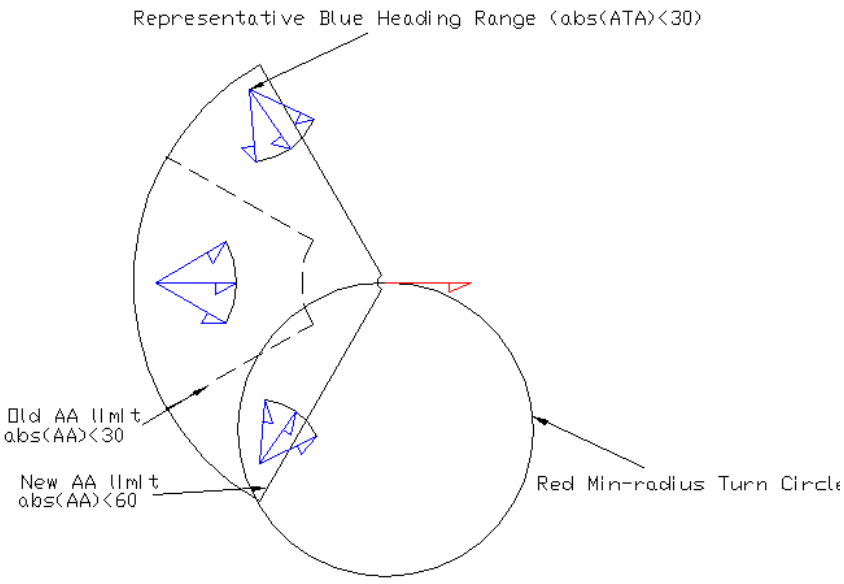


Figure 4-5: Defined position of advantage behind adversary. Maneuvering into this Goal Zone provides a reward to the blue aircraft. It was enlarged from the goal zone used in chapters 2 and 3 to ensure flight on the red aircraft turn circle was rewarded. The minimum allowed range was decreased. A limitation was also placed on antenna train angle (`ATA`), requiring the blue aircraft to be within $\pm 30°$ of pure pursuit to receive the reward.

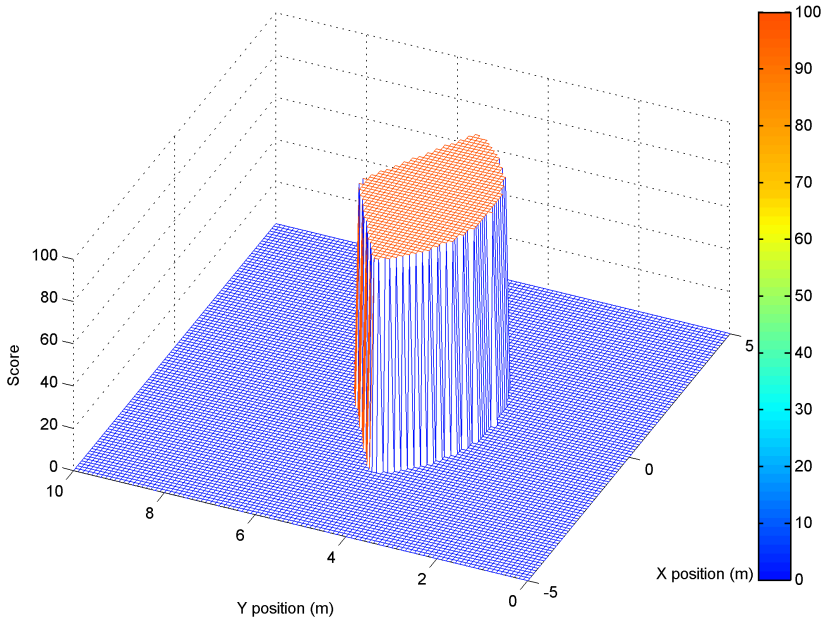


Figure 4-6: Plot of reward function for flight within Goal Zone.

Algorithm 5 Goal Reward Function $g_{pa}(x)$

Input: $\{x\}$
R =Euclidean distance between aircraft
if $(0.1\ m < (R) < 3.0\ m)$ & $(|\texttt{AA}| < 60^{\circ})$ & $(|\texttt{ATA}| < 30^{\circ})$ **then**
  $g_{pa}(x) = 1.0$
else
  $g_{pa}(x) = 0.0$
end if
Output Reward: (g_{pa})

position of advantage can be maintained regardless of evasive maneuvers, thus, allowing weapons deployment. By rewarding states in the goal zone, the DP should learn a $J_{approx}(x)$ that will guide the blue aircraft toward the defined position of advantage.

The two goal functions were combined to create $g(x)$, which was used in the DP learning algorithm described in Section 4.7:

$$g(x) = \begin{cases} \text{if } \ x = x_{term}, & \text{return } \ -1 \\ \text{else} & \text{return } w_g g_{pa} + (1 - w_g) S_{bl} \end{cases} \tag{4.12}$$

where w_g is a weighting value in the range $[0, 1]$. The value of w_g used was determined experimentally as discussed in Chapter 5. The goal function $g(x)$ is used in Bellman backup operation (Equation 4.13) similar to Equation 4.7. The goal function $g(x_i)$ is now evaluated at $x_{i+1} = f(x, u)$ for all states in set X.

$$\hat{J}^{k+1}(X) \equiv T J^k_{approx}(X) = \max_u [\gamma J^k(f(X, u)) + g(f(X, u))] \tag{4.13}$$

The set of states, X is evaluated by the state transition function inside of $g(\cdot)$. This means that for each x^n_i, the function $g(\cdot)$ is evaluated at x^n_{i+1}, the state of the next time-step given the control action, u is applied to the system. Thus, the g_{pa} component of reward is given to states which are not in the goal zone only *if* an action is selected that causes them to arrive in the goal zone in the following time-step. This modification effectively enlarges the goal zone slightly, while encouraging movement into the zone. The blue aircraft receives the reward in Equation 4.12 for each time-

Algorithm 6 Probability of Termination Function $p_t(x)$

Input: $\{x\}$
if $R < 5.0\ m$ & $\texttt{AA} > 165°$ **then**
  $p_t = 0.1$
else if $R < 5.0\ m$ & $\texttt{AA} > 150°$ **then**
  $p_t = 0.05$
else
  $p_t = 0$
end if
$p_t = p_t + 0.05^{(2R)}$, (add collision hazard penalty)
$p_t = p_t w_p$, (apply weight factor)
Output Probability: (p_t)

step it is in the goal zone and continues to accumulate this reward if it is able to stay in the goal zone. However, states at the edge of the goal zone will not receive the reward from $g_{pa}(\cdot)$ if the selected control action moves the aircraft outside of it.

4.4 Avoid Risk (Negative Rewards)

The complete air combat problem entails balancing risk while attaining rewards. While the reward structure is shown above, the risk is captured in the p_t function which defines the probability (per second) of entering a terminal state x_{term}. For example, $p_t(x)$ (Algorithm 6) would return $p_t \approx 0.1$ for a range=4.5 m and $\texttt{AA} = 180°$. This means that for each 1 s spent in this location, there is a 10% probability that the aircraft will end up in the terminal state. This is intended to represent the real-world situation of flying into the weapons engagement zone of a threat aircraft. As the time spent there increases, so does the chance the adversary will successful employ ordnance.

The blue aircraft's probability of termination (p_t) discourages the blue aircraft from flying through the defined danger zone, plotted in Figure 4-7. An increased p_t increases the probability that the aircraft will enter the terminal state, x_{term}. Upon entering the terminal state, the aircraft receives a one-time reward of $g(x_{term} = -1)$. Once in the terminal state, no future rewards are accrued (i.e. $J(x_{term}) = 0$). This has the effect of reducing the *expected reward*.

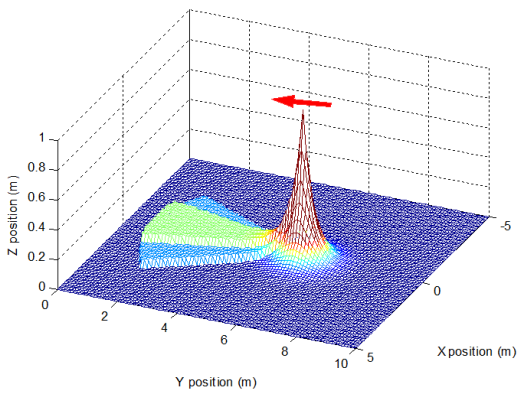


Figure 4-7: p_t function representing areas of increased probability of termination due to threat weapons and aircraft collision.

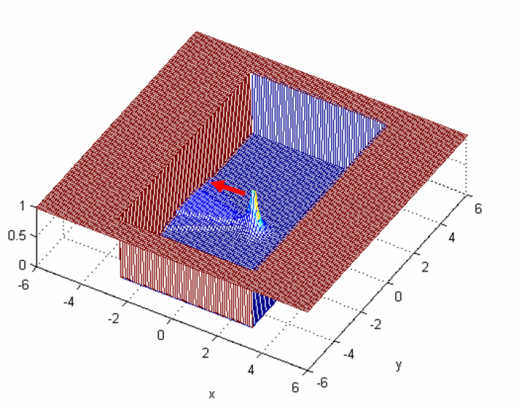


Figure 4-8: p_T function including wall/terrain avoidance in addition to threat weapons and aircraft collision.

To discourage maneuvering that incurs unnecessary risk, equation 4.13 is modified to utilize the probability of termination at the current time-step to compute the expected future reward.

$$\begin{aligned} \hat{J}^{k+1}(X) &\equiv T_{pt} J^k_{approx}(X) \\ &= \max_u \left[(1 - p_t(X))(\gamma J^k_{approx}(f(X,u))) + g(f(X,u)) - g(x_{term}) p_t(X) \right] \end{aligned} \tag{4.14}$$

The operator T_{pt} is defined to represent the Bellman backup process when p_t is included in the computation. Note that if the current state results in $p_T > 0$, then the blue vehicle is considered to be at risk, which is captured in the cost by reducing $J^{k+1}(X)$ through the RHS terms $\{1 - p_t(X)\}$ and $-Rp_t(X)$. This should encourage control actions that quickly get the blue vehicle out of harms way.

An additional use for p_t can be to create terrain avoidance. Figure 4-8 expands the p_t definition to include the walls of the RAVEN, where $p_t = 1.0$. This capability was not used in this thesis, but could have potential for incorporating terrain avoidance in the computation of π.

Algorithm 7 State Transition Function $f(x_i, u_b, u_r)$ (Simulates 0.25 s, given red and blue actions)

Input: $\{x, u_b, u_r\}$
$\Delta t = 0.05$, $v = 2.5\ m/s$, $\dot{\phi}_{\max} = 40°/\text{s}$
$\phi^{red}_{\max} = 20°$, $\phi^{blue}_{\max} = 25°$
for i=1:5 (once for each 0.05 s time increment) **do**
  for both red and blue aircraft **do**
    if $u = 1$ **then**
      $\phi = \phi - \dot{\phi}$
      if $\phi < -\phi_{\max}$ **then**
        $\phi = -\phi_{\max}$
      end if
    else if $u = 3$ **then**
      $\phi = \phi + \dot{\phi}$
      if $\phi > \phi_{\max}$ **then**
        $\phi = \phi_{\max}$
      end if
    end if
    $\dot{\psi} = \frac{9.81}{v} \tan(\phi)$
    $\psi = \psi + \dot{\psi}\Delta t$
    $x^{pos} = x^{pos} + \Delta t \sin(\psi)$
    $y^{pos} = y^{pos} + \Delta t \cos(\psi)$
  end for
end for
Output: (x_{i+1})

4.5 Control Inputs and Vehicle Dynamics

The micro-UA vehicle dynamics were modeled by a simple state transition function $f(x, u_b, u_r)$. This function takes both red and blue control actions into account and simulates forward one time-step, $\Delta t = 0.25$. The control actions available to both aircraft are $u\epsilon\{roll - left, maintain - bank, roll - right\}$ which is equivalently represented as $u\epsilon\{1, 2, 3\}$. Thus, the aircraft maintains control action u_i for Δt, then executes $u_i + 1$ for Δt, etc. The pseudo-code in Algorithm 7 defines the operation of the state transition function.

4.6 Feature Development

The approximation architecture uses features of the state to estimate the value function. Good features are the key to good estimation. Human decision making gives some insight to the process. Pilots use on-board system information (radar and flight performance instruments) and visual cues to select maneuvers. The same information is encoded as state features (Table 4.2). To produce good value function estimates. The features were developed as follows.

Decisions made during BFM are primarily based on relative aircraft position. The main exception is when terrain, or other obstacles, become a factor. Terrain avoidance is not being considered in the initial formulation in order to keep the state space smaller, but can be included later as an additional penalty function in the dynamic programming formulation. Thus, the air combat decisions will be made solely on relative positions. We can compute the decisions for the blue aircraft, therefore, the convention is to represent the feature information from the perspective of the blue aircraft. The features x_{rel}^{pos} and y_{rel}^{pos} represent the blue aircraft position in relation to the red aircraft. The orientation of the x–y axes remain fixed to the global frame. While this provides some information regarding the current combat situation, combinations with other information are much more useful.

The range, R, is clearly an important tool for assessing the tactical situation. Range coupled with `AA`, `ATA` and `HCA` (see Figure 2-1) provides complete instantaneous information about the current state. For reference, a graphical representation of `AA` can be seen in Figure 4-9. However, the current state change rate is also relevant to maneuvering decisions. $\dot{\texttt{AA}}$ represents the rotation rate of the red aircraft from the perspective of the blue aircraft. $\dot{\texttt{AA}}$ incorporates the adversaries bank angle and turn rate, range and own-ship velocity into one piece of information. $\dot{\texttt{AA}}$ is typically determined visually by a human pilot and is used as an initial indication of an impending adversary aggressive maneuver. It is also part of the information included in the heuristic representation of the turn circle as discussed in in Chapter 2. See Figures 4-10 and 4-11 for a graphical representation of $\dot{\texttt{AA}}$. $\dot{\texttt{ATA}}$ is also known as the line-of-

Table 4.2: Features Considered for Function Approximation

Feature	Description
x_{rel}^{pos}	Relative position on X axis
y_{rel}^{pos}	Relative position on Y axis
R	Euclidean distance between aircraft
v_c	Closure velocity
$\|\|v_{rel}\|\|$	Norm of Relative velocity
θ_c	Closure Angle
$\mathtt{AA}$	Aspect Angle
$\|\mathtt{AA}\|$	Absolute Value of Aspect Angle
$\mathtt{AA}^+$	$\max(0, \mathtt{AA})$
$\mathtt{AA}^+$	$\min(0, \mathtt{AA})$
$\dot{\mathtt{AA}}$	Aspect Angle rate
$\dot{\mathtt{AA}}_{int}$	$10 - \left\|\dot{\mathtt{AA}}\right\|$
$\mathtt{ATA}$	Antenna Train Angle
$\|\mathtt{ATA}\|$	Absolute Value of Antenna Train Angle
$\mathtt{ATA}^+$	$\max(0, \mathtt{ATA})$
$\mathtt{ATA}^+$	$\min(0, \mathtt{ATA})$
$\dot{\mathtt{ATA}}$	Antenna Train Angle rate
$\dot{\mathtt{ATA}}_{int}$	$10 - \left\|\dot{\mathtt{AA}}\right\|$
$\mathtt{HCA}$	Heading Crossing Angle
$\|\mathtt{HCA}\|$	Absolute Value of Heading Crossing Angle
S_A	Combination of $\mathtt{ATA}$and $\mathtt{AA}$(defined in Equation 2.1)
S_R	Desired Range Heuristic (defined in Equation 2.2)
x_b^{pos}	Blue Aircraft x-position
y_b^{pos}	Blue Aircraft y-position
ϕ_b	Blue Aircraft Bank Angle
ψ_b	Blue Aircraft Heading
x_r^{pos}	Red Aircraft x-position
y_r^{pos}	Red Aircraft y-position
ϕ_r	Red Aircraft Bank Angle
ψ_r	Red Aircraft Heading

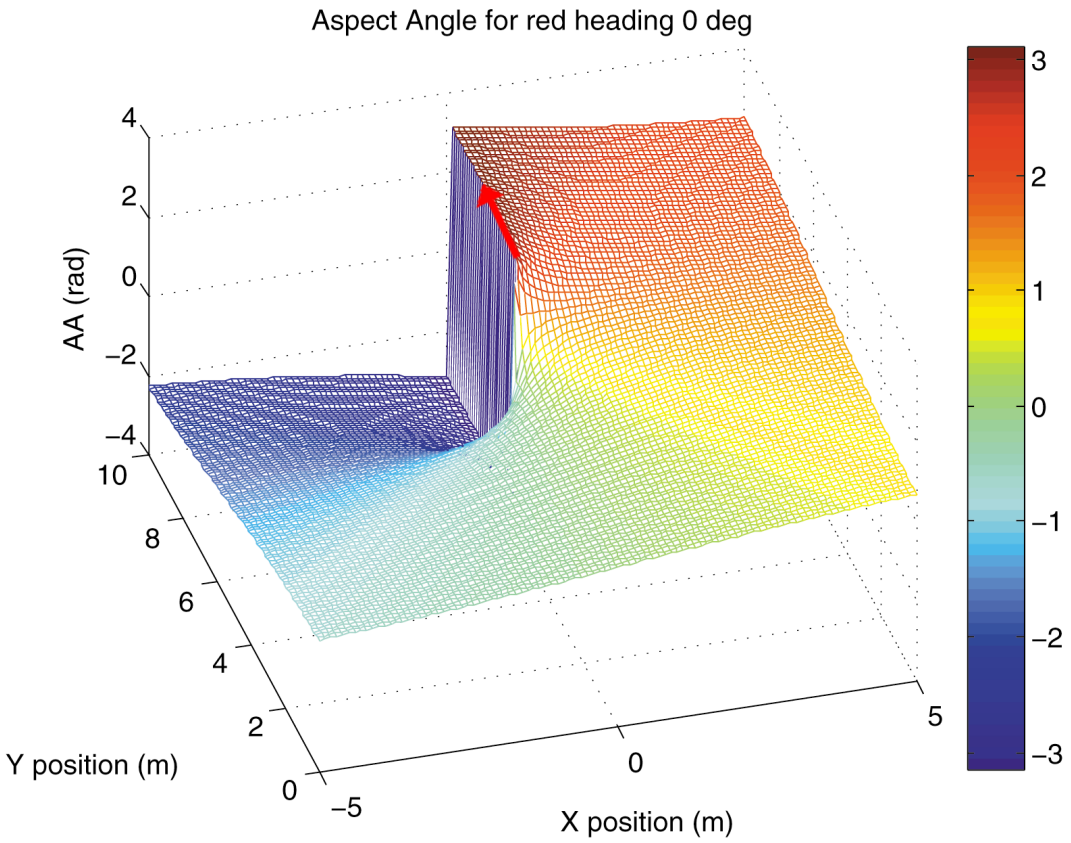


Figure 4-9: A graphical representation of the aspect angle (AA) feature used as part of the function approximation. AA represents part of the geometric relationship between the aircraft (see Figure 2-1). In this plot the red aircraft is in a fixed position heading 0 degrees. The plot represents the AA for various locations of the blue aircraft.

sight rate of the red aircraft. From the perspective of the blue aircraft $\dot{\text{AA}}$ is the rate in radians per second at which the opposing aircraft tracks across the windscreen. It incorporates own-ship bank angle and turn rate, range and adversaries velocity. $\dot{\text{AA}}$ is another piece of information which can be determined visually and is used to make critical decisions such as turn reversals during adversary turn circle overshoots. It is interesting that the approximation algorithm was able to achieve good function fits using the same features that a pilot uses for decision making during the course of BFM.

A critical and challenging component of successfully learning $\pi^*(x)$ is generating $J_{approx}(x)$ from the set of $\hat{J}(x)$ values derived from repeated Bellman backup

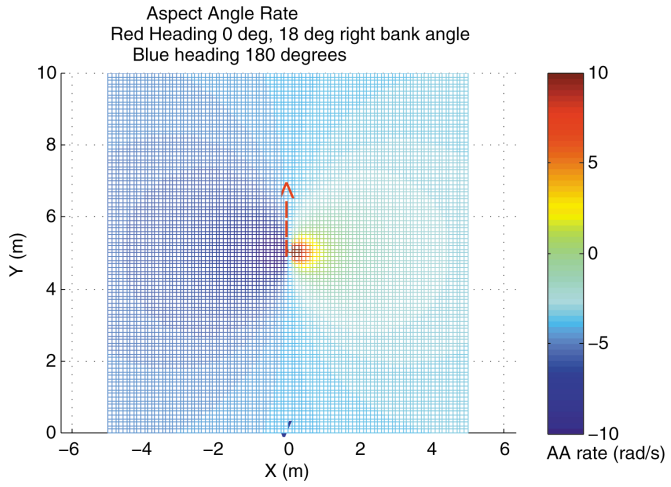


Figure 4-10: A graphical representation of the feature $\dot{AA}$ used in the function approximation. For this moment of time the red aircraft position is fixed and has a bank angle which relates to a specific turning rate. The scale to the right shows the $\dot{AA}$ perceived by the blue aircraft at various locations.

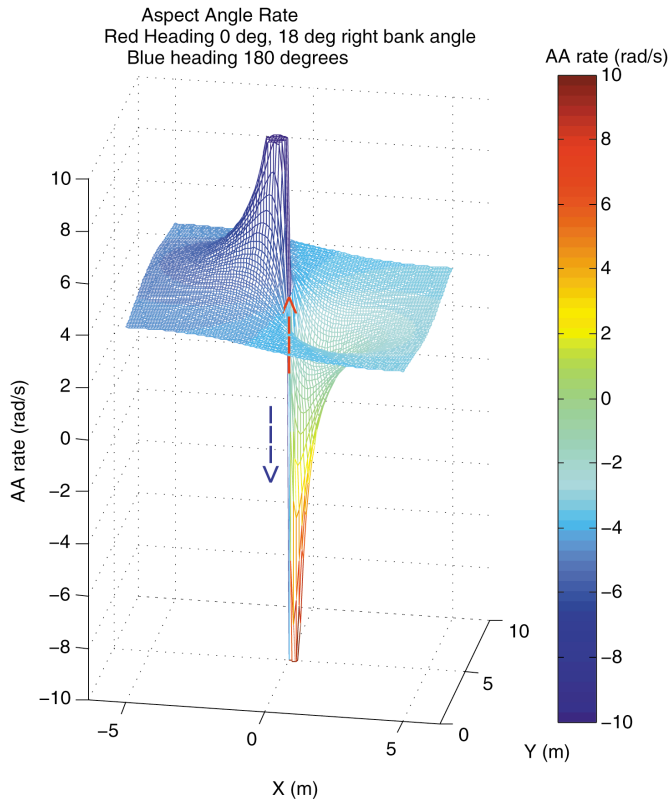


Figure 4-11: A rotated view of the $\dot{AA}$ Figure. The portion of the graph which passes through an $\dot{AA}$ rate of 0 rad/s also represents the red aircraft's current turn circle, as explained in the development of the heuristic scoring function. Continued flight on this circle allows the blue aircraft to stabilized the current aircraft relative geometry.

operations across the state space. Failure of the function approximation to fit and interpolate this data set will lead to errors. As it is an iterative process, those errors can grow to be quite large. The goal is to select a set of features that is capable of fitting the data. Likely features were selected based on the types of information used in actual air combat (see Table 4.2). The features selected are expanded via a 2^{nd} order polynomial expansion to build the feature vector, $\phi(x)$ that is used in the approximation process. This produces combinations of features for use by the function approximator. For example, if three features ($A(x)$, $B(x)$, and $C(x)$) were selected, the feature vector would consist of the following components:

$$\phi(x) = \left\{A(x), B(x), C(x), A^2(x), A(x)B(x), A(x)C(x), B^2(x), B(x)C(x), C^2(x)\right\} \tag{4.15}$$

The polynomial expansion successfully produces useful feature sets, however, using a large number of features in this manner proves to be computationally expensive, making manipulation of $J_{approx}(x)$ time consuming. This has the effect of slowing the learning process and, more critically, slowing the policy extraction process.

The forward–backward algorithm [19] was adapted to search the available features for a set that could accurately fit a $J_{approx}(x)$ function to a $\hat{J}(X)$ set. A single Bellman backup operation was performed on a set of states X to produce $\hat{J}(X)$. Six different selections of w_g and w_p were chosen to ensure the resulting features were capable of fitting the resulting $\hat{J}(x)$ sets. These were: $(w_g, w_p) \in [(0.0, 0.0)\ (0.0, 1.0)\ (1.0, 0.0)\ (0.0, 1.0)\ (0.3, 0.0)\ (0.5, 1.0)]$. The six different lines in Figure 4-12 each represent a different (w_g, w_p) setting.

The forward–backward search begins with an empty set of features. It searches each available feature for the one that minimizes the mean squared error (MSE) of $J_{approx}(x)$ as compared to $\hat{J}$. Cross validation [19] was used to determine the average MSE of each feature. The feature that minimizes MSE the most is added to the feature set. This process continues until each feature has been added to the set; this is the forward portion of the algorithm. The backward portion removes features one at a time, also selecting the feature that minimizes the MSE.

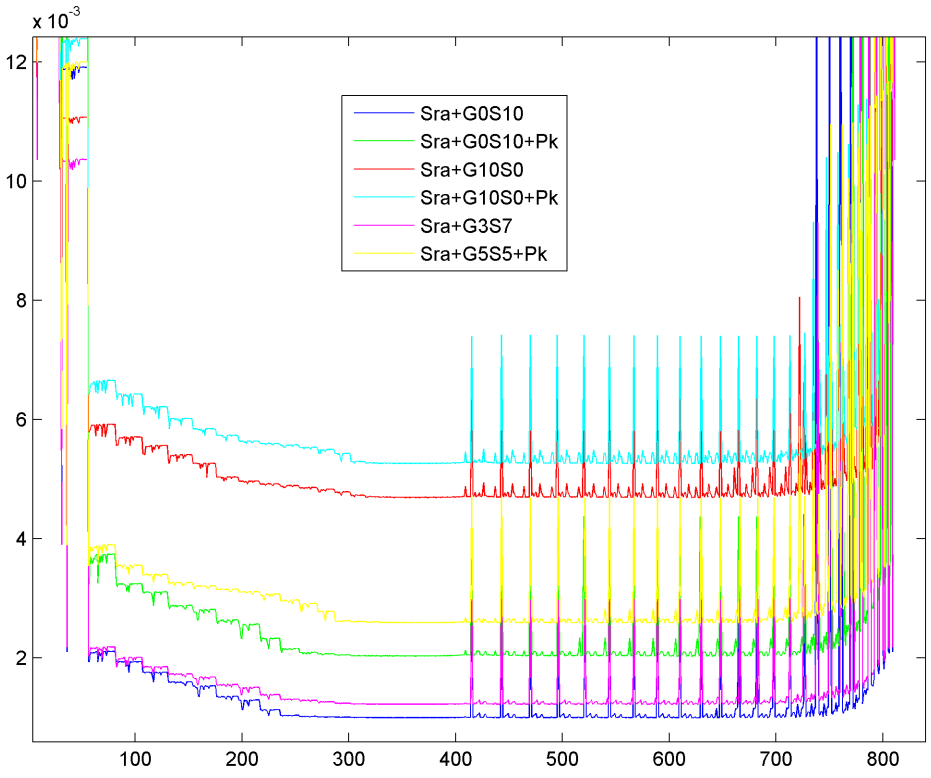


Figure 4-12: A forward-backward search was used to select the set of features used for the function approximation algorithm. The plot shows the mean squared error (MSE) of the function approximation produced from each set of features tested. Cross validation was used to determine the set of features that provided the lowest MSE.

The results of this process are shown in Figure 4-12. The stair-step appearance of the graphs on the left side represents the decreasing MSE as features are added on by one. The MSE reaches a minimum between 250 and 300 iterations when the feature sets have grown to a size of between 11 and 14 different features. Adding additional features has essentially no effect right up to iteration 407 when all 32 features were used in the set. As features are selected for removal, during the backward phase, the minimum MSE remains the same until iteration 700 (when there are 14 features remaining). At this point the MSE begins increasing as features are removed from the set.

The feature set that produced the absolute minimum MSE contained 22 different features. A subset of this feature set with 13 different features was selected for

use in the function approximation. The reduced number of features decreased the computation time significantly with only a 1.3% increase in MSE over the minimum found. The features selected were:

$$\{|\mathtt{AA}|,\ R,\ \mathtt{AA}^{+},\ \mathtt{ATA}^{-},\ S_A,\ S_R,\ |\mathtt{HCA}|,\ \dot{\mathtt{AA}}_{int},\ \dot{\mathtt{ATA}},\ \dot{\mathtt{ATA}}_{int},\ \theta_c,\ \phi_r,\ \phi_b\} \tag{4.16}$$

All of the features are derived from the eight components of the state, x, so there is a considerable amount of redundant information available in the features. As a pilot, I consider range, AA, ATA, $\dot{\mathtt{AA}}$ and $\dot{\mathtt{ATA}}$ to be the most critical pieces of information, so it seems logical that they would be chosen as features capable of approximating the future reward function $J(x)$. Furthermore, knowledge of the bank angle of each aircraft is important in understanding the current maneuvering situation, just as a pilot observes an opposing aircraft's orientation to make maneuvering decisions. Some of the bank angle information is built into $\dot{\mathtt{AA}}$ and $\dot{\mathtt{ATA}}$. To grasp the current turning performance of the two aircraft, the bank angles are the only possible source of that information.

4.7 Air Combat: Putting It All Together

In this section the ideas developed to this point are assembled to clarify the process used to apply approximate DP to the air combat game. The policy training process is discussed first, followed by an explanation of the policy extraction process.

4.7.1 Learning a Policy

The objective is to learn a maneuvering policy for a specific aircraft for use when engaged in combat against another specific aircraft. The flight dynamics of both aircraft are known and are defined by the state transition function $f(x, u_b, u_r)$ (Algorithm 7). The threat aircraft's weapons capability is known and is represented by the p_t function (Algorithm 6). Based on the maneuvering capabilities of both aircraft, a desired position of advantage can be defined. The $g(x)$ is defined to reward flight in this

Algorithm 8 Air Combat Policy Learning Algorithm

Initialize $J^1_{approx}(x) \equiv S_{bl}(x)$
Initialize N: the number of Bellman backup iterations desired
for $k = 1 : N$ **do**
$\overline{f} = f(X, u_b, \pi^{nom}_r(X))$
$\hat{J}^{k+1}(X) = \max_{u_b}[(1 - p_t(X))\gamma J^k_{approx}(\overline{f}) + g(\overline{f}) - p_t(X)g(x_{term})]$
$\Phi(X) = [\phi(x) \forall\ x \in \{X\}]$
$\beta^{k+1} = (\Phi^T\Phi)^{-1}\Phi^T\hat{J}^{k+1}(X)$
$J^k_{approx}(x) \equiv \phi(x)\beta^{k+1}$
end for
Output: $(J^N_{approx}(x))$

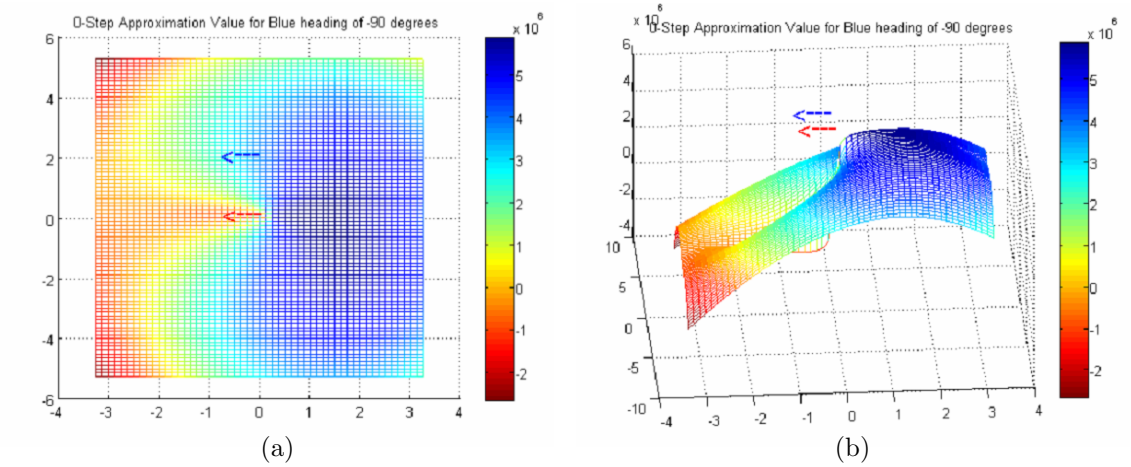
desired flight region behind the adversary aircraft. Finally, some expected maneuvering policy must be assumed for the adversary pilot. This policy can be developed based on knowledge of expected tactics and maneuvering techniques observed from the adversary pilots, or based on some form of optimization. The initial strategy is defined as $\pi^{nom}_r(x)$ and outputs the control actions u_r for the red aircraft given a state x. A neural-network classifier was found to be a good tool for representing $\pi(x)$. This policy can be updated in future iterations by the policies learned through approximate DP.

The next step is to sample the space to select a set of states for training (X). Selecting the appropriate number and the proper distribution of states is a challenging problem, see Section 4.2 for a discussion on the trajectory sampling technique. Selecting a feature vector $(\phi(x))$ is another difficult problem. Section 4.6 describes the types of features found useful for air combat. Once the mentioned functions are defined, Algorithm 8 can be used to produce a maneuvering policy.

The resulting J^N_{approx} defines the blue aircraft maneuvering strategy:

$$u_b = \pi^N_{approx}(x_i) \equiv= \arg\max_{u_b} \left[g(x_i) + \gamma J^N_{approx}(f(x, u_b, \pi^{nom}_r(x_i)))\right] \tag{4.17}$$

This will select the best blue control action given any state of the game. However, this assumes that $J^N_{approx}(x)$ is a perfect representation of the true $J^*(x)$, which it is not. To minimize the effect this difference has on the resulting policy, a policy



(a) (b)

Figure 4-13: Function approximation from dynamic program. Function is used at each time-step with a one-step look-ahead maximization to determine best control action. In this graph the red and blue heading and bank angle are fixed. The color represents the relative value (blue=good, red=bad) given to blue aircraft positions surrounding the red aircraft.

extraction method using rollout can be used.

4.7.2 On-line Policy Extraction

Rollout is a technique used to extract a policy from $J^N_{approx}(x)$ that more closely approximates the optimal policy $\pi^*(x)$ than $\pi^N_{approx}(x_i)$ manages to do. This is done by selecting each possible u_b as the first action in a sequence, then simulating subsequent actions using $\pi^N_{approx}(x_i)$ for a selected number of rollout *stages* [6]. The policy resulting from rollout is referred to as $\bar{\pi}^N_{approx}(x_i)$. Algorithm 9 shows the procedure used to determine $\bar{\pi}^N_{approx}(x_i)$ on-line in both simulation and flight tests.

sectionChapter Summary

Section 4.7 outlines how the approximate DP method is utilized in this research to first learn and then execute air combat maneuvering. I would like to comment on how this method can be applied in a slightly broader sense. This sequence of processes can be considered the first of a continuing cycle of learning and adapting. A human pilot's maneuvering decisions will vary based on the type of aircraft they are engaged with and what the opponent's maneuvering policy is. Similarly, an operational UAS would need to have access to a variety of maneuvering policies for each opponent

Algorithm 9 Air Combat Policy Rollout Extraction Algorithm, $\bar{\pi}^N_{approx}(x_i)$

Input: x_i
$J_{Best} = -\infty$
for $u_b = \{left, straight, right\}$ **do**
  $x_{temp} = f(x_i, u_b, \pi^{nom}_r(x_i))$ (where x_{temp} is temporary future state variable)
  for $j = \{1 : N_{rolls}\}$ (where N_{rolls} is desired number of rollout stages) **do**
    $x_{temp} = f(x_{temp}, \pi^N_{approx}(x_{temp}), \pi^{nom}_r(x_{temp})$
  end for
  $f_{temp} = f(x_{temp}, u_b, \pi^{nom}_r(x_{temp}))$
  $J_{Current} = [(1 - p_t(x_{temp}))\gamma J^N_{approx}(f_{temp}) + g(f_{temp}) - p_t(X)g(x_{term})]$
  if $J_{Current} > J_{Best}$ **then**
    $u_{best} = u_b$
    $J_{Best} = J_{Current}$
  end if
end for
$u_b = u_{best}$
Output: u_b

airframe and for different types of red policies. During a continued conflict, the tactics demonstrated from both sides have the tendency to evolve. Human pilots are capable of learning and adapting to changing tactics. A successful algorithm for flying air combat would need to take the data learned one day and apply it the next. By using data collected by a number of engagements, the models used for red policies can be continually updated. A neural net classifier proves to be an effective method of generating a maneuvering policy from a set of data. The updated red policy can then be used to learn a new, improved blue maneuvering policy. This new policy can easily be distributed and utilized by any number of UAS aircraft. This process probably requires human interaction at some level, but it does not require intensive human interaction to update or teach. The algorithm is capable of learning the required maneuvering on it's own.

Chapter 5

Combat Implementation Using Function Approximation

The process outlined in Chapter 4 was successfully in generating maneuvering policies capable of making decisions for air combat. Numerous aspects of the learning and policy extraction process allow for adjustments that have an impact on the resulting maneuvering decisions. This chapter describes a combat simulation and how it was utilized as a metric for policy comparison. The results from large numbers of simulations were used to select algorithm settings. The settings that were adjusted based on simulation results are as follows:

- The number of iterations to train $J_{approx}(x)$,
- The weight placed on the goal functions, w_g,
- The weight placed the risk function, w_{pt}
- The number of rollout stages to use during policy extraction.

Additionally a neural-network classifier was used to approximate the red maneuvering policy, thus decreasing the time required for policy extraction. The calibration and performance of the neural-network is discussed, as are the resulting simulated combat performance data. Finally, the ability of the calibrated maneuvering policy to perform

Table 5.1: Six initial states (called setups) used for simulation testing.

x_{init}	Desc.	x_b^{pos}	y_b^{pos}	ψ_b	ϕ_b	x_r^{pos}	y_r^{pos}	ψ_r	ϕ_r
1	offensive	0 m	-2.5 m	0°	0°	0 m	0 m	0°	0°
2	1–circle	2.75 m	0 m	0°	-25°	0 m	0 m	0°	20°
3	defensive	0 m	0 m	0°	0°	0 m	-2.5 m	0°	0°
4	high aspect	0 m	-4.0 m	0°	0°	0 m	0 m	180°	0°
5	reversal	0 m	0 m	40°	25°	0.25 m	-0.25 m	-45°	0°
6	2–circle	0 m	0.1 m	270°	-25°	0 m	-0.1 m	90°	-20°

real-time air combat using micro-UAS aircraft was tested. The results from these flight tests are included in Section 5.4.

5.1 Combat Simulation

The policy, π learned by the dynamic programming method, was tested in air combat using a simulation program. The simulation is based on the state transition function described in Algorithm 7. The aircraft were initialized at the specific starting points defined in Table 5.1. Complete plots of the simulations from each setup can be seen in Appendix A. The simulation accepted a control action, u, from both aircraft, then progressed the state forward $\Delta t = 0.25$ s using $x_{t+1} = f(x_k, u_b, u_r)$. The simulation terminates when one aircraft manages to receive the reward $g_{pa} = 1.0$ for 10 consecutive steps (2.5 s), thus demonstrating the ability to achieve and maintain flight in the defined position of advantage.

The six initial states in Table 5.1 were chosen to evaluate a range of specific maneuvering tasks. The specific setups were designed to assist in easy evaluation of maneuvering performance. For example Setup #1, is an offensive setup for the blue aircraft. The blue aircraft is initialized inside the goal zone behind the red aircraft. With the appropriate maneuvering, the blue aircraft can claim victory in 2.5 s, simply by maintaining the position of advantage for 10 time-steps. If a policy were to fail to accomplish this basic task, it would be obvious that it is failing to produce reasonable decisions.

Of course, evaluating air combat performance is not simply a matter of either good

or bad performance. To compare the algorithms in a more continuous manner, two metrics were chosen to represent success level: time to intercept (TTI) and probability of termination (p_t). TTI was measured as the elapsed time required to maneuver to and maintain flight within the goal zone for 2.5 s. A smaller TTI is better than a larger value. Either aircraft has the possibility of winning each of the setups, however, it is expected that blue should win due to the performance advantage enjoyed by the blue aircraft ($\phi_{b\_\max} > \phi_{r\_\max}$). The p_t was also accumulated over the course of each simulation to produce a total probability of termination for the entire engagement. To compute this value we must define a new term representing the probability of continuing, p_c:

$$p_c = 1 - p_t \tag{5.1}$$

$$p_{c\_total} = p_{c\_total}(1 - \Delta t \; p_t(x_t)) \tag{5.2}$$

Equation 5.2 must be evaluated at each time-step. At the end of the simulated combat, p_{c_total} can be used to determine the p_{t_total} for the engagement.

$$p_{t\_total} = 1 - p_{c\_total} \tag{5.3}$$

A minimum amount of risk is desirable. While, the primary goal is to minimize TTI, a secondary goal is that of minimizing p_{t_total}.

In order to have some basis for comparison, a nominal maneuvering strategy was used for the blue aircraft. As explained in Section 4.2, the red aircraft uses a minimax search with the baseline scoring function to produce u_r. The nominal maneuvering strategy, π_b^{nom}, used for the blue aircraft is generated using the same technique. While both aircraft have equivalent strategies, the blue aircraft consistently wins the engagements due to the available performance advantage. An additional comparison strategy was produced by the author using manual human control of the blue aircraft in simulation. Utilizing approximately 10 years and 1000 flight hours of U.S. Air Force combat training, the author manually input u_b at each time-step of the simulation.

The best results obtained are similarly used for comparison with the DP generated maneuvering policies. This maneuvering policy will be labeled π_b^{pilot}.

5.2 Parameter Calibration

To calibrate the learning algorithm, a series of simulation tests were done on various policies. The naming convention for policies will be: $\pi^k_{wg\_wp}$. This policy was produced after k iterations, using a goal weight value of w_g and a penalty weight of w_p.

In the first phase of testing the number of iterations was varied to determine the effect of the resulting policies on combat performance. The policy used was $\pi^k_{0\_0}$, where w_g=w_p=0, and k was varied from 0 to 80. Because each Bellman backup iteration can also be considered equivalent to a 0.25 s increase in planning horizon, this could be considered as varying the planning horizon between 0 s and 20 s. The results from this series of simulations is plotted in Figure 5-1. The policies $\pi^k_{0\_0}$ were determined using a rollout of 3, 4, and 5 stages. The resulting TTI is plotted and compared to the average TTI resulting from the π_b^{nom} and π_b^{pilot} policies. The value of 40 iterations, or 10 s look-ahead was chosen for use in continued testing as it provided the best TTI and p_t performance.

The next step in calibration was to select the weighting value w_g that produced the most effective policy. The number of iterations was fixed at 40, and w_p was fixed at 0. The policies, $\pi^{40}_{wg\_0}$, were evaluated in the simulator with rollout of 1, 2, 3, 4 and 5 stages. The results are plotted in Figure 5-2. Values of $w_g > 0.8$ caused inconsistencies and poor behavior in both TTI and p_t. Based on this testing the goal weighting was chosen to be $w_g = 0.7$ for future tests.

With the number of iterations and w_g values determined, the best value of w_p was computed using a similar series of policies, $\pi^{40}_{0\_wp}$ where w_p was varied from 0 to 1.0 in 0.1 increments. The value for w_g was optimized first because w_g has a greater effect on overall performance than p_t. As seen in Figure 5-3 the results of both TTI and p_t are erratic. The simulation performance seems somewhat indifferent to w_p except when $w_p > 0.8$, then the policy performance degrades drastically. This particular

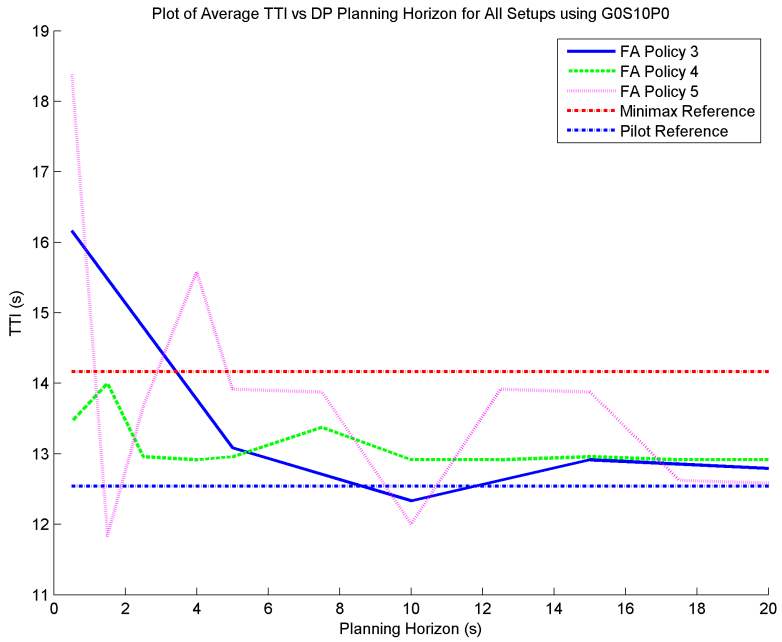
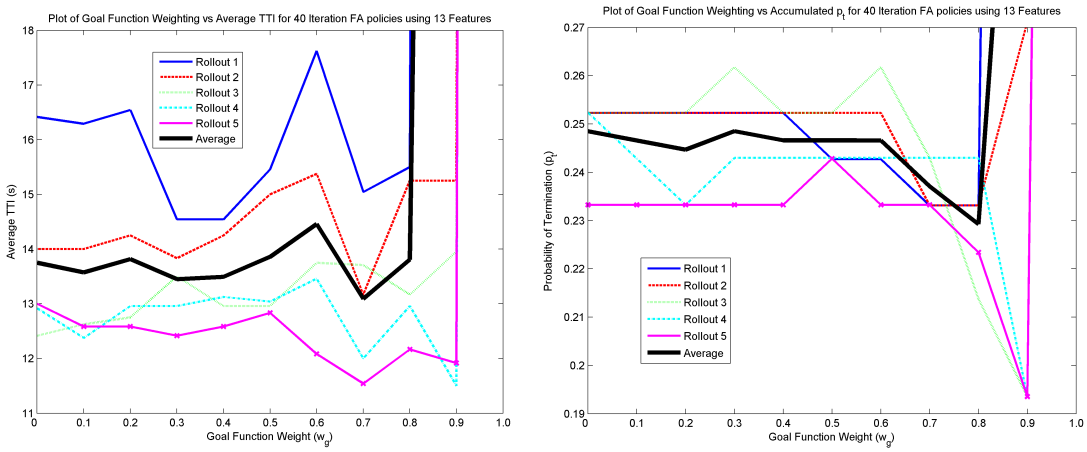


Figure 5-1: The approximate dynamic program performs Bellman backups over 0.25 s discrete time interval. The policy resulting from a variety of algorithm iterations were tested in the simulator for performance. 40 iterations, representing a 10 s look ahead, proved to provide the best results.

method of reinforcing aversion to risk appears to be insufficient, or simply incorrect. A value of 0 was used for w_p for the remainder of the testing to eliminate the effect of p_t on the learning and policy extraction process.

As explained in Section 4.7.2, the red maneuvering policy must be evaluated multiple times during the policy extraction process. The red policy uses a minimax search, which is relatively time consuming to compute. When executing a 3-step rollout, the red policy must be evaluated 30 times. In order to accomplish the policy extraction process in real-time, a fast method must be used to determine the assumed red control action. A probabilistic neural-network classifier is available in the Matlab® `Neural Net Toolbox` [23]. This function called, `newpnn`, accepts a set of feature vectors, $\phi(X)$ and a target vector, which in this case is the corresponding set of red control actions $U_r = \pi_r^{nom}(X)$ (computed using the minimax algorithm). Using the same architecture described in Section 4.6, a forward–backward algorithm was used to search for a feature set that produced the highest correct percentage of red policy classification.



(a) Time to intercept.

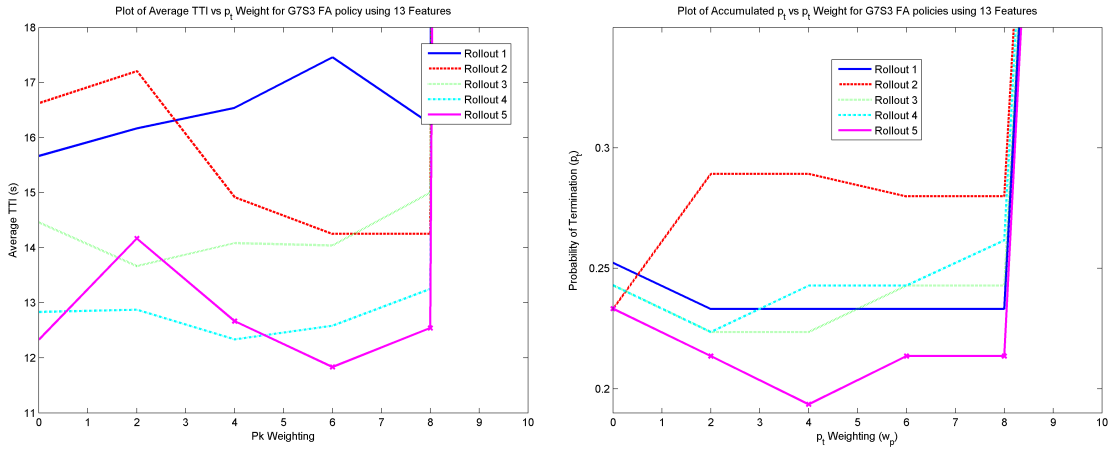
(b) Probability of termination.

Figure 5-2: Multiple maneuvering polices were generated as the weight placed on the two goal reward functions were varied. The value of w_g was varied between 0 and 1.0. The resulting average TTI derived from running the simulator with the respective policy is plotted versus the various weight combinations. The lowest TTI seem to be obtained using approximately $w_g = 0.7$.

A plot of the classifier performance during the search process is shown in Figure 5-4. A set of 5000 states was used to generate the features and associated u_r used to train the neural net. Larger data sets created networks that were slower to evaluate. Likewise, the larger the number of features selected, the slower the neural net operated. Fortunately, the highest classification percentage for the neural net was obtained with only five features. Figure 5-4 shows this point occurred during the forward portion of the search and produced the correct value for u_r 95.2% of the time. The features selected were $\{\texttt{AA}, R, S_{bl}, x_{rel}^{pos}\ v_{rel}\}$.

This neural-net helped to increase the operating speed of the policy extraction algorithm by an order of magnitude. Figure 5-5 shows the improvement of computation time over the use of the minimax function. The neural net allows for a 4-step rollout to be accomplished in real-time (represented by the horizontal line at 10^0). The neural net is a useful tool that could most likely be adapted to classify most π_r regardless of the specific method used to produce the policy (i.e. hard coded maneuvers, game theory, model predictive control).

With the neural-net classifier calibrated, a second series of simulations was done to verify the goal weight (w_g) used. The policy $\pi_{wg\_0}^{40}$ was used as w_g was varied from 0 to



(a) Time to intercept. (b) Probability of termination.

Figure 5-3: Multiple maneuvering polices were generated as the penalty weighting factor w_p was varied. w_g was varied from 0 to 1 in 0.1 increments. Simulation performance appears to be relatively indifferent to this method of penalty reinforcement.

1.0. The results from a simulation using a 3-step rollout is shown in Figure 5-6. The results show a marked improvement over the pilot reference policy and a minimum TTI and p_t at $w_g = 0.8$. Thus the policy $\pi^{40}_{8\_0}$ was selected as the best obtainable with the current arrangement. A rollout of 3-steps is used for future testing.

The performance of the $\pi^{40}_{8\_0}$ policy as compared to the baseline blue policy, π^{nom}_b, is shown in Figure 5-7. In Figure 5-7(a) the average TTI per engagement and accumulated p_t is shown for both the $\pi^{40}_{8\_0}$ (left column in each figure) and π^{nom}_b. The $\pi^{40}_{8\_0}$ policy is approximately 23.6% faster in achieving the position of advantage and does so with a 13.0% decrease in p_t. This performance is also 7.2% better than the pilot reference results in TTI and 14.6% in p_t. Figure 5-7(b) and 5-7(c) show the results of the individual setups. Setup #5 (reversal) is the one engagement where the π^{nom}_b policy managed a shorter TTI. The difference is small, approximately 1 s, and the improvements in the other setups are comparatively large. $\pi^{40}_{8\_0}$ accumulated an equal or lower p_t than π^{nom}_b for all setups.

The $\pi^{40}_{8\_0}$ policy was tested against policies other than the π^{nom}_r policy that it was trained against. This demonstrates the ability to maneuver successfully against an adversary that doesn't quite do what is expected, which is an important attribute of any combat system. The results appear promising. Figure 5-8 shows four plots similar

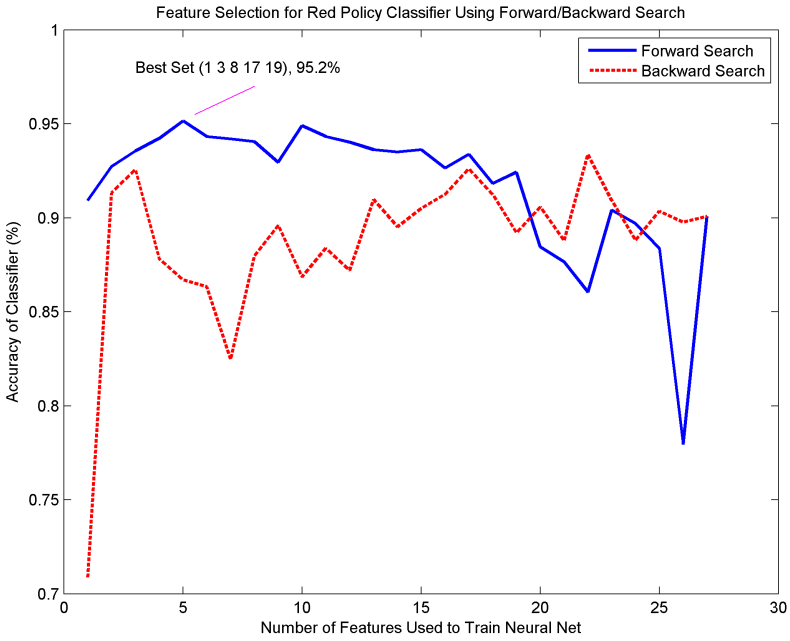


Figure 5-4: Calibration process for a neural-net used to classify the 6-step minimax red maneuvering policy. A forward-backward search was used to find a set of features that would produce minimal classifier error. The classifier was used to reduce the on-line time required to extract a decision from the maneuvering policy.

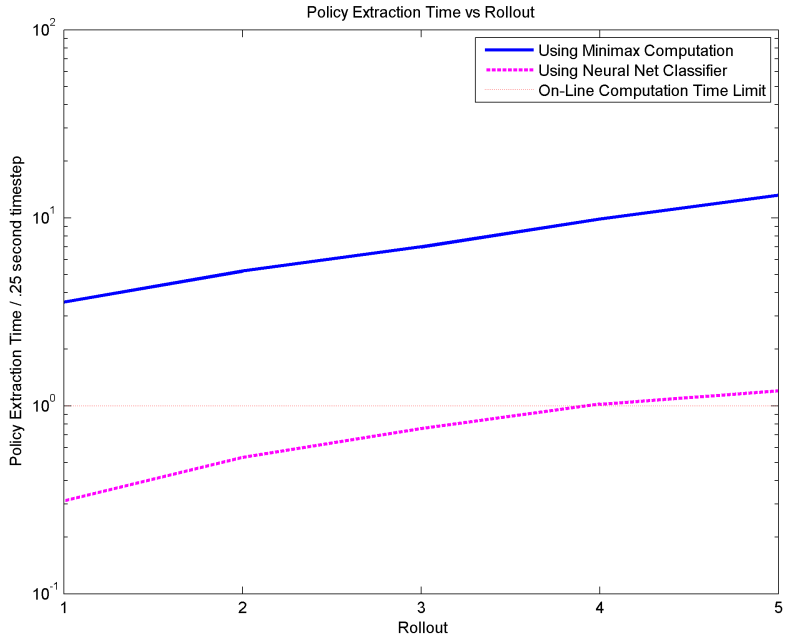
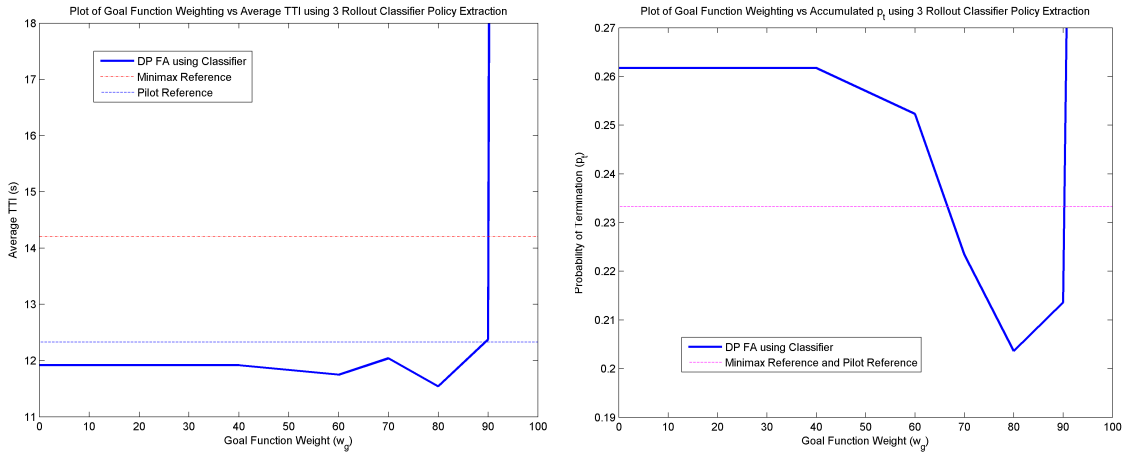


Figure 5-5: This plot demonstrates the decrease in policy extraction time enjoyed through the use of a classifier to represent the red maneuvering policy, eliminating the need to perform a mini-maximization during the rollout process.



(a) Time to intercept

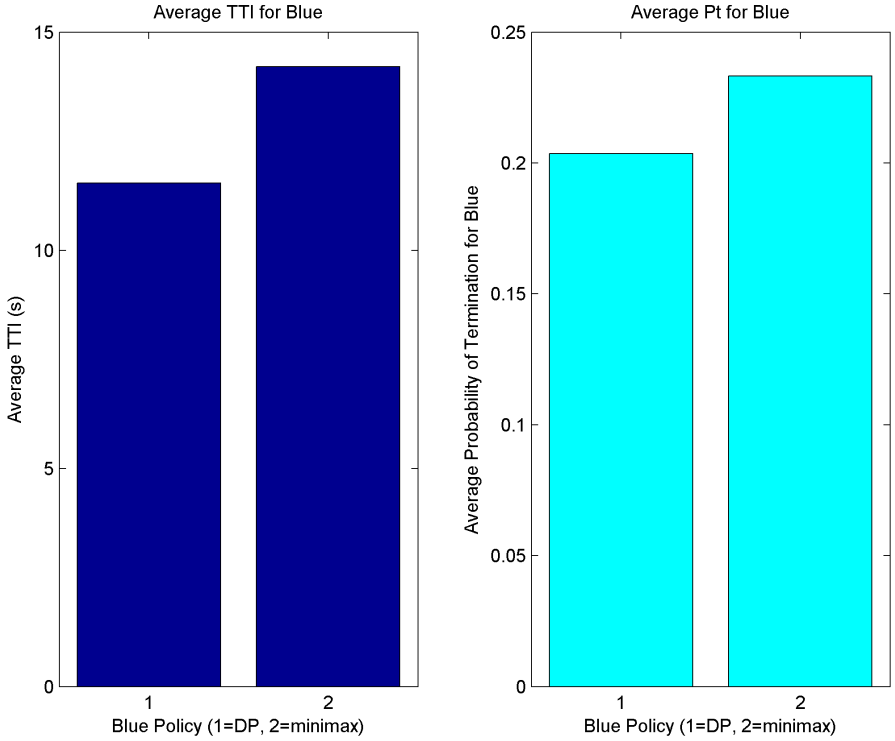
(b) Probability of termination.

Figure 5-6: Simulation performance using the neural net classifier for the red policy in the rollout algorithm.

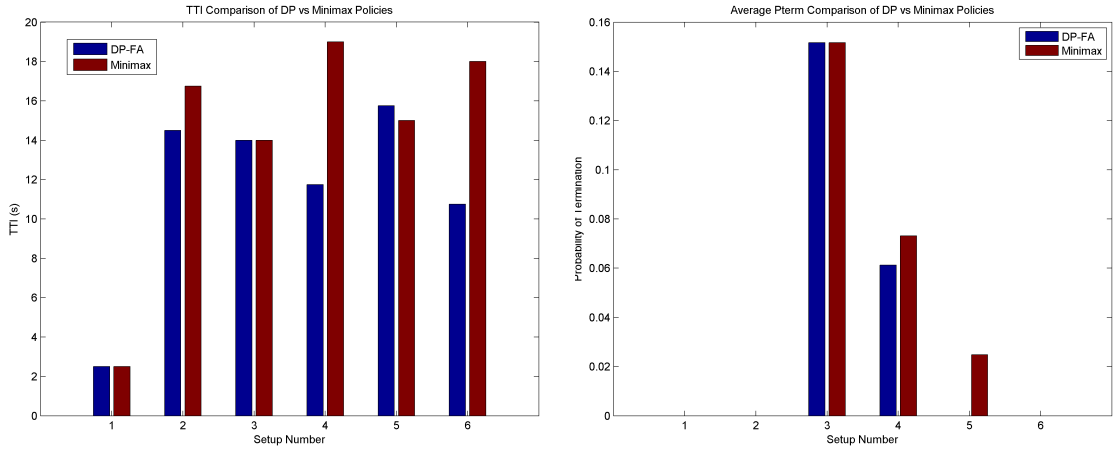
to Figure 5-7(a); the difference here being that each chart represents the performance of $\pi_{8\_0}^{40}$ and π_b^{nom} policies in combat versus a different red policy. The policies were a 10-step minimax search, a pure-pursuit policy, a left turning policy and a right turning policy. In each case $\pi_{8\_0}^{40}$ did better than π_b^{nom} in TTI. Note the considerable additional average time required against the 10-step minimax search, as compared to Figure 5-7(a). The additional look ahead of the 10-step minimax policy creates u_r maneuvering decisions that are much more difficult to counter than the policy used to train $\pi_{8\_0}^{40}$. However, $\pi_{8\_0}^{40}$ still shows improvement over the nominal blue policy.

5.3 Simulation Results

Figure 5-9 shows a typical perch setup simulation flown by a function approximation policy. Upon initial setup, the blue aircraft is positioned behind the red aircraft, who is showing a +40 degree `AA`. At the initiation of the simulation, the red aircraft begins a maximum performance right turn. The blue aircraft drives ahead then initiates a break turn which concludes with flight in the goal zone behind the red aircraft. At the termination of the break turn, the blue aircraft's flight path is aligned with the red aircraft's flight path; this allows continued flight in the goal zone, without a flight path overshoot. This is excellent behavior with respect to traditional BFM techniques.



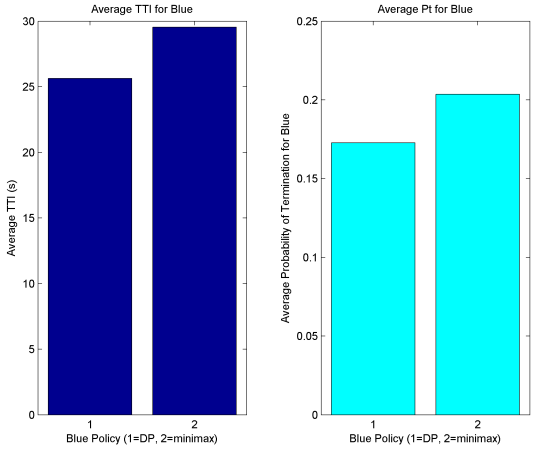
(a) Overall Performance comparison.



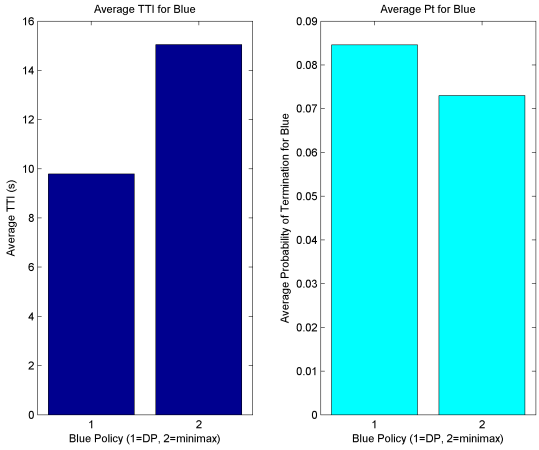
(b) TTI of each setup.

(c) P_t of each setup.

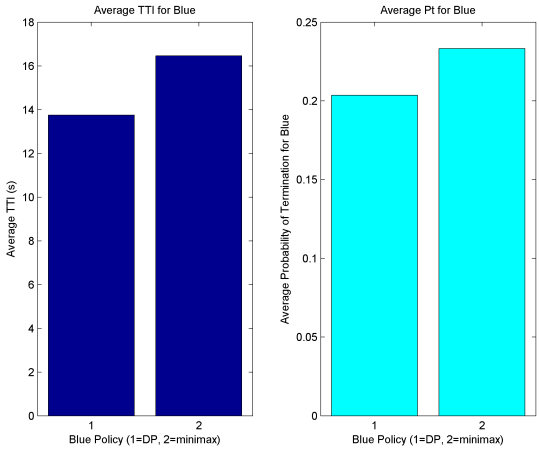
Figure 5-7: Simulation performance of best maneuvering policy evaluated with a 3-step rollout using the neural net classifier for red maneuvering policy evaluation. This represents a large improvement of performance over the baseline π_b^{nom} policy, and the pilot reference performance.



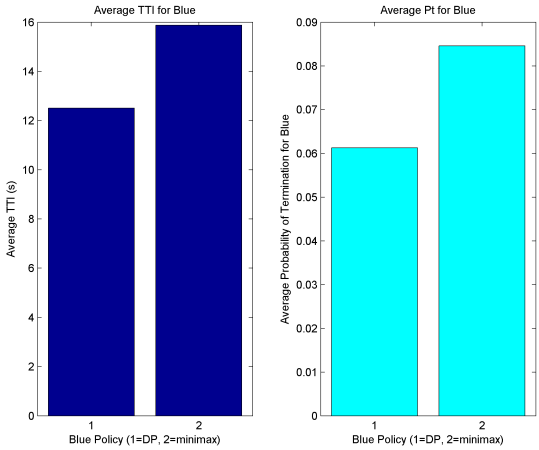
(a) 10 step minimax red policy.



(b) Left turn only red policy.



(c) Pure pursuit red policy



(d) Right turn only red policy

Figure 5-8: The calibrated function approximation blue maneuvering policy was tested against different red policies. The blue maneuvering policy was trained against a 6-step minimax red maneuvering policy. Here the blue policy shows it is still effective in combat against policies other than the one it was trained on.

In Figure 5-10 a high aspect setup is shown. Both aircraft initially fly toward each other. The red aircraft initiates a left turn forcing a 1-circle fight[1]. The blue aircraft correctly chooses a maximum performance right turn. At approximately 5 s into the engagement, after one leaf of the 1-circle fight, the blue aircraft has established flight in the goal zone.

Appendix A contains complete drawings from each setup during simulation testing. The plots were drawn every 3 s during combat simulation and show 4 s history trails of both the red and blue aircraft. Side by side comparison of the simulations enables the reader to see some of the subtle differences in maneuvering from the $\pi^{40}_{8\_0}$ policy that result in considerable improvements. The ability of the function approximation to learn small nuances that lead to better performance would appear to validate its usefulness in solving complicated problems of this nature.

Setup #1 in Figure A-1 is not particularly interesting; both policies found the best solution and the lines look equivalent. During Setup #2 (Figure A-2 and A-3) the $\pi^{40}_{8\_0}$ policy does better than the π^{nom}_{b} policy. In Figure A-2(e) one can see that the red aircraft chooses to reverse the turn to the left, while in Figure A-2(f) the red aircraft continues to the right. There is no noticeable difference in the previous frame, however, close inspection of the lines at 5 s shows a small difference. In the following Figure, A-2(e), $\pi^{40}_{8\_0}$ quickly takes advantage of the red aircraft's decision and wins. Note that these simulations are completely deterministic, so any deviation on the part of red is due to some difference in the blue maneuvering. Red is reacting to something different that blue did. In essence $\pi^{40}_{8\_0}$ is capable of "faking-out" red by presenting a maneuver that appears attractive to red, but blue is capable of exploiting in the long term. The $\pi^{40}_{8\_0}$ policy was trained against the red policy and learned based on the decisions observed. The ability to learn how to elicit a response from the adversary that is advantageous to yourself is a very powerful tool.

Setup #4 in Figures A-6, A-7, and A-8 shows a very similar behavior. In the very first frame (Figure A-6(b)) the $\pi^{40}_{8\_0}$ policy makes a small check turn to the left, then

[1] A 1-circle fight is the term given to a portion of a BFM engagement in which both aircraft are established on the same turn circle, but are traveling in opposite directions around the circle.

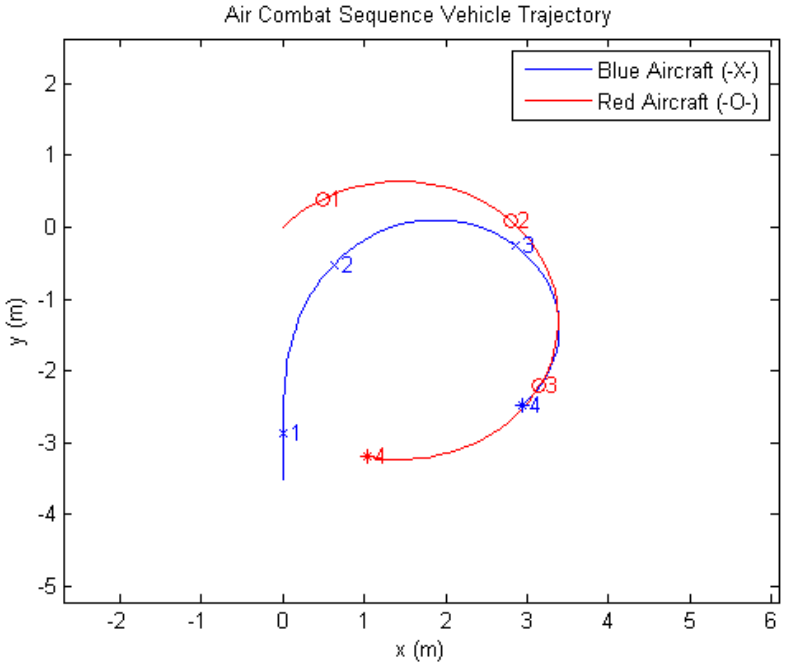


Figure 5-9: Simulation result from dynamic program function approximation demonstrating effective performance in a perch BFM setup. The numbers along each trajectory represent time in seconds.

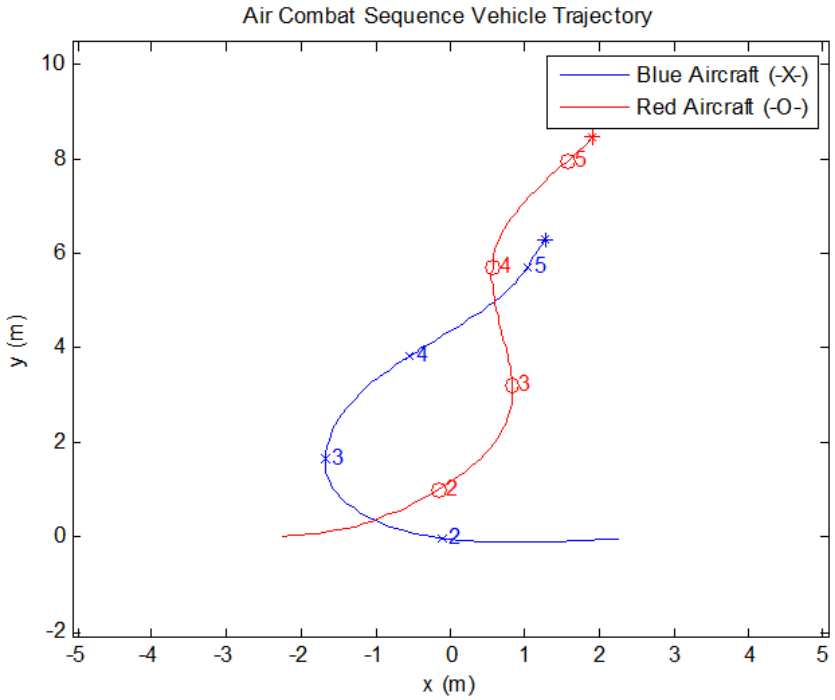


Figure 5-10: Simulation result from dynamic program function approximation demonstrating effective performance in a high aspect BFM setup. The numbers along each trajectory represent time in seconds.

immediately initiates a right hand lead turn. This forces a 2-circle fight and allows the blue aircraft to make a small gain prior to the initial merge. In the following frame, at 4 s, $\pi_{8\_0}^{40}$ is extremely offensive, while the π_b^{nom} is practically neutral. This small improvement in maneuvering pays big benefits as the fight unfolds causing $\pi_{8\_0}^{40}$ to complete the engagement much quicker.

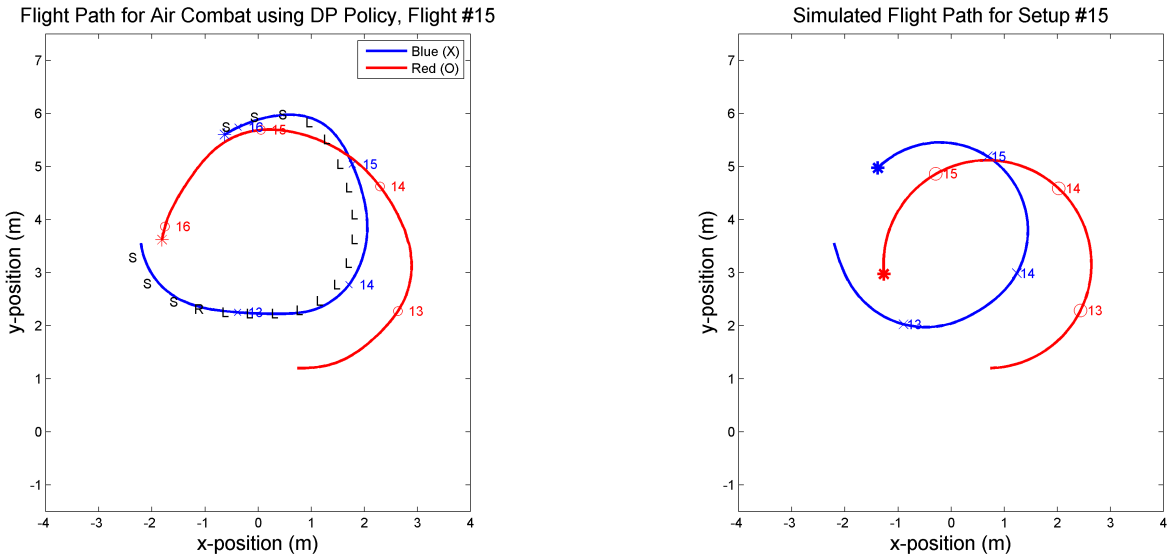
A final example of this behavior is in Setup # 6 in Figures A-11(e) and A-11(f). The $\pi_{8\_0}^{40}$ manages to force the red aircraft to reverse back to the right, culminating in a faster victory. The red aircraft will only choose to reverse when it appears to be advantageous based on the minimax search algorithm applied. The $\pi_{8\_0}^{40}$ appears to have done a good job of exploiting its knowledge of the reds minimax maneuvering strategy.

Setup #5 in Figure A-9 was designed to be right on the edge of a reversal situation. A reverse is a maneuver the blue aircraft would make in this situation when the red aircraft overshoots the blue aircraft's flight path. A severe overshoot countered by an appropriately timed reversal can result in a swap of the offensive/defensive role. This situation is not as severe, and therefore the decision to reverse or not is more difficult make. The algorithms selected opposite choices in this case. As it turns out the decision made by $\pi_{8\_0}^{40}$ resulted in a quicker engagement.

Apart from the situations mentioned, it appears that based on accepted methods of basic fighter maneuvering, $\pi_{8\_0}^{40}$ continues to make good maneuver selection. Once the two different maneuvering policies deviate, is difficult to make direct comparisons, but $\pi_{8\_0}^{40}$ appears to be thinking further ahead and, therefore, completes the intercepts in less time and with less accumulated risk.

5.4 Flight Testing

Section 5.3 demonstrated the efficiency of the DP method in a simulated environment, and the results showed that the DP method was able to learn an improved blue policy. Furthermore, using the red policy classifier we are able to execute that policy in real-time. This section completes the results by demonstrating the policy using flight tests



(a) Flight Trajectory.

(b) Simulated Trajectory.

Figure 5-11: Flight and simulation results comparison. The simulation was started at the same initial state as this particular flight sample to compare actual flight with the simulation used to train the blue policy.

on a real micro-UA in RAVEN.

The $\pi_{8\_0}^{40}$ policy was tested using the micro-UA aircraft and system architecture described in Chapter 3. The policy extraction algorithm (Algorithm 9) was run on a desktop computer linked with the RAVEN vehicle controllers. State data was received from RAVEN, processed using the Matlab® code used for simulation testing. The blue control action (u_b) was then sent directly to the vehicle controllers, where the PID controllers generated the vehicle commands.

In order to generate technically interesting results in RAVEN, flight tests using an extended perch setup (similar to Setup #1 in Table 5.1). In the perch setup, blue is positioned behind red where red has already entered a banked turn. To keep the fight within the restricted flight environment, the red aircraft followed a (left-hand) circular trajectory with no additional evasive maneuvers. The circle represented the maximum performance turn allowed in the simulation. This procedure was necessary to avoid the walls and other obstacles in RAVEN. However, a hard left turn is exactly the evasive maneuver performed by red in simulation starting from Setup #1. Thus, the flight tests demonstrate realistic behavior.

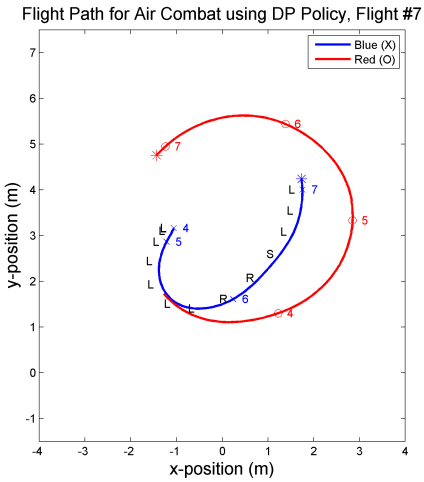
Effective maneuvering from the perch setup requires lead pursuit to decrease range. In the extended perch, blue is positioned further behind red than Setup #1, thus, requiring additional lead pursuit maneuvers as well as real-world corrections.

Figure 5-11 demonstrates these deviations and the associated corrections. For example, in the simulated trajectory (Figure 5-11(b)), red makes a perfect left hand turn. However, in the actual flight test (Figure 5-11(a)) red experiences turbulence caused by blue's presence resulting in an imperfect circle. This is also different than the undisturbed flight in Figure 3-5(a). After the disturbance, red corrects in order to track the prescribed circle, and thus sometimes exceeds the bank limit imposed in the simulation.

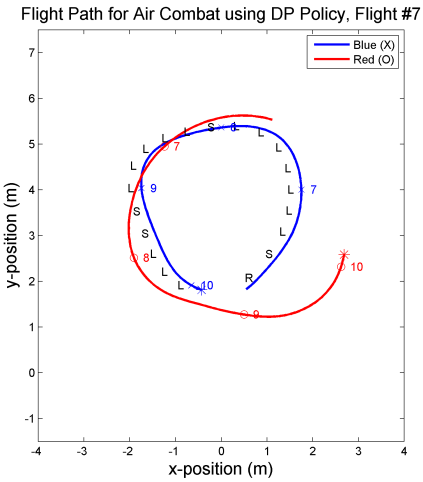
Figures 5-12 and 5-13 demonstrate two different fights started from the extended perch setup. We can track blue's actions by the $\{L, S, R\}$ labels plotted at 0.2 s intervals along the blue flight path. In the first flight, blue aggressively pulls lead pursuit (Figure 5-12(a)). Blue then eases to accommodate red's elongated turbulence induced turn (Figure 5-12(a)), then continues lead pursuit in Figure 5-12(b). By Figure 5-12(d) blue has attained the goal zone position and maintains it until a disturbance sets the aircraft off course. Blue quickly recovers and reattains the goal zone positions.

In the second fight blue again aggressively pulls lead pursuit (Figure 5-13(a)). Blue continues lead pursuit through Figure 5-13(b) in order to close the range with red. Figures 5-13(c), 5-13(d), and 5-13(e) reflect significant real world disturbances and corrections involving both the blue and red aircraft. Toward the end of Figure 5-13(e) blue is in an aggressive pursuit and begins to ease (indicated by the right (R) control inputs) in order to converge onto red's path.

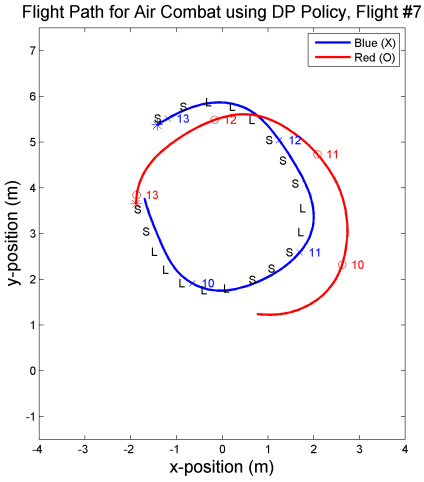
The flight results clearly validate the efficacy of the air combat strategy as well as the flight controller. Blue demonstrated correct strategy and red's flight controller demonstrated correct flight path corrections. Overall the flight tests were a success.



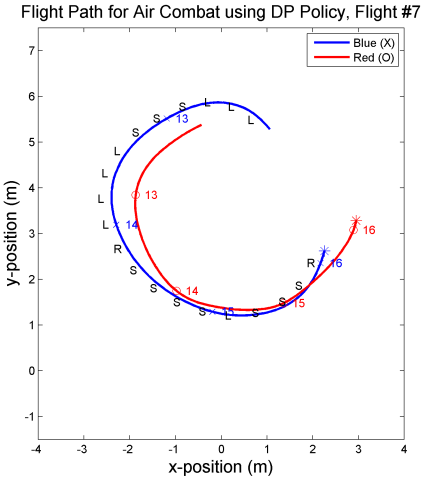
(a) 7.1 s.



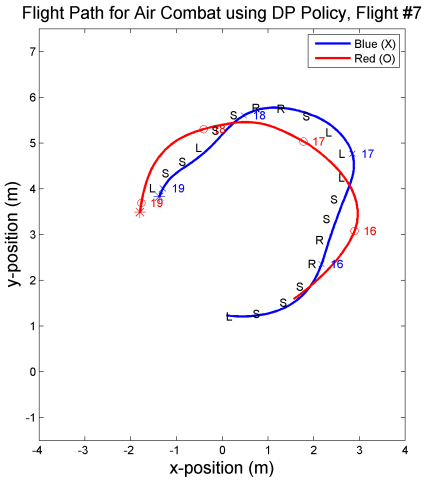
(b) 10.1 s.



(c) 13.1 s.

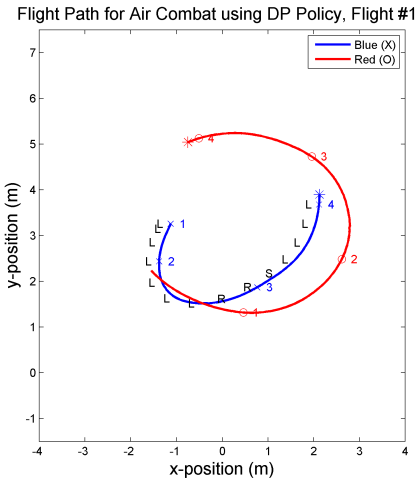


(d) 16.1 s.

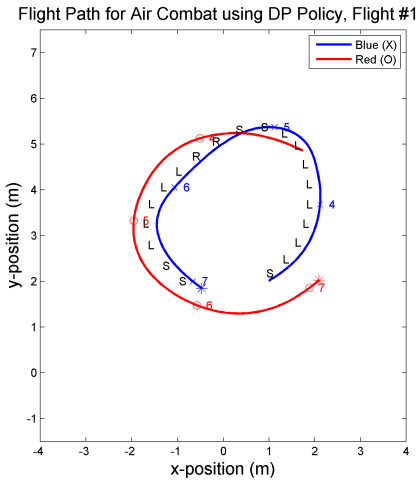


(e) 19.1 s.

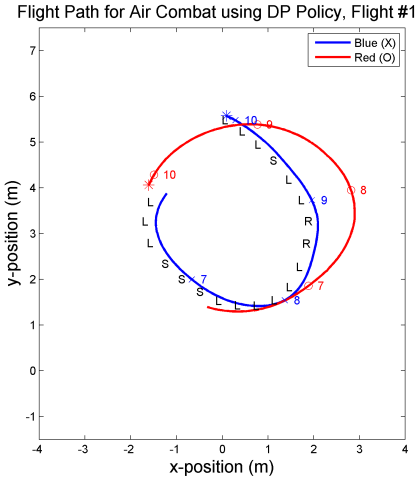
Figure 5-12: Test flight #7 using policy $\pi^{40}_{8\_0}$ against a left turning red aircraft. The red and blue numbers along the respective flight numbers represent seconds. The black letters L, S, and R represent the current blue maneuver selection, which are left, straight, or right, respectively.



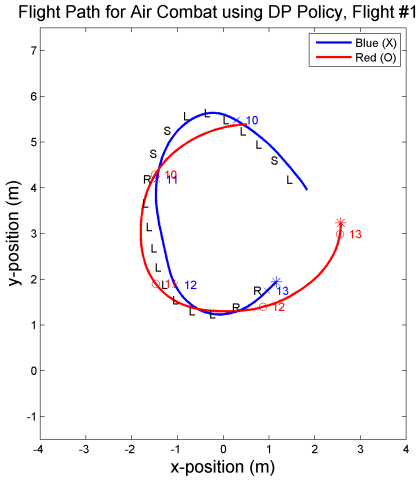
(a) 4.1 s.



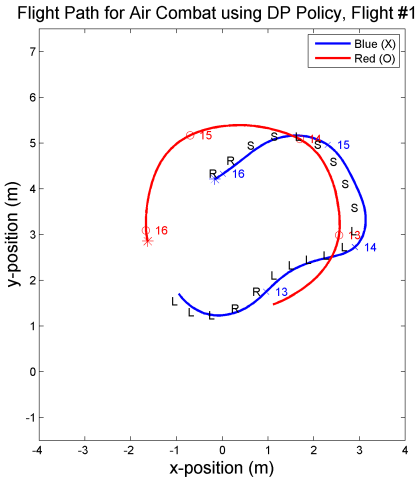
(b) 7.1 s.



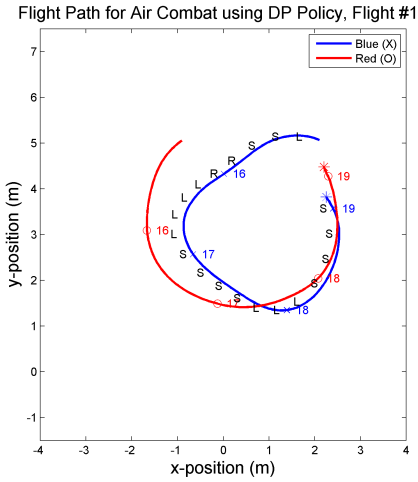
(c) 10.1 s.



(d) 13.1 s.



(e) 16.1 s.



(f) 19.1 s.

Figure 5-13: Test flight #14 using policy $\pi^{40}_{8\_0}$ against a left turning red aircraft. The red and blue numbers along the respective flight numbers represent seconds. The black letters L, S, and R represent the current blue maneuver selection, which are left, straight, or right, respectively.

Chapter 6

Conclusions

6.1 Review of Thesis Objectives

The purpose of this research was to develop a method which enables an autonomous UAS to successfully fly air combat. Several objectives were set to fill gaps found in the current state of the art. The first objective was to develop a method capable of operating in real-time, which is a requirement if there is to be any hope of practical application on operational vehicles. The associated goal of integration with RAVEN platform demonstrates the potential for real-time operation, providing a "trial by fire" to expose the strengths and weaknesses of the proposed solution. The second objective was to develop a method with a relative long planning horizon. The motivation for a long planning horizon is based on the decision processes used by human pilots during air combat. As a human pilot flys combat, near term maneuvering decisions are made within a framework of longer term goals. The consideration of these goals enable smart choices that generate greater longer-term benefit, critical to successful air combat. The third objective minimize the need for intensive human involvement in the learning process encoded strategies; and instead develop an algorithm that could learn appropriate maneuvers on its own. The final objective was to develop a flexible algorithm that was capable of switching rolls from defender to offender during an engagement.

Chapter 2 began by defining a function capable of representing the combat state.

Attempts to produce an improved function by incorporating knowledge from an experienced pilot into the function proved successful. The resulting scoring function produced better results in simulation. However, the method was still limited to short look ahead times. Chapter 3 demonstrated the usefulness of the scoring functions when applied to the air combat problem. Real-time combat flight was demonstrated utilizing a short look ahead. The flights proved the viability of the RAVEN and micro-UAS as a tool for flight testing air combat algorithms. Chapter 4 develops the process used to formulate the air combat game as a dynamic programming problem. The scoring function provided in Chapter 2 was used in developing a goal reward structure for the DP architecture. Approximate DP offers the potential for learning optimal policies, increased look-ahead future planning and real-time implementation. Once the learning algorithm was tuned in Chapter 5, the resulting policies performed well in simulation. The blue policies demonstrated the ability to exploit the known policy of the red aircraft. Additionally, the policies generated by the approximate DP proved to be successful against a variety of red maneuvering strategies. Finally, the policy derived from the approximate DP was implemented on micro-UAS within the RAVEN system. These flight tests proved the ability of the policy to be evaluated in real-time and control an actual aircraft maneuvering against another aircraft.

Within the context of the simplified air combat game used throughout this thesis, the approximate DP method manages to meet the objectives set forth at the beginning of the project. The simulations in Chapter 5 show some very interesting results. The subtle actions taken by the function approximation policy, which improve performance, suggest that the method could learn in even larger state spaces. Some human interaction was required during the learning process to appropriately define the reward function and model the adversary aircraft. However, the model and rewards are defined, the DP learns the strategy required to defeat the opponent. Overall, the results are promising. The simplified air combat game used in this project certainly does not represent the extremely complex problem of real-world flight. An extension of the simplified game is in order, and, as with most research, there are several other directions this thesis could lead in terms of continued work.

6.2 Extensions of this Research

The successful results in solving the air combat game using approximate DP leads to a great potential for future continued research on this topic. First and foremost is to extend the 2-D, fixed velocity combat game. Expanding the game to 3-D would require considerable changes to the flight dynamics. This would require removal of the restriction on bank angle, altitude and velocity. Additionally, throttle and elevator control would need to be incorporated, thus increasing the number of possible control action combinations. The varying velocity and altitude would mean that the energy state of the aircraft would no longer be constant. Energy would need to be incorporated into the scoring function to accurately represent the desirability of a particular combat state. All told, expansion to a 3-D combat game would increase the size of the state space considerably. Careful selection of sample states and efficient programming will be required to learn and extract the maneuvering policy in a reasonable amount of time.

In actual combat training programs designed for human pilots, an incremental approach is taken. First 1-v-1 combat is taught, followed by 2-v-1, 2-v-2, 2-v-4, 4-v-4, and 4-v-X, where X is some large number of threat aircraft. The first step is expansion to 2-v-1 maneuvering, which involves two blue aircraft engaged with one red aircraft. The mechanics are the same, but certain rules need to be followed to ensure deconfliction. Also, some mutually beneficial behavior can result in a quicker win for the blue team than would be the case if both blue aircraft were operating independently. By incorporating the required additional state variables, the approximate DP should be able to learn such cooperative behavior in addition to the basic fighter maneuvering techniques. A similar approach to each increase in complexity could be taken until the policy is capable of maneuvering with large numbers of cooperative aircraft, to maximize effectiveness against a group of red aircraft.

Finally, the issue of imperfect sensor information should be considered. In this thesis, information about the adversary was assumed to be complete and instantaneous. However, data loss, corruption of data, and intermittent coverage certainly

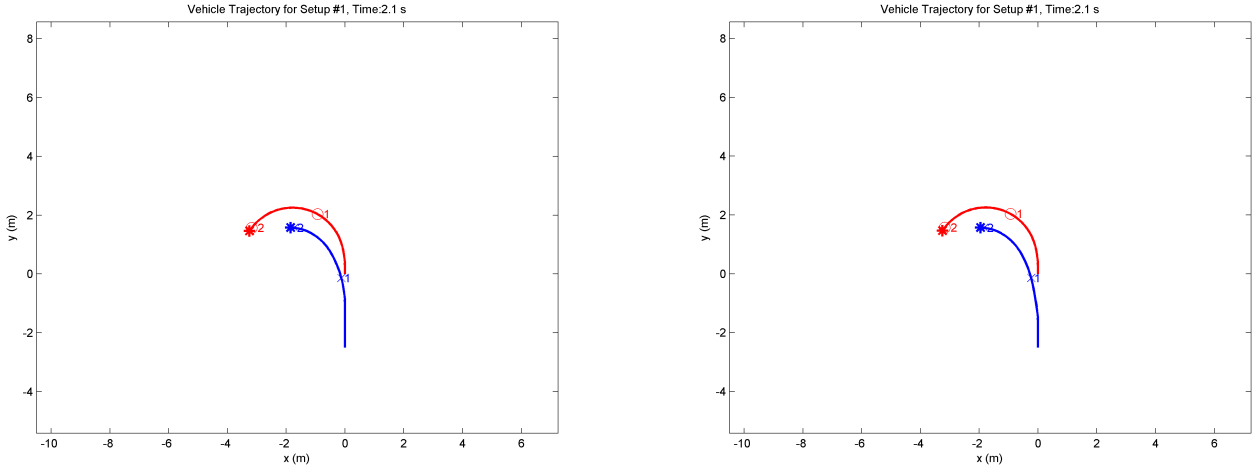
complicate the problem. These issues need to be addressed prior to installation on a combat UAS.

Overall, the method of approximate dynamic programming shows the potential to learn and execute a successful maneuvering policy within the large state space and complicated environment concomitant with air combat. However, the current state of the art has limitations accomplishing this feat. Thus, this thesis represents a significant step toward autonomous air combat.

Appendix A

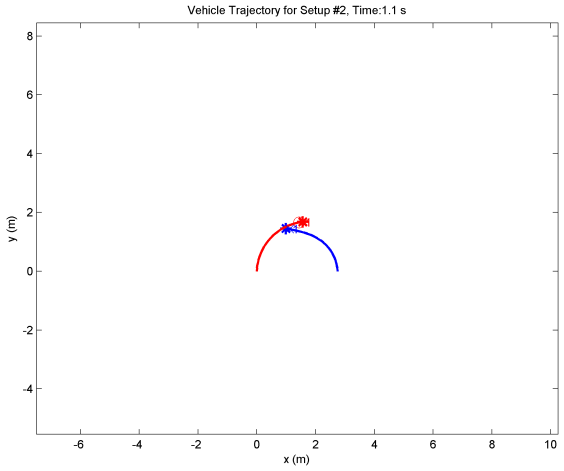
Complete Policy Comparison Simulation Results

The following are the results from flight testing. Complete lines for each of the six setups listed in Table 5.1 are presented here. The left hand lines are from the dynamic programming function approximation policy $\pi^{40}_{8\_0}$, the right hand figures are from the baseline policy π^{nom}_{b}. The adversary aircraft is using π^{nom}_{r} in both cases.

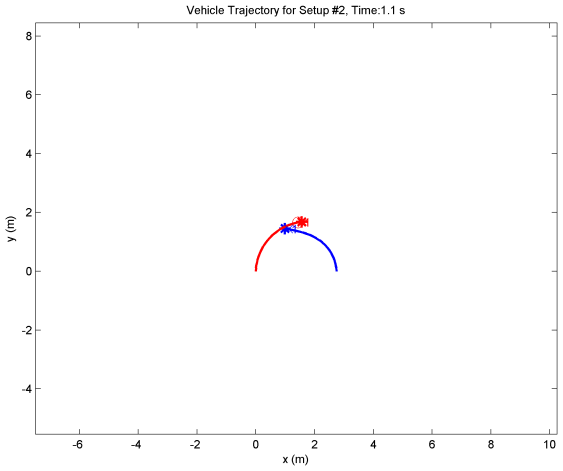


(a) Policy $\pi^{40}_{8\_0}$ (b) Policy π^{nom}_{b}

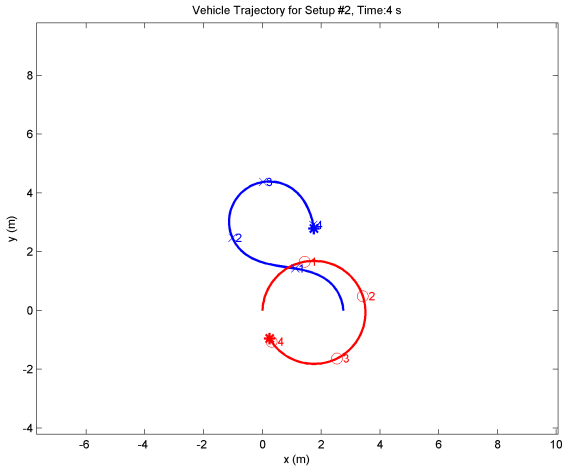
Figure A-1: Setup 1.



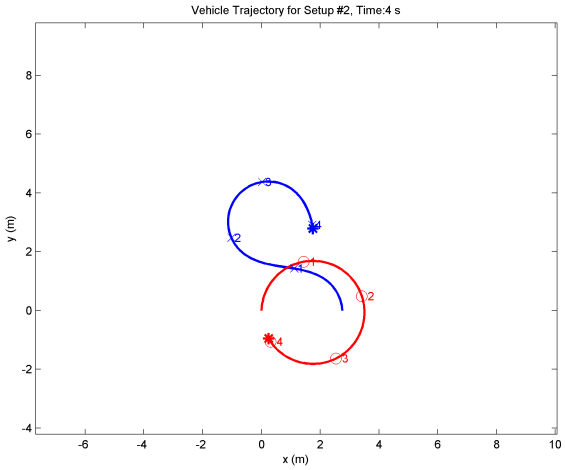
(a) Policy $\pi_{8\_0}^{40}$



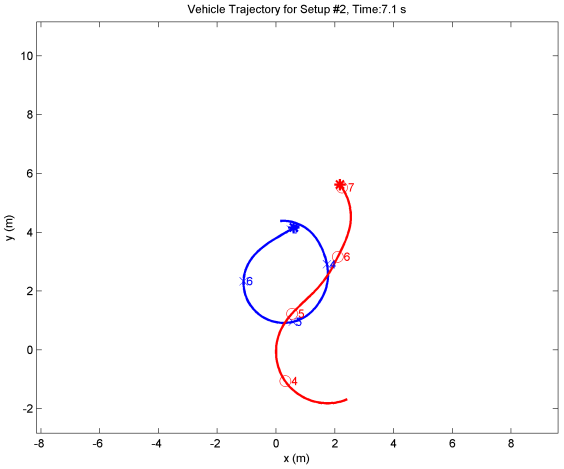
(b) Policy π_b^{nom}



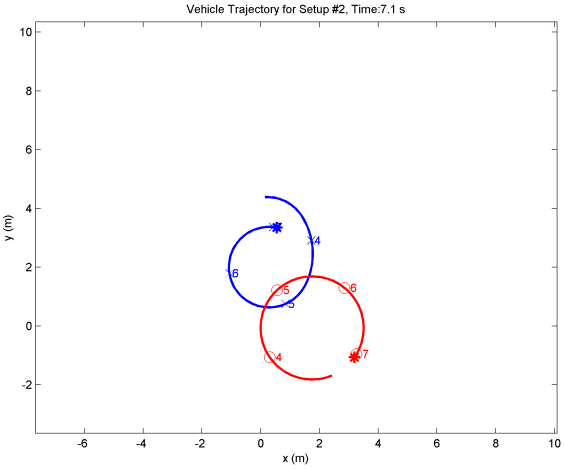
(c) Policy $\pi_{8\_0}^{40}$



(d) Policy π_b^{nom}

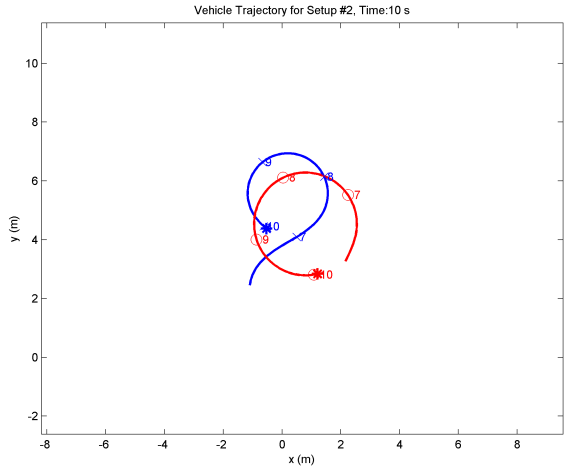


(e) Policy $\pi_{8\_0}^{40}$

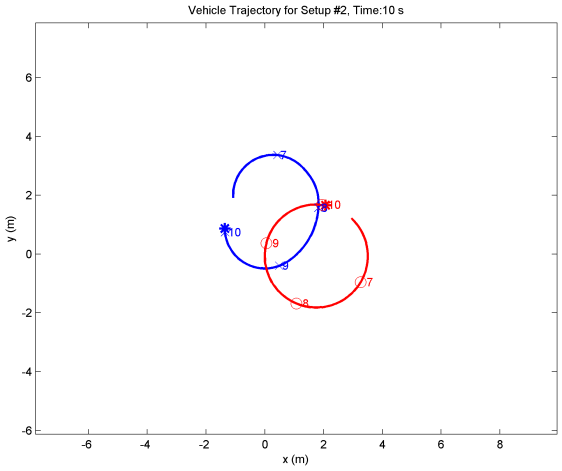


(f) Policy π_b^{nom}

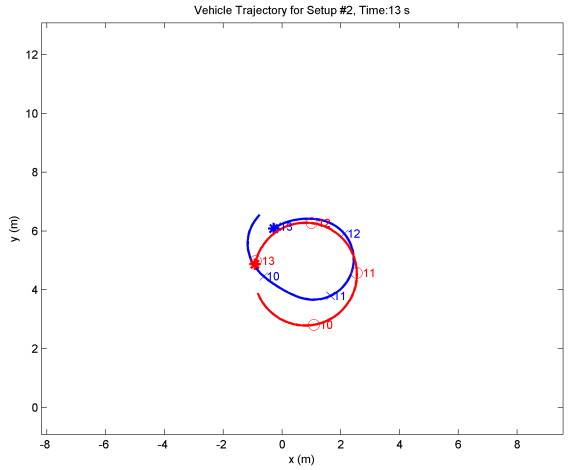
Figure A-2: Setup 2a.



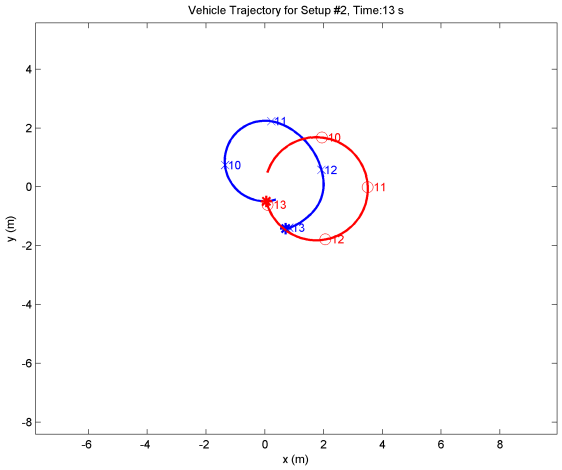
(a) Policy $\pi_{8\_0}^{40}$



(b) Policy π_b^{nom}



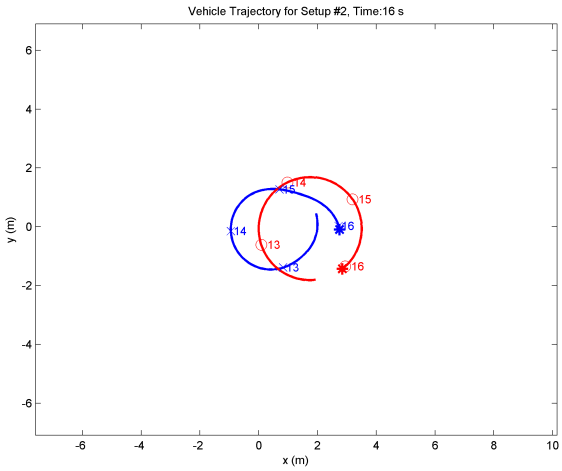
(c) Policy $\pi_{8\_0}^{40}$



(d) Policy π_b^{nom}

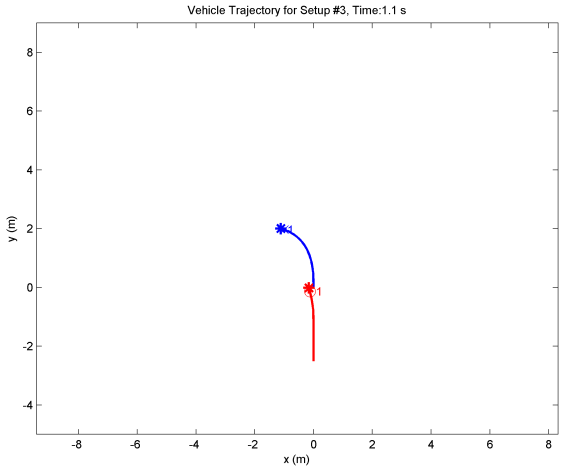
Engagement Complete

(e) Policy $\pi_{8\_0}^{40}$

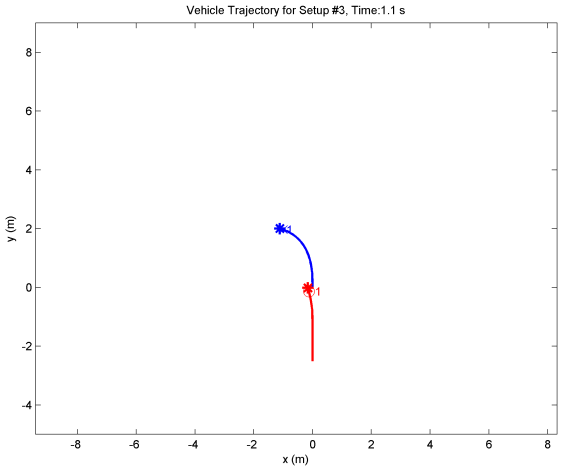


(f) Policy π_b^{nom}

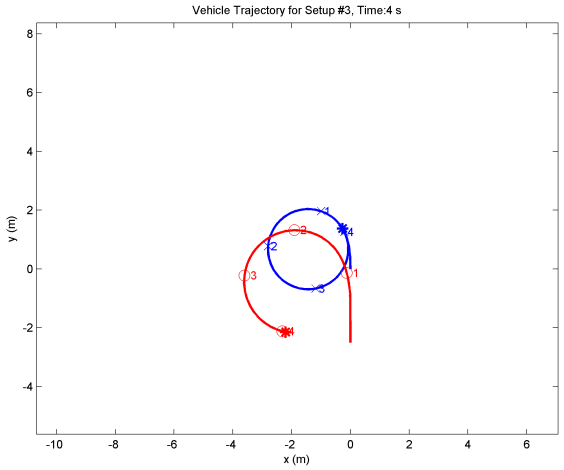
Figure A-3: Setup 2b.



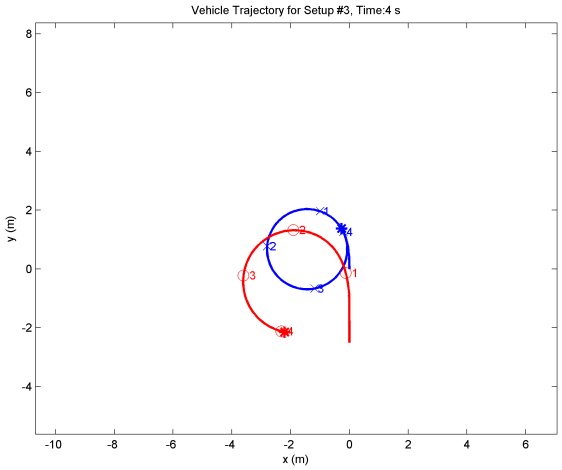
(a) Policy $\pi_{8\_0}^{40}$



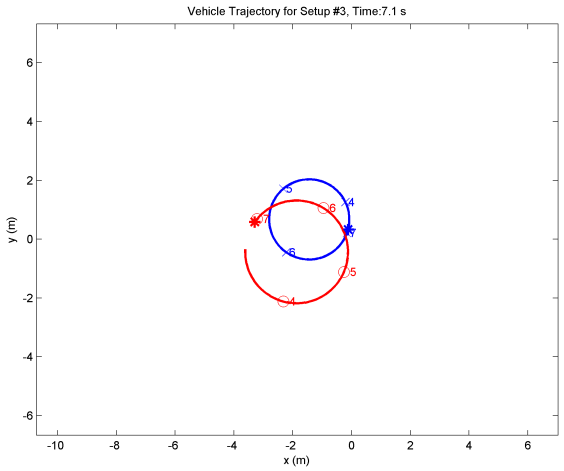
(b) Policy π_b^{nom}



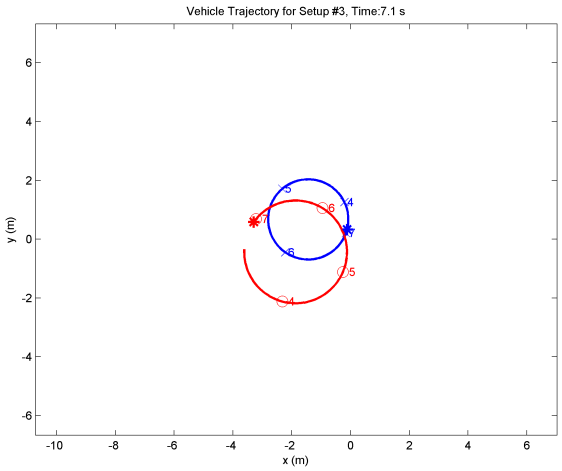
(c) Policy $\pi_{8\_0}^{40}$



(d) Policy π_b^{nom}

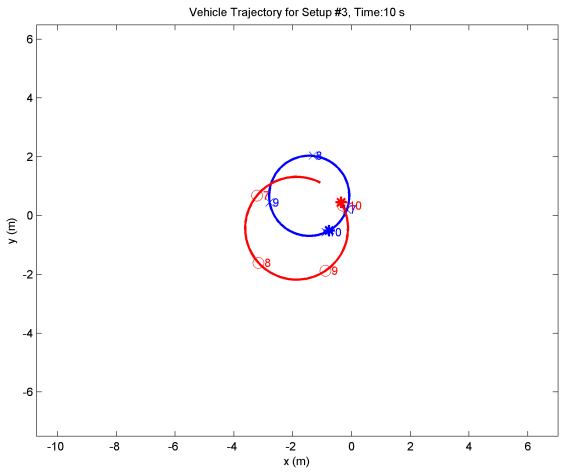


(e) Policy $\pi_{8\_0}^{40}$

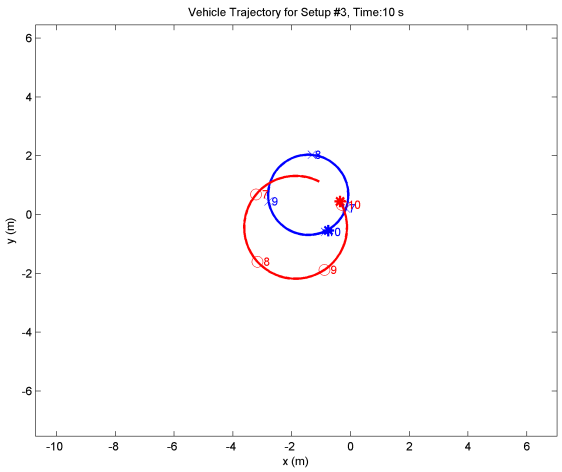


(f) Policy π_b^{nom}

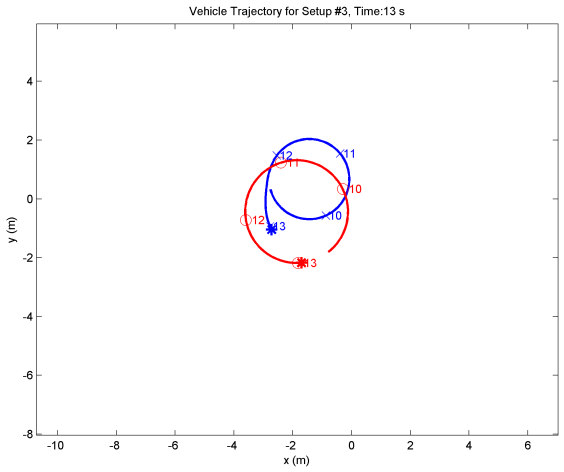
Figure A-4: Setup 3a.



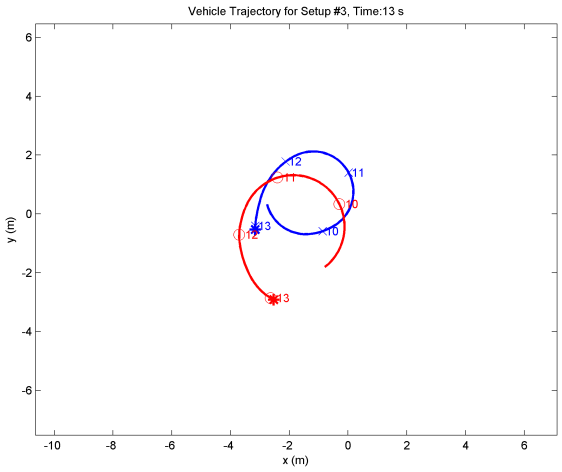
(a) Policy $\pi_{8\_0}^{40}$



(b) Policy π_b^{nom}

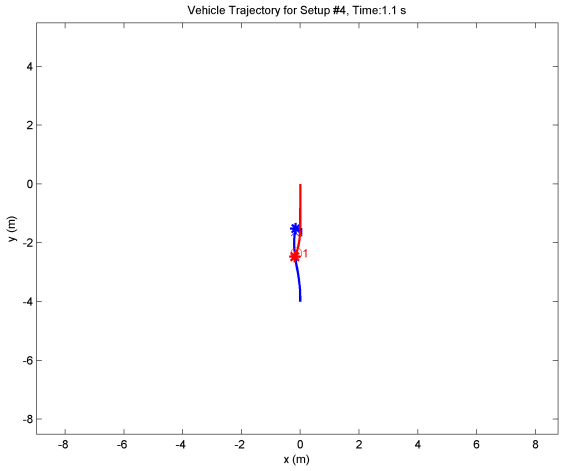


(c) Policy $\pi_{8\_0}^{40}$

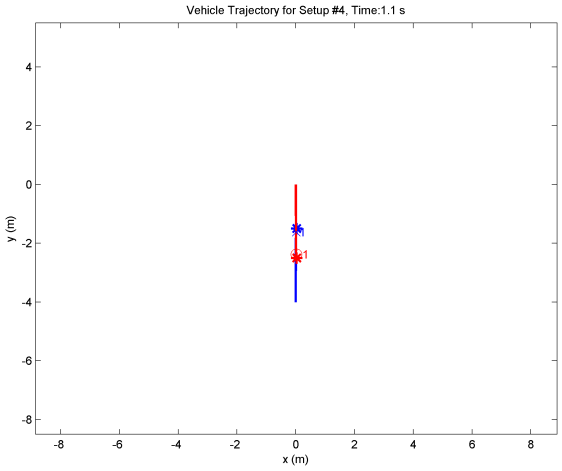


(d) Policy π_b^{nom}

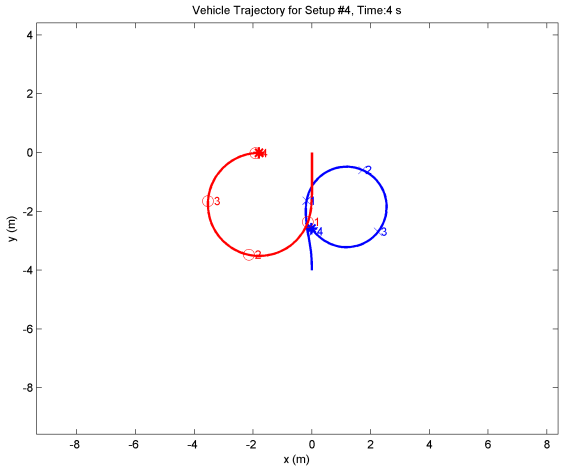
Figure A-5: Setup 3b.



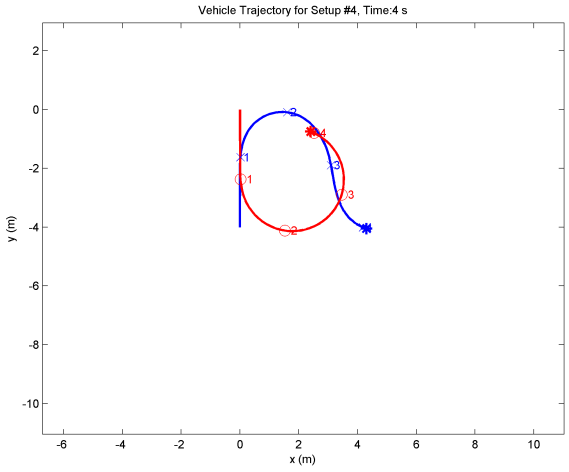
(a) Policy $\pi_{8\_0}^{40}$



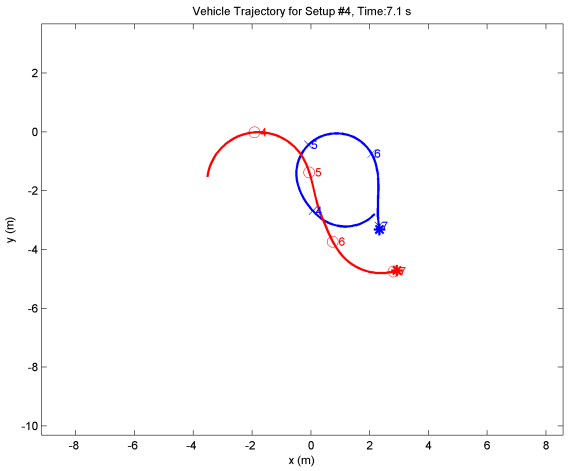
(b) Policy π_b^{nom}



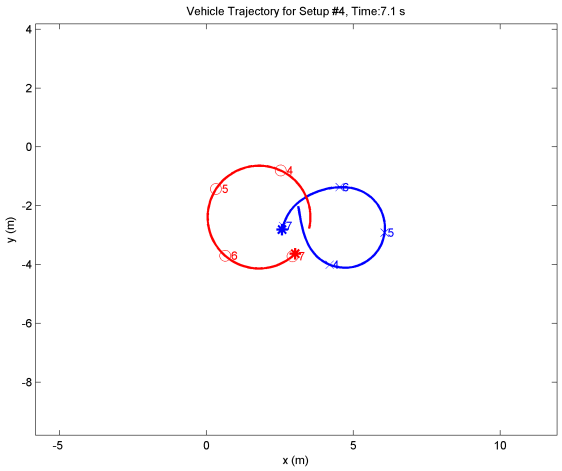
(c) Policy $\pi_{8\_0}^{40}$



(d) Policy π_b^{nom}

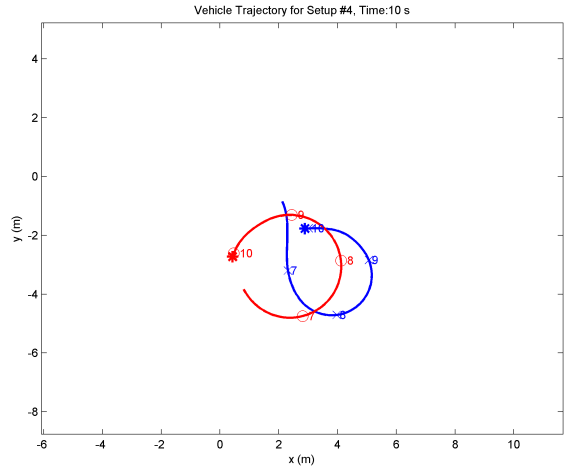


(e) Policy $\pi_{8\_0}^{40}$

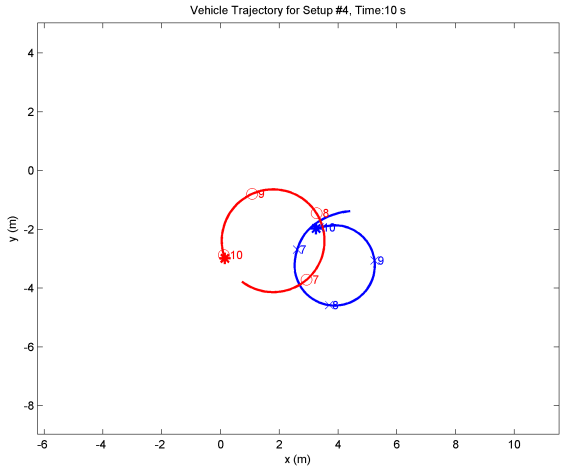


(f) Policy π_b^{nom}

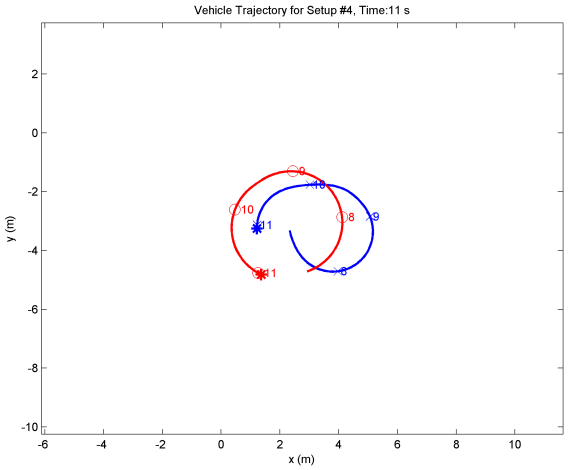
Figure A-6: Setup 4a.



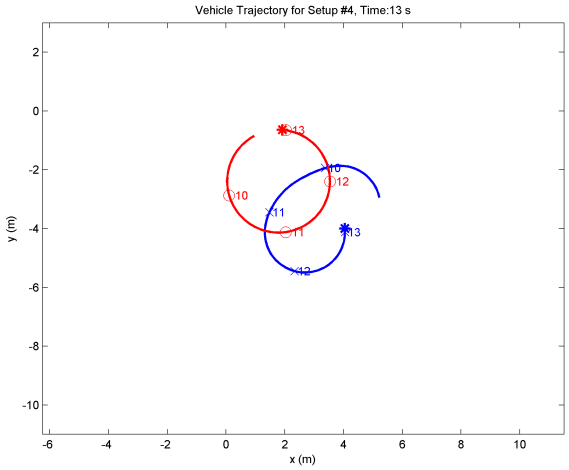
(a) Policy $\pi_{8\_0}^{40}$



(b) Policy π_b^{nom}



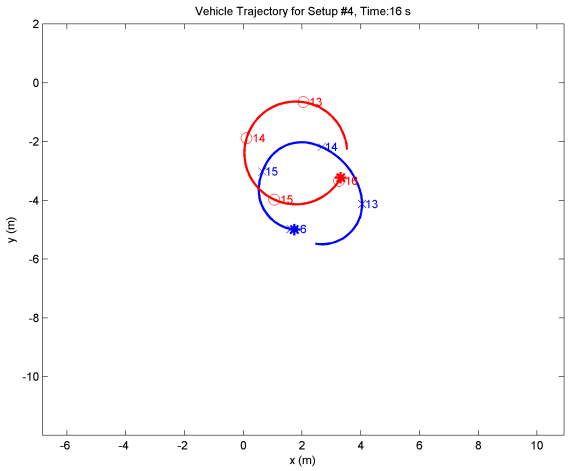
(c) Policy $\pi_{8\_0}^{40}$



(d) Policy π_b^{nom}

Engagement Complete

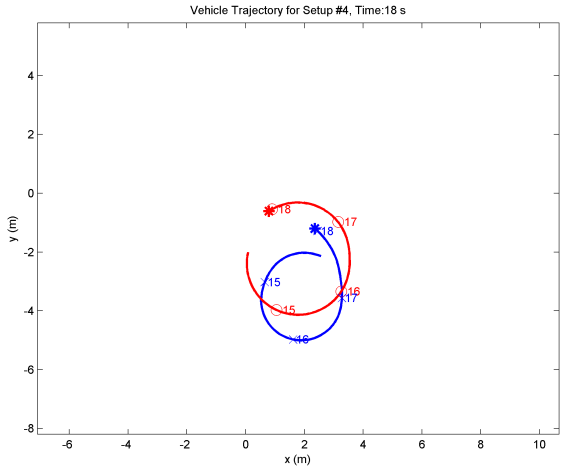
(e) Policy $\pi_{8\_0}^{40}$



(f) Policy π_b^{nom}

Figure A-7: Setup 4b.

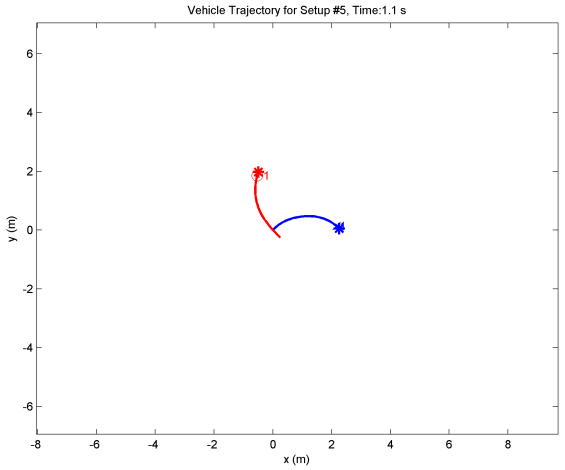
Engagement Complete



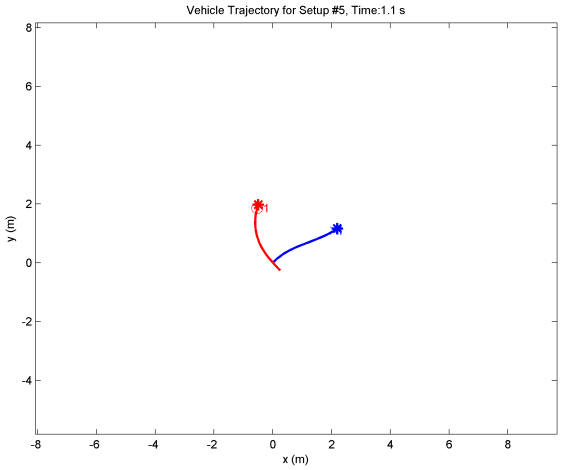
(a) Policy $\pi_{8\_0}^{40}$

(b) Policy π_b^{nom}

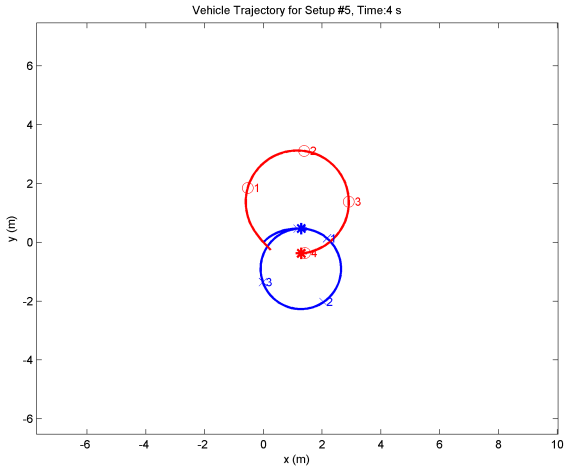
Figure A-8: Setup 4c.



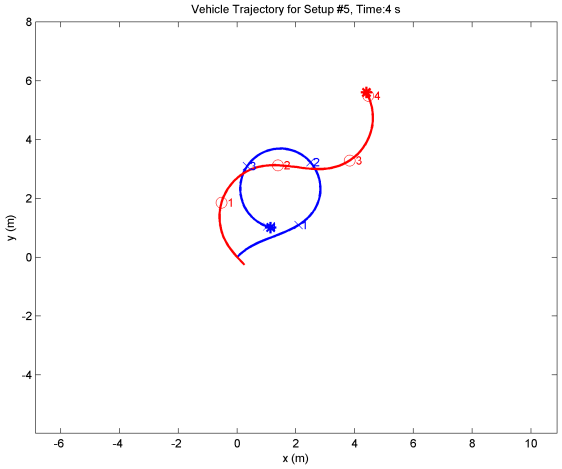
(a) Policy $\pi_{8\_0}^{40}$



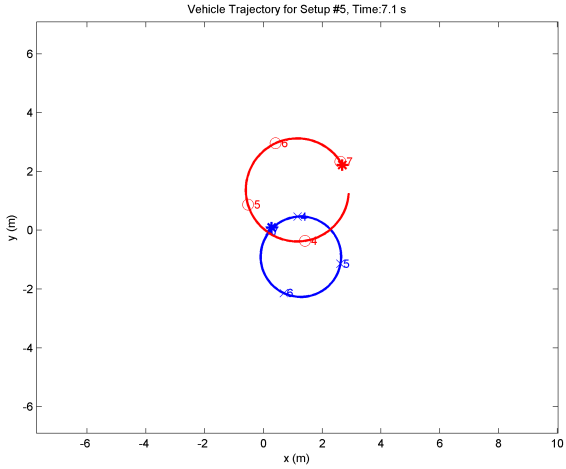
(b) Policy π_b^{nom}



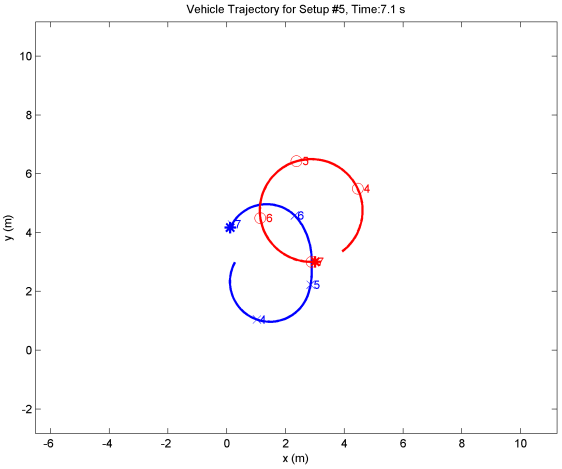
(c) Policy $\pi_{8\_0}^{40}$



(d) Policy π_b^{nom}

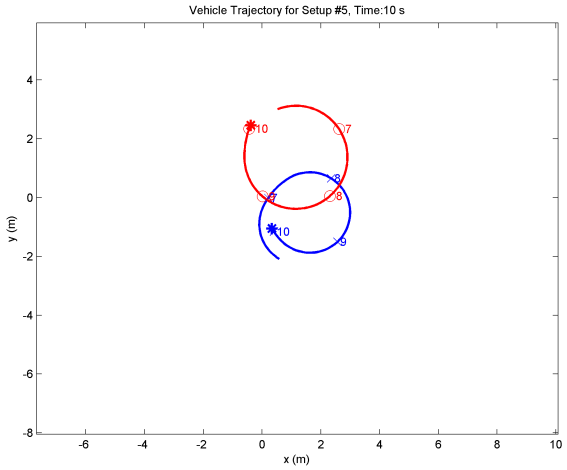


(e) Policy $\pi_{8\_0}^{40}$

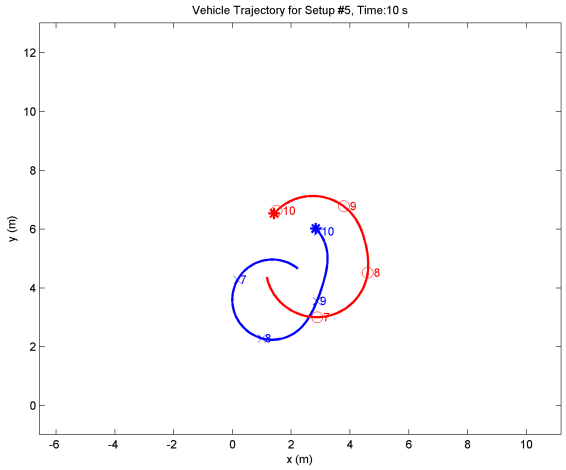


(f) Policy π_b^{nom}

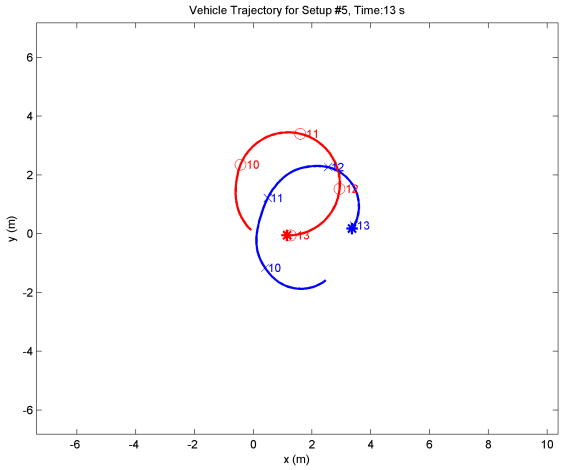
Figure A-9: Setup 5a.



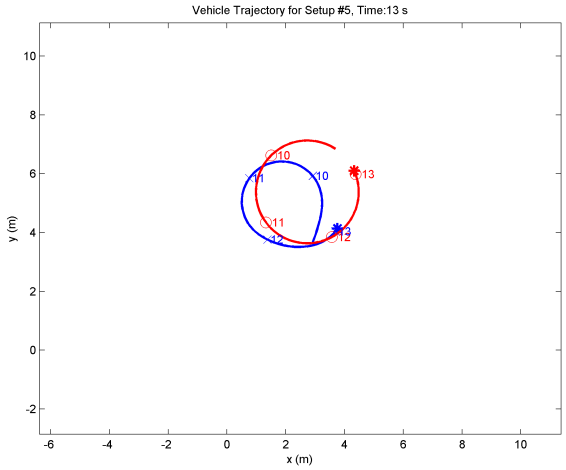
(a) Policy $\pi_{8\_0}^{40}$



(b) Policy π_b^{nom}



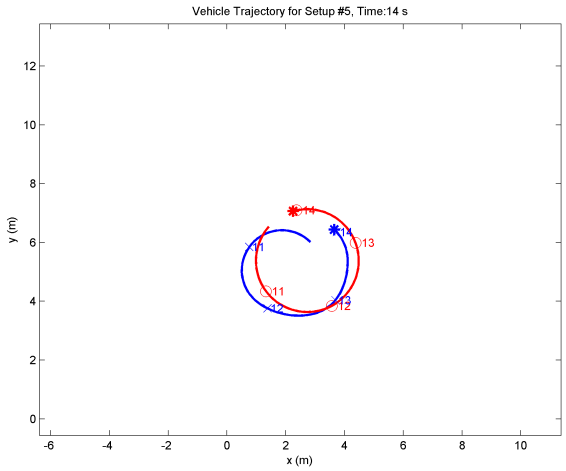
(c) Policy $\pi_{8\_0}^{40}$



(d) Policy π_b^{nom}

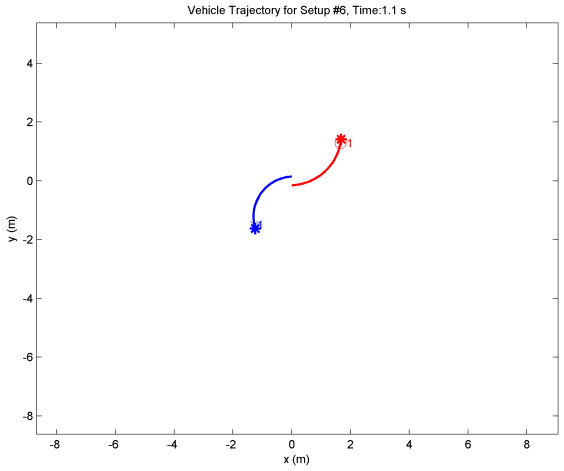
Engagement Complete

(e) Policy $\pi_{8\_0}^{40}$

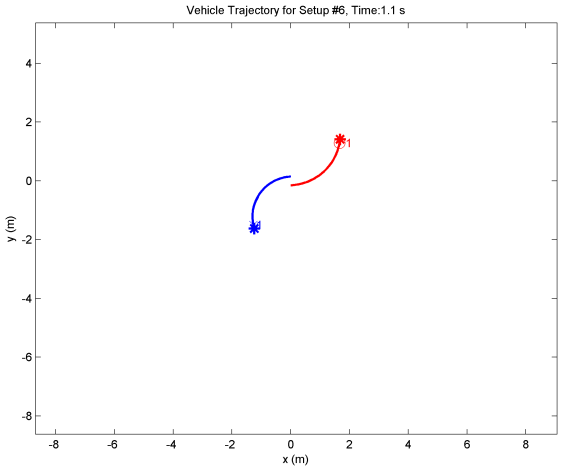


(f) Policy π_b^{nom}

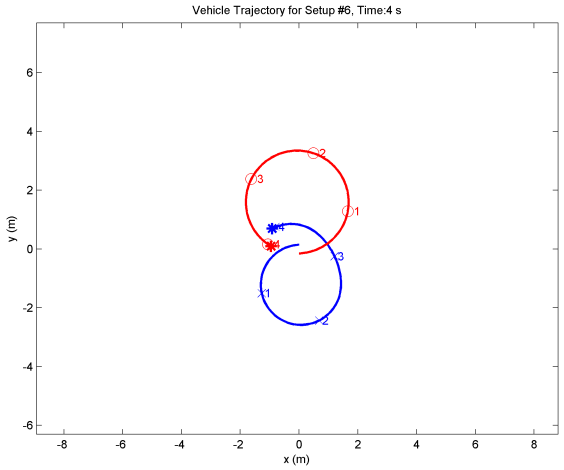
Figure A-10: Setup 5b.



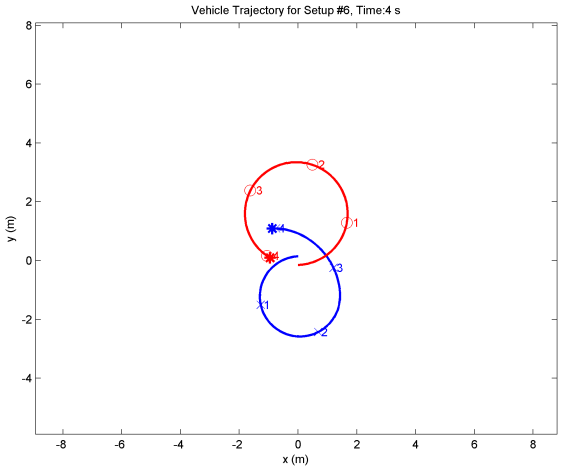
(a) Policy $\pi_{8\_0}^{40}$



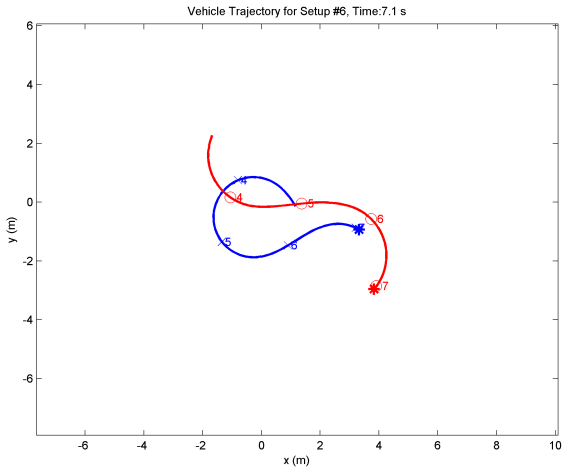
(b) Policy π_b^{nom}



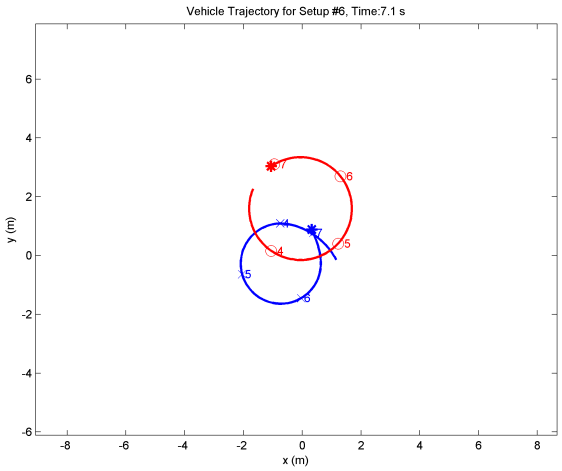
(c) Policy $\pi_{8\_0}^{40}$



(d) Policy π_b^{nom}

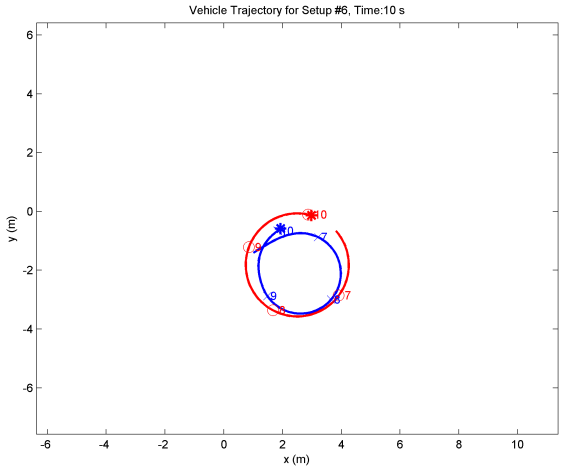


(e) Policy $\pi_{8\_0}^{40}$

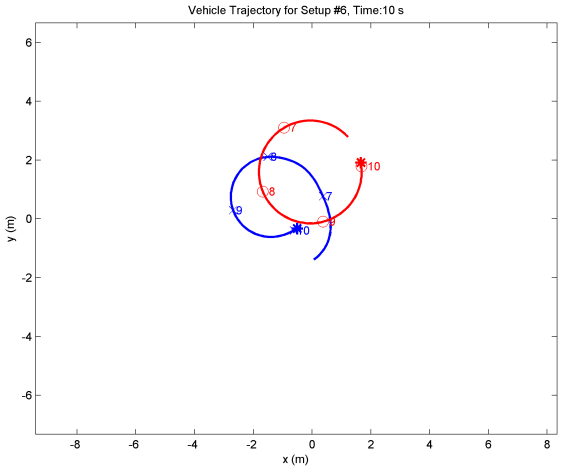


(f) Policy π_b^{nom}

Figure A-11: Setup 6a.

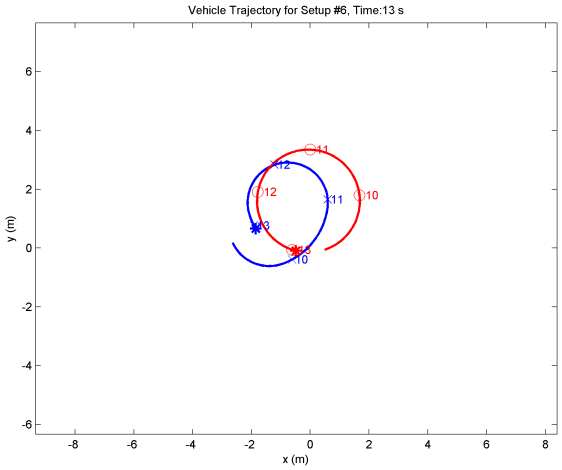


(a) Policy $\pi_{8\_0}^{40}$



(b) Policy π_b^{nom}

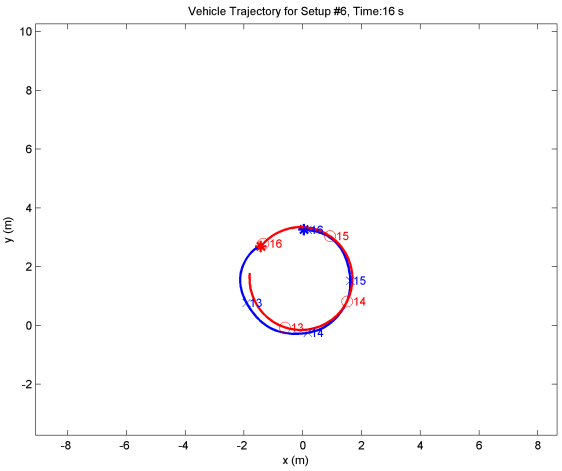
Engagement Complete



(c) Policy $\pi_{8\_0}^{40}$

(d) Policy π_b^{nom}

Engagement Complete

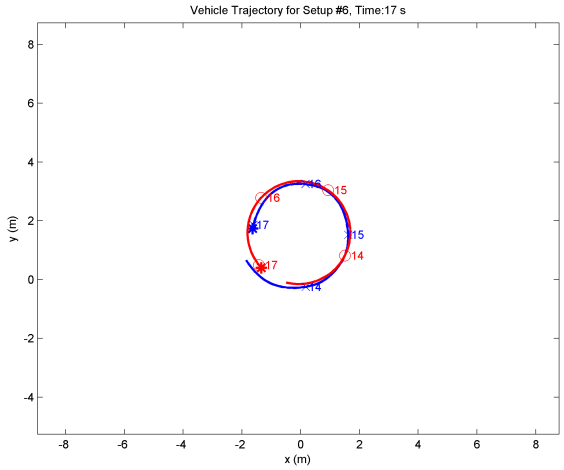


(e) Policy $\pi_{8\_0}^{40}$

(f) Policy π_b^{nom}

Figure A-12: Setup 6b.

Engagement Complete



(a) Policy $\pi_{8\_0}^{40}$

(b) Policy π_b^{nom}

Figure A-13: Setup 6c.

Bibliography

[1] Air Midi Micros. Available at http://www.airmidimicros.com, 2009.

[2] F. Austin, G. Carbone, M. Falco, and H.Hinz. Game Theory for Automated Maneuvering During Air-to-Air Combat. *Journal of Guidance, Control and Dynamics*, 13(6):1143–1149, Nov-Dec 1990.

[3] F. Austin, G. Carbone, M. Falco, and H. Hinz. Automated Maneuvering During Air-to-Air Combat. Technical report, Grumman Corporate Research Center, Bethpage, NY, CRC Rept. RE-742, Nov 1987.

[4] R. Bellman. On the theory of dynamic programming. Technical report, Proceedings of the National Academy of Sciences, 1952.

[5] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[6] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, Massachusetts, 1996.

[7] G.H. Burgin and L.B. Sidor. Rule-Based Air Combat Simulation. Technical report, National Aeronautics and Space Administration, CR-4160, 1988.

[8] A. Frank, M. Valenti J. McGrew, D. Levine, and J. How. Hover, Transition, and Level Flight Control Design for a Single-Propeller Indoor Airplane. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, August 2007.

[9] Homefly. Available at http://www.homefly.com/, 2008.

[10] R.P. Isaacs. Games of pursuit. Technical report, The Rand Corporation, Santa Monica, CA, 17 November 1951.

[11] J. Knutson . F-4U Energy-Maneuverability Diagram, 20,000 feet. Available at http://www.cactus.org/AirWarrior/Library/AWInfo/EMDiagrams/f4u1_20k.gif, 1994.

[12] J.Eklund, J. Sprinkle, H. Kim, and S. Sastry. Implementing and Testing a Nonlinear Model Predictive Tracking Controller for Aerial Pursuit/Evasion Games on a Fixed Wing Aircraft. In *Proceedings of 2005 American Control Conference*, volume 3, pages 1509–1514, June 2005.

[13] K. Virtanen and J. Karelahti and T. Raivio. Modeling Air Combat by a Moving Horizon Influence Diagram Game. *Journal of Guidance, Control and Dynamics*, 29(5):1080–1091, Sep-Oct 2006.

[14] Philipp W. Keller, Shie Mannor, and Doina Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 449–456, New York, NY, USA, 2006. ACM.

[15] M. Valenti, B. Bethke, G. Fiore, J. How, and E. Feron. Indoor multi-vehicle flight testbed for fault detection, isolation, and recovery. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Keystone, CO, August 2006.

[16] MIT Real-time indoor Autonomous Vehicle test ENvironment (RAVEN). RAVEN home page. Available at http://vertol.mit.edu/, 2007.

[17] Naval Air Training Command. CNATRA P-1289 Air Combat Maneuvering. Available at https://www.cnatra.navy.mil//pubs/ppub_t45_str.htm, 2007.

[18] R. Tiron. Can UAVs Dogfight? *Association for Unmanned Vehicle Systems International: Unmanned Systems*, 24(5):39–42, Nov-Dec 2006.

[19] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002.

[20] S. Park, J. Deyst, and J. P. How. A New Nonlinear Guidance Logic for Trajectory Tracking. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Providence, RI, August 2004.

[21] R. Shaw. *Fighter Combat Tactics and Maneuvering*. Naval Institute Press, Annapolis, Maryland, 1985.

[22] J. Sprinkle, J.Eklund, H. Kim, and S. Sastry. Encoding Aerial Pursuit/Evasion Games with Fixed Wing Aircraft into a Nonlinear Model Predictive Tracking Controller. In *43rd IEEE Conference on Decision and Control*, December 2004.

[23] The Math Works. Neural Network Toolbox newpnn article. Available at http://www.mathworks.com, 2008.

[24] M. Valenti, B. Bethke, D. Dale, A. Frank, J. McGrew, S. Ahrens, J. P. How, and J. Vian. The MIT Indoor Multi-Vehicle Flight Testbed. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation (ICRA '07)*, Rome, Italy, April 2007. Video Submission.

[25] Vicon Company. Vicon Motion Capture Systems. Available at http://www.vicon.com/, 2007.