Air-Combat Strategy Using Approximate Dynamic Programming

Air Combat Strategy using Approximate Dynamic Programming

James S. McGrew* and Jonathan P. How†

Aerospace Controls Laboratory

Massachusetts Institute of Technology

Cambridge, MA, 02139

and

Lawrence A. Bush‡, Brian Williams§ and Nicholas Roy¶

Computer Science and Artificial Intelligence Laboratory

Massachusetts Institute of Technology

Cambridge, MA, 02139

Unmanned Aircraft Systems (UAS) have the potential to perform many of the dangerous missions currently flown by manned aircraft. Yet, the complexity of some tasks, such as air combat, have precluded UAS from successfully carrying out these missions autonomously. This paper presents a formulation of a level flight, fixed velocity, one-on-one air combat maneuvering problem and an approximate dynamic programming (ADP) approach for computing an efficient approximation of the optimal policy. In the version of the problem formulation considered, the aircraft learning the optimal policy is given a slight performance advantage. This ADP approach provides a fast response to a rapidly changing tactical situation, long planning horizons, and good performance without explicit coding of air combat tactics. The method's success is due to extensive feature development, reward shaping and trajectory sampling. An accompanying fast and effective rollout based policy extraction method is used to accomplish on-line implementation. Simulation results are provided that demonstrate the robustness of the method against an opponent beginning from both offensive

*S.M., Department of Aeronautics and Astronautics, jsmcgrew@alum.mit.edu

†Professor of Aeronautics and Astronautics, jhow@mit.edu, Associate Fellow AIAA

‡Ph.D. Candidate, Department of Aeronautics and Astronautics, bushL2@csail.mit.edu

§Professor of Aeronautics and Astronautics, williams@mit.edu.

¶Assistant Professor of Aeronautics and Astronautics, nickroy@mit.edu.

and defensive situations. Flight results are also presented using micro-UAS flown at MIT's Real-time indoor Autonomous Vehicle test ENvironment (RAVEN).

Nomenclature

x	=	state vector
x^n	=	n^{th} state vector in X
x_b^{pos}	=	blue x coord. in $x-y$ plane
y_b^{pos}	=	blue y coord. in $x-y$ plane
X	=	set of state vector $[x^1, x^2, \ldots, x^n]^T$
$f(x,u)$	=	state transition function
$\pi(x)$	=	maneuvering policy
$\pi^*(x)$	=	optimal maneuvering policy
$\pi^N_{approx}(x)$	=	approximate maneuvering policy generated directly from $J^N_{approx}(x)$
$\bar{\pi}^N_{approx}(x)$	=	approximate maneuvering policy generated via rollout from $J^N_{approx}(x)$
$J(x)$	=	future reward value of state x
$J^k(x)$	=	k^{th} iteration of $J(x)$
$J^*(x)$	=	optimal value of $J(x)$
$\hat{J}(X)$	=	$[\hat{J}(x^1), \hat{J}(x^2) \ldots \hat{J}(x^n)]^T$
$J_{approx}(x)$	=	function approximation form of $J(x)$
$J^N_{approx}(x)$	=	function approximation form of $J^k(x)$ after k=N iterations
$\hat{J}(x)$	=	scalar result of Bellman backup on x
$S(x_b)$	=	scoring function evaluated for blue
γ	=	future reward discount factor
u	=	control or movement action
$\phi(x)$	=	feature vector of state x
$\Phi(X)$	=	$[\phi(x^1), \phi(x^2), \ldots, \phi(x^n)]^T$
β	=	function parameters vector
$g(x)$	=	goal reward function
$g_{pa}(x)$	=	position of advantage goal reward
p_t	=	probability of termination function
T	=	Bellman backup operator

I. Introduction

Despite long range radar and missile technology improvements, modern fighter aircraft (e.g., F/A-22, F-35, and F-15) are still designed for close combat and military pilots are still trained in air combat basic fighter maneuvering (BFM). Unmanned Aircraft Systems (UASs) 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 solely accomplished by manned platforms. One approach to using Unmanned Aircraft (UA) for air combat is to pilot the aircraft remotely, as was first accomplished by an MQ-1 Predator UAS in 2002[1]. Remote operations has the significant benefit of removing the pilot from harm's way. However, a one-to-one pilot-to-aircraft ratio, which does not fully leverage the strengths of combat UAS. Furthermore, current pilot-vehicle interface technology leaves a UAS pilot at a great disadvantage to a manned platform in the BFM arena due to limited situational awareness. It seems clear that if a UAS is ever going to fulfill the air combat missions performed by these manned aircraft the ability to fly BFM will be a requirement.

Automating BFM reduces the pilot workload, removes the need to supply the pilot with complex spatial orientation cues, and reduces bandwidth issues related to time delay and signal jamming. The pilot is, therefore, allowed additional capacity for supervisory tasks such as consent to fire and monitoring other airborne vehicles[a].

The purpose of the research is to develop a solution technique for computing near-optimal UAS BFM decisions. While the proposed solution may require off-line training, it must be capable of producing decisions in real-time when implemented. To achieve near-optimal decision making, a long planning horizon must be used. For example, human pilots make near-term maneuvering decisions within a framework of longer term goals, which is critical to successful air combat. However, the necessary computations are not possible to perform on-line using current techniques. Flight maneuver decisions can be planned efficiently by applying approximate dynamic programming (ADP). The proposed ADP formulation provides a fast response to a rapidly changing tactical situation, long planning horizons, and good performance without explicit coding of air combat tactics. A simplified simulated air combat problem was used which restricts the vehicles to level flight and constant velocity. Additionally, the friendly, or blue, aircraft was given a slight performance advantage in the simulation to facilitate policy learning; this is a technique commonly used in manned aircraft combat training. These simulation results showed a significant 18.7% improvement over the current state of the art for this problem formulation. Additionally, actual micro-UAS flight results are presented using Real-time indoor Autonomous Vehicle test ENvironment (RAVEN)[2,3].

[a]The lead 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.

I.A. Approach Summary

The goal of air combat is to maneuver your aircraft into a position of advantage over the other aircraft, from either an offensive or defensive starting position, while minimizing risk to your own aircraft. Achieving this goal requires selecting control actions (e.g., desired roll rate), given the vehicle dynamics of both aircraft and an assumed adversary strategy. The research objective is to develop a method that can make maneuvering decisions on-line in real-time, can incorporate a long planning horizon, has the ability to compute control sequences of desirable maneuvers without direct expert pilot inputs, and allows switching from pursuit to evasion roles during an engagement. Dynamic programming[4] (DP) is a general purpose planning technique that 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 solution is capable of producing good results in a finite time. The key contribution of this paper is to show that real time autonomous air combat with performance comparable to human decision making is achievable by employing approximate dynamic programming[5] (ADP) to air combat. A variety of techniques were applied to accomplish this, including extensive feature development, trajectory sampling, reward shaping and an improved policy extraction technique using rollout. Finally, to facilitate real-time operation, a neural net classifier was utilized to model the adversary aircraft maneuvering policy.

I.B. Literature Review

Air combat has been explored by several researchers in the past. The optimal solution to a general pursuer-evader game was first defined in Ref. [6], and this seminal work led to the principle of optimality and dynamic programming[4]. However, subsequent application of dynamic programming to air combat has been limited due to computational complexity and the need for fast computation during aerial combat. For example, Virtanen *et al.*[7] modeled air combat using an influence diagram, which could be solved using dynamic programming. Although they demonstrated sensible control choices in real-time, they used a limited planning horizon to mitigate the computational complexity. Long planning horizons are essential to making good (non-greedy) maneuver choices during air combat.

Other approaches to planning flight maneuvers include limited search, rule-based methods and nonlinear model predictive control. Austin *et al.*[8,9] suggests using a game theoretic approach involving a recursive search over discrete maneuver choices to maximize a heuristic scoring function with a fixed planning horizon. Austin *et al.*[8,9] demonstrate the feasibility of real-time autonomous combat in simulation. The authors state that the maneuver selection only guaranteed optimality in the short term, and only with respect to the chosen heuristic scoring function. Even so, the method produced some maneuvering decisions similar to

those made by experienced human pilots. Burgin and Sidor developed a rule-based Adaptive Maneuvering Logic Program[10], which was successful in simulated air combat against human adversaries. The authors' foray into rule-based control generated insight into the complexity of real-life air combat and an appreciation for algorithm evaluation using skilled human pilots. The rule-based method requires hard 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. Finally, Ref. [11,12] presented a nonlinear, model predictive tracking control that implemented a real-time game theoretic evasion controller for fixed wing aircraft. The authors formulated the control problem as a cost optimization with input and state constraints, which was solved using a computationally fast gradient-descent method. The authors commented on the need to encode proven aircraft maneuvering tactics (from Ref. [13]) into the cost functions because the method, by itself, did not produce the required behaviors. This need suggests a reliance on some previous knowledge of the maneuvers required to be successful. Additionally, the demonstrated algorithm did not have the ability to switch between pursuit and evasion roles.

While the aforementioned approaches achieved some success, the goal is to improve upon them in terms of real-time implementation, increased planning horizons, and increased optimality without explicitly coding air combat tactics or relying on expert human involvement. This objective is achieved via approximate dynamic programming (ADP), more specifically two particular ADP techniques: rollout with an approximate value function representation. Both have been applied to problems unrelated to air combat. Rollout[14] was introduced as a Monte Carlo policy evaluation technique, and the literature includes subsequent examples of using rollout to improve upon an existing policy[15] or heuristic[16]. Approximate value function representation was introduced in Ref.[17] as a way to extend the use of dynamic programming to large state space problems. There are many examples that use dynamic programming with an approximate value function[5,18,19]. Combining rollout with an approximate value function was done in Ref. [20], but the approach taken in this paper combines rollout with an approximate value function and applies it to air combat.

II. Approximate Dynamic Programming Method

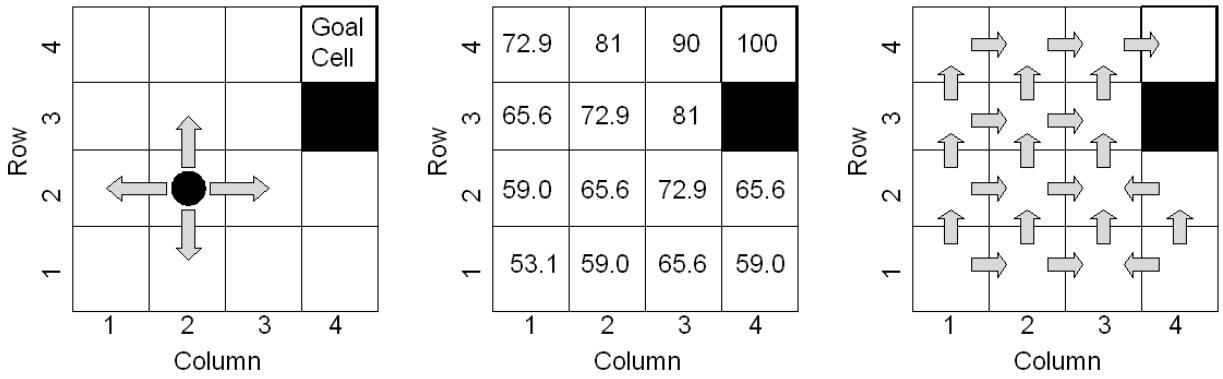
Given a model of vehicle dynamics and an approximate objective function, dynamic programming (DP) provides the means to precisely compute an optimal maneuvering strategy for the proposed air combat game. The resulting strategy or *policy* provides the best course of action given any game state (i.e., the relative positions of two vehicles engaged in combat), eliminating the need for extensive on-line computation. Dynamic programming is uniquely suited to the air combat problem and the goals set forth for this research. The resulting DP policy is fast to evaluate, thus allowing real-time implementation. Additionally, a long planning horizon can be utilized while computing the maneuvering policy. Furthermore, the model based dynamic programming formulation allows for computation of actual maneuvers, which eliminates the requirement to hard-code specific maneuvers as is typically required in rule-based approaches.

Computing the optimal policy using exact DP is intractable because of the exponential growth of the state space size with the number of state space variables. Approximate dynamic programming (ADP)[5] can be used to reduce the computations required to produce a solution to such complex problems. The current section reviews ADP using a pedagogical example and motivates the need for an approximate solution. The section concludes with a detailed explanation of how ADP is applied to air combat.

II.A. Dynamic Programming Example

Dynamic programming (DP) was first introduced by Bellman[4]. An example shortest path DP problem shown in Figure 1 will be used in this section to define terminology and methods used in future sections. A robot, represented by the circle in Figure 1(a), is capable of 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 at each timestep is defined by the $[row,\ column]$ coordinates in the state vector $x_i = [row_i,\ col_i]$. A state transition function $f(x, u)$ is defined that computes the next state of the game, given a control action. The state transition function models the dynamics of movement and enforces the constraints of the game, that the robot cannot move outside the grid or to the blocked square.

The objective of DP is to determine a movement strategy that results in the shortest path to the goal from any location. This is accomplished by computing the optimal future reward value of each state, $J^*(x)$. The goal state for the problem shown in Figure 1 square



(a) Shortest path problem. (b) J^* future reward value of each state for $g(4,4) = 10$ and $\gamma = 0.9$. (c) π^* optimal movement policy.

Figure 1. Example shortest path problem solved using dynamic programming.

(4,4). The reward for success defined by the function $g(x)$:

$$g(x) = \begin{cases} 10, & \text{if } x = [4,4] \\ 0, & \text{else} \end{cases} \tag{1}$$

A *value function* $J(x)$ is defined at each state representing the approximate future reward value, for starting at that state and applying actions according to a control policy π. The initial value of $J(x)$ is set to zero, such that $J^0(x) = 0$ for all x. This optimal future reward function can be computed by repeatedly performing a Bellman backup[21] on each state using Equation 2. The optimal future reward value will be referred to as $J^*(x)$. The Bellman backup operator, T is defined as:

$$J^{k+1}(x) = TJ^k(x) = \max_u [\gamma J^k(f(x,u)) + g(x)] \tag{2}$$

where $\gamma < 1$ is the discount factor applied at each step. The vector x can also be replaced by a set of states, X, to accomplish Bellman backup operations on a number of states simultaneously. Additionally, x^n refers to the n^{th} state vector when referring to set of states $X = [x^1, x^2, \ldots, x^n]^T$. After performing multiple Bellman backup operations, $J^k(x)$ converges to the optimal value $J^*(x)$, see Figure 1(b).

An optimal policy, π^* can then be computed from $J^*(x)$, where the optimal action at time-step i is defined as:

$$u_i \quad = \quad \pi^*(x_i) = \arg\max_u \left[g(x_i) + \gamma J^*(f(x_i,u)) \right]$$

The optimal policy π^* provides the shortest path move from any given state, see Figure 1(c).

This discrete two dimensional path planning problem has very few states. Unfortunately, the required number of discrete states for typical real-world problems make exact DP impractical.

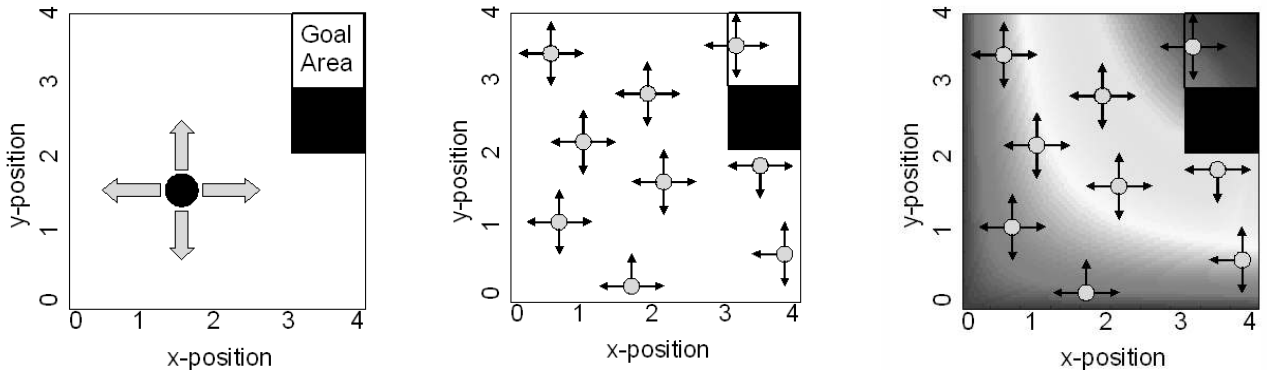
II.B. Approximate Dynamic Programming Example

ADP uses a continuous function to approximately represent the future reward over the state-space[5]. A continuous function approximator eliminates the need to represent and compute the future reward for every discrete state. The function approximator requires many fewer parameters to represent the value function of a high-dimensional state space than would be required for a table lookup in a discrete representation. By reducing the number of parameters, the amount of time required to compute the optimal parameter values is also reduced. For example, the simple shortest path problem in Figure 1 can be redefined with continuous values for the coordinates (see Figure 2). The components of x can now take any value between 0 and 4. $J(x)$, which is conceptually a table of values at discrete points, is replaced by $J_{approx}(x)$, a continuous function that approximates the future reward of states. The state transition function $f(x, u)$ is redefined to allow movements from any arbitrary point using the velocity of the robot, v, and $v\Delta t$ as an estimate of the distance traveled after 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 2(b). The set of samples states will be used repeatedly in place of the discrete states used in the DP problem. The set of state samples will be referred to as X. A Bellman backup operator (T) is applied to each state sample as in Equation 2. The resulting values are stored in target vector $\hat{J}^{k+1}(X)$:

$$\hat{J}^{k+1}(X) = TJ^{k}_{approx}(X) \tag{3}$$

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 vector $\phi(x)$ 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 III.C. Each feature contained in $\phi(x)$ produces a scalar value for a given state. Therefore, with m features the $\phi(x)$ vector will contain m 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



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

problem is the Euclidean distance from the robot to the goal location. The feature sets are computed for all state samples $x^i \in X$ and stored in Φ so that:

$$\Phi(X) = \left[\phi(x^1)\ \phi(x^2)\ \ldots\ \phi(x^n)\right]^T \tag{4}$$

The new $J_{approx}(x)$ can now be computed using standard least squares estimation as follows[5]:

$$\beta^{k+1} = (\Phi^T\Phi)^{-1}\Phi^T\hat{J}^{k+1}(X) \tag{5}$$

J_{approx} is computed as:

$$J^{k+1}_{approx}(x) \equiv \phi(x)\beta^{k+1} \tag{6}$$

where β^{k+1} are the value function parameters computed in Equation 5. The function J^{k+1}_{approx} can now be used to evaluate the future reward of any state x. 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. After sufficient iterations, $\hat{J}^{k+1}(x)$ will approach $J^*(x)$, this function is refered to as $J^*_{approx}(x)$. Figure 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 3.

Additional discussion on this function approximation method are in Ref. [5]. Obviously this method for solving an approximate DP can be extended to problems with much larger state spaces than in this example. Bellman refers to the "curse of dimensionality"[4] as additional dimensions are added to the state space; large state spaces are typically required to represent real-world problems such as air combat. The ADP architecture relieves some of the difficulty associated with this curse in classical DP techniques, and the next section uses

ADP to solve the air combat game.

III. ADP Applied to Air Combat

In this section ADP is applied to the air combat game. First, the system states, goal, control inputs and dynamics are described. Next, the control policy learning is discussed followed by policy extraction methods.

III.A. States, Goal, Control Inputs and Dynamics

The air combat system state x is defined by the position, heading and bank angle

$$x = [x_b^{pos}, y_b^{pos}, \psi_b, \phi_b, x_r^{pos}, y_r^{pos}, \psi_r, \phi_r]^T \tag{7}$$

of a blue aircraft (denoted by the b subscript) and a red aircraft (denoted by the red subscript). The position variables of the aircraft (x_b^{pos}, y_b^{pos}, x_r^{pos}, and y_r^{pos}) have no limits, thus allowing for flight in any portion of the x–y plane. The aircraft bank angle is limited based on the maximum capabilities of the actual UAS and the desire to limit the maximum turning capabilities of the aircraft, see Section IV.A for the actual limits used. The heading variables are allowed to take any value between $\pm 180°$.

The goal of the blue aircraft is to attain and maintain a *position of advantage* behind the red aircraft. The terms Aspect Angle (`AA`) and Antenna Train Angle (`ATA`) are used to quantify this position, see Figure 3. Additionally, Heading Crossing Angle (`HCA`) (also depicted in Figure 3) is used later to describe the difference in aircraft headings between the red and blue aircraft. A specific goal zone (depicted in Figure 4) defines the *position of advantage* as the area between 0.1 and 3 m behind the red aircraft. This region was selected based on the speed and performance of the model aircraft used. A *position of advantage* reward function is defined as $g_{pa}(x)$ as shown in Algorithm 1.

A simulation was developed to model micro-UA vehicle dynamics. The dynamics are captured by the state transition function $f(x, u_b, u_r)$, see Algorithm 2, which takes both red and blue control actions (u_b and u_r) as inputs and simulates flight for a total of $0.25s$. This duration was selected based on computation time and vehicle controllability. The control actions available to both aircraft are $u\epsilon\{$ *roll-left* , *maintain-bank-angle* , *roll-right* $\}$ which is equivalently represented as $u\epsilon\{L, S, R\}$. Thus, the aircraft executes control action u for $0.25s$ ($\Delta t = 0.25s$). However, the simulation is performed as five, 0.05 second increments ($\delta t = 0.05s$). For each δt, bank angle ϕ is changed by the roll rate $\dot{\phi}$ given the action such that $\phi = \phi + u\dot{\phi}\delta t$, subject to the bank angle limits. The turn rate is updated with respect to the new bank angle: $\dot{\psi} = \frac{9.81}{v}\tan(\phi)$ ($v = 2.5\ m/s$). The heading is updated with respect

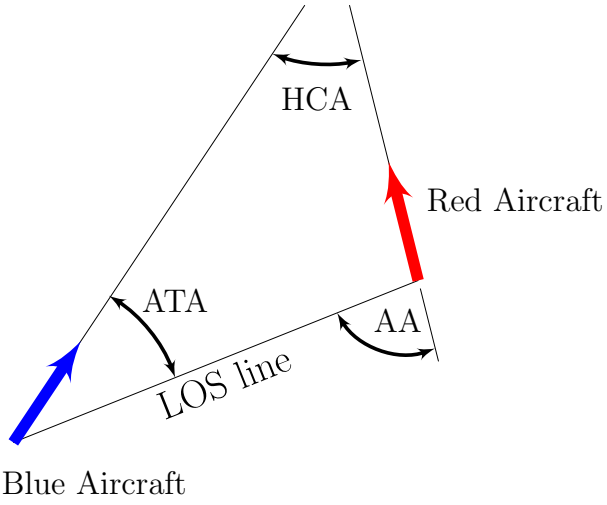


Figure 3. Aircraft relative geometry showing Aspect Angle (AA), Antenna Train Angle (ATA) and Heading Crossing Angle (HCA).

Algorithm 1 Goal Reward Function $g_{pa}(x)$

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

to the new turn rate: $\psi = \psi + \dot{\psi}\delta t$. Likewise, the position components (x^{pos} , y^{pos}) of each aircraft are updated given the new heading and fixed velocity v : $x^{pos} = x^{pos} + v\delta t \sin(\psi)$ and $y^{pos} = y^{pos} + v\Delta t \cos(\psi)$.

The red aircraft maneuvering strategy was based on Ref. [9], which was successful at producing realistic maneuvers for adversaries. This technique computes a $u_r(x)$ at each state using a limited look-ahead minimax search. The minimax search uses a scoring function ($S(x)$ from Equation 14 discussed in Section III.E) to determine the score of some future state. The specific search algorithm used is minimax with alpha–beta pruning as outlined in Ref. [22]. The recursive minimax algorithm returns the u_r that maximizes the scoring function $S(x)$ at each time-step under the assumption that the blue aircraft will select a u_b that minimizes $S(x)$. The minimax search was performed over a 0.75 s receding search horizon, thus giving the red aircraft a relatively short look ahead. Nevertheless, the algorithm manages to produce a π_r that was challenging to fly against and allowed the red aircraft to

Algorithm 2 State Transition function $f(x_t, u_b, u_r)$

Input: $\{x_t, u_b, u_r\}$, where $x_t = \{x^{pos}, y^{pos}\}$
for $1:5$ (once per $\delta t = .05s$) **do**
  for $\{red, blue\}$ **do**
    $\phi = \phi + u\dot{\phi}\delta t$, where $u\epsilon\{L, S, R\}$) and $\dot{\phi} = 40°/s$
    $\phi = \max(\phi, -\phi_{\max})$, where $\phi^{red}_{\max} = 18°$ and $\phi^{blue}_{\max} = 23°$
    $\phi = \min(\phi, \phi_{\max})$
    $\dot{\psi} = \frac{9.81}{v}\tan(\phi)$ $(v = 2.5\ m/s)$
    $\psi = \psi + \dot{\psi}\delta t$; $x^{pos} = x^{pos} + v\delta t\sin(\psi)$; $y^{pos} = y^{pos} + v\delta t\cos(\psi)$
  end for
end for
Output: $(x_{t+\Delta t})$, where $\Delta t = 5 \times \delta t$

act as a good training platform. This will be referred to as the 6-step minimax policy, because 6 decisions are made; three decisions for blue and three decisions for red at 0.25 s intervals over a 0.75 s time period. The 6-step minimax policy was selected for the red aircraft due to the fact that some assumption must be made about the adversary's expected tactics in order to generate training data. Additionally, in actual air combat, adversaries almost always exhibit some suboptimal behavior stemming from their training. The policy selected did a reasonable job of generating realistic maneuvers, but this policy could be replaced by any representation of the expected red tactics based on available information or observed behavior.

III.B. Policy Learning

The objective was 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 defined by the state transition function $f(x, u_b, u_r)$ (Algorithm 2). An adversary maneuvering policy must be assumed. In this case the 6-step minimax policy is used to produce control action u_r where $u_r = \pi_r^{nom}(x)$. Based on the maneuvering capabilities of both aircraft, a desired position of advantage has been defined in Algorithm 1.

Given the problem definition, Algorithm 3 learns approximate value function J_{approx} [b] and the associated blue maneuvering strategy π_{approx}, which is used to select the blue control action u_b given the game state x:

$$u_b = \pi_{approx}(x) \equiv \arg\max_{u_b} \left[g(f(x, u_b, \pi_r^{nom}(x))) + \gamma J_{approx}(f(x, u_b, \pi_r^{nom}(x))) \right] \text{[c]} \qquad (8)$$

The algorithm works as follows. First, a set of state space samples X are selected (see

[b] The iteration counter superscript is left out for notational simplicity.
[c] Note that the algorithm computes the current reward with respect to the resulting state.

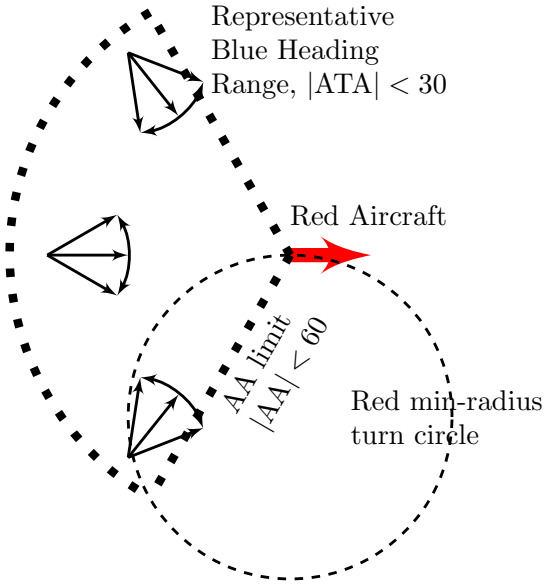


Figure 4. The blue aircraft is rewarded for maneuvering into the goal zone / *position of advantage* (shown) behind the red aircraft.

Section III.D) and the initial parameters of J_{approx} are set to approximate the scoring function (defined in Section III.E) at sample points X using least squares regression. The approximation is iteratively improved by performing a Bellman backup[4] at sample points X and updating the parameters of J_{approx} based on the new state values. Specifically, for each state sample and blue control action, we simulate forward one step to find the resulting state $x' = f(x, u_b, \pi_r^{nom})$. The action $(u_b)$is found that results in the maximum combined current and future reward, defined as $(\hat{J}_x)$. $\hat{J}_x$ for all state samples is defined as:

$$\hat{J}_X = \max_{u_b}[\gamma J_{approx}(X') + g(X')] \tag{9}$$

The parameters of J_{approx} are updated to estimate $\hat{J}_X$ via least squares regression. The estimation is performed with respect to feature vector $\Phi(X) = [\phi(x) \forall\ x \in \{X\}]$, resulting in an updated set of parameters:

$$\beta = (\Phi^T \Phi)^{-1} \Phi^T \hat{J}_X \tag{10}$$

which define

$$J_{approx}(x) \equiv \phi(x)\beta \tag{11}$$

A plot of a portion of a typical $J_{approx}(x)$ function is shown in Figure 5. Due to the complex nature of $J_{approx}(x)$ the plot is only valid for one specific set of aircraft state vari-

Algorithm 3 Combat Policy Learning

Initialize X: state space samples
Initialize $J_{approx}(x) \approx S(X)$
Initialize N: desired iterations
for $k = 1 : N$ **do**
$X' = f(X, u_b, \pi_r^{nom}(X))$
$\hat{J}_X = \max_{u_b}[\gamma J_{approx}(X') + g(X')]$
$\Phi(X) = [\phi(x) \forall\ x \in \{X\}]$
$\beta = (\Phi^T \Phi)^{-1} \Phi^T \hat{J}_X$
$J_{approx}(x) \equiv \phi(x)\beta$
end for
Output: $(J_{approx}(x))$

ables. In order to effectively learn a policy, ADP requires an approximation architecture that estimates the function well. As shown above, a linear architecture is used with respect to feature vector $\phi(x)$. The simple linear architecture pushes, to the features, a great deal the responsibility for accurately representing the value function. The extensive feature development process is discussed below.

III.C. Feature Development

As described in Section III.B, the approximation architecture used features of the state to estimate the value function. Human decision making gives some insight to the process. Pilots use on-board system information (e.g., radar and flight performance instruments) and visual cues to select maneuvers. Pilot preferences were considered when selecting information to encode as state features (Table 1). Decisions made during BFM are primarily based on relative aircraft position and orientation[d]. Typically pilots consider range between aircraft (R), aspect angle (AA), antenna train angle (ATA), aspect angle rate ($\dot{\text{AA}}$), and antenna train angle rate ($\dot{\text{ATA}}$) to be the most critical pieces of information during an engagement; these are briefly described below.

Range (R) is clearly an important tool for assessing the tactical situation. Range coupled with AA, ATA and HCA provides complete information about the current state. For reference, a graphical representation of AA is shown in Figure 6. However, the current state change rate is also relevant. $\dot{\text{AA}}$ represents the rotation rate of the red aircraft from the perspective of the blue aircraft. $\dot{\text{AA}}$ incorporates the adversary's bank angle and turn rate, range and own-ship velocity into one piece of information. $\dot{\text{AA}}$ is typically determined visually by a human pilot and is used as an initial indication of an impending aggressive maneuver by the adversary. (See Figure 7 for a graphical representation of $\dot{\text{AA}}$.) $\dot{\text{ATA}}$ is also known as

[d]The main exception is when terrain, or other obstacles, become a factor.

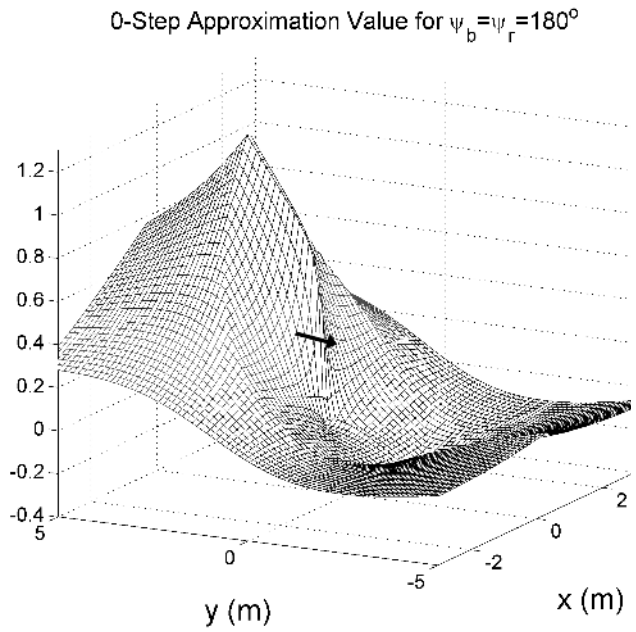


Figure 5. Function approximation from dynamic program ($J_{approx}(x)$). Function is used at each time-step by the policy extraction algorithm (Algorithm 4) to determine best control action. In this graph the heading and bank angle are fixed for both the red and blue aircraft. The arrow represents the red aircraft location and orientation. The value of the mesh represents the blue aircraft's function approximation value across the state space.

the line-of-sight rate of the red aircraft. From the perspective of the blue aircraft $\dot{\mathtt{ATA}}$ 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 adversary's velocity. $\dot{\mathtt{ATA}}$ is another piece of information which can be determined visually by a pilot and is used to make critical maneuvering decisions during close-in combat.

The features used to generate the feature vector ($\phi(x)$) were expanded via a second order polynomial expansion. 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:

$$\phi(x) = \left\{A(x), B(x), C(x), A^2(x), A(x)B(x), A(x)C(x), B^2(x), B(x)C(x), C^2(x)\right\} \qquad (12)$$

The polynomial expansion successfully produced feature vectors which generated good air combat performance. However, using a large number of features in this manner proves to be computationally expensive, making manipulation of $J_{approx}(x)$ time consuming.

The forward–backward algorithm[22] was adapted to search the available features for the smallest set that could accurately fit a $J_{approx}(x)$ function to a $\hat{J}(X)$ set. The forward-backward algorithm enables selection of a sequence of features that produce good results, without evaluating each possibility. The forward-backward search begins with an empty set of features. It searches each available feature for the one that minimizes the mean squared

Table 1. Features Considered for Function Approximation

Feature	Description	Feature	Description
x_{rel}^{pos}	Relative position on X axis	$\mathtt{ATA}^{+}$	$\max(0, \mathtt{ATA})$
y_{rel}^{pos}	Relative position on Y axis	$\mathtt{ATA}^{-}$	$\min(0, \mathtt{ATA})$
R	Euclidean distance between aircraft	$\dot{\mathtt{ATA}}$	Antenna Train Angle rate
v_c	Closure velocity	$\dot{\mathtt{ATA}}_{int}$	$10 - \left\lvert\dot{\mathtt{AA}}\right\rvert$
$\lvert\lvert v_{rel}\rvert\rvert$	Norm of Relative velocity	HCA	Heading Crossing Angle
θ_c	Closure Angle	$\lvert\mathtt{HCA}\rvert$	Abs. Value of HCA
AA	Aspect Angle	x_b^{pos}	Blue Aircraft x-position
$\lvert\mathtt{AA}\rvert$	Abs. Value of Aspect Angle	y_b^{pos}	Blue Aircraft y-position
$\mathtt{AA}^{+}$	$\max(0, \mathtt{AA})$	ϕ_b	Blue Aircraft Bank Angle
$\mathtt{AA}^{-}$	$\min(0, \mathtt{AA})$	ψ_b	Blue Aircraft Heading
$\dot{\mathtt{AA}}$	Aspect Angle rate	x_r^{pos}	Red Aircraft x-position
$\dot{\mathtt{AA}}_{int}$	$10 - \left\lvert\dot{\mathtt{AA}}\right\rvert$	y_r^{pos}	Red Aircraft y-position
ATA	Antenna Train Angle	ϕ_r	Red Aircraft Bank Angle
$\lvert\mathtt{ATA}\rvert$	Abs. Value of Antenna Train Angle	ψ_r	Red Aircraft Heading

error (MSE) of $J_{approx}(x)$ as compared to $\hat{J}$. Cross validation[23] was used to determine the average MSE of each feature. The feature that minimizes MSE the most is added to the feature vector. This 'forward' process continues until each feature has been added to the set. The 'backward' portion removes features one at a time, also selecting the feature that minimizes the MSE. The feature vector that produced the absolute minimum MSE contained 22 different features. A subset of this feature vector 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:

$$\{\lvert\mathtt{AA}\rvert,\ R,\ \mathtt{AA}^{+},\ \mathtt{ATA}^{-},\ S_A,\ S_R,\ \lvert\mathtt{HCA}\rvert,\ \dot{\mathtt{AA}}_{int},\ \dot{\mathtt{ATA}},\ \dot{\mathtt{ATA}}_{int},\ \theta_c,\ \phi_r,\ \phi_b\} \tag{13}$$

The feature vector was expanded to produce the feature vector ($\phi(x)$) which was used in the combat policy learning algorithm as described in Section III.B. The results of using this feature vector are shown in Section IV. All of the features are derived from the eight state (x) components, suggesting that there is a considerable amount of redundant information in the features. However, the selected features produced function approximations with smaller error than with simply using the components of the state alone.

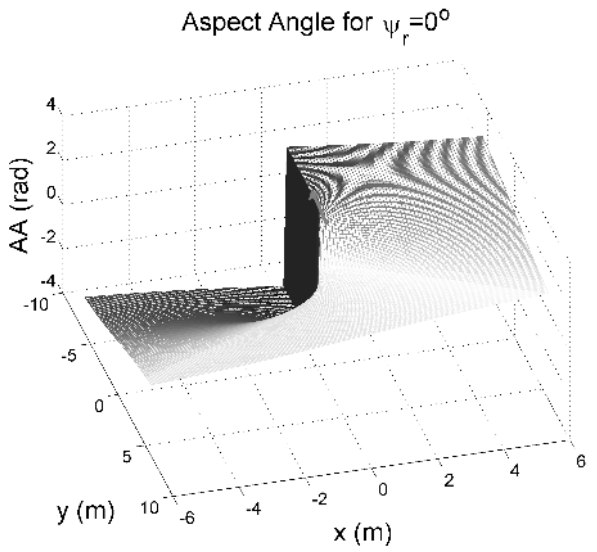


Figure 6. Plot of inter-aircraft geometry feature AA. The red aircraft is located at location (0,0) with a 0 degree heading, as indicated by the arrow. The plot represents the aspect angle for various blue aircraft relative positions. Note the aspect angle ranges from -π to π.

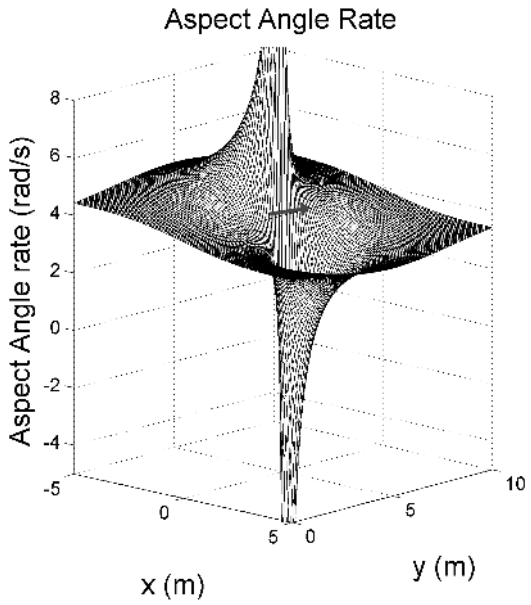


Figure 7. Plot shows the aspect angle rate ($\dot{AA}$) as perceived by the blue aircraft at various locations, given red aircraft position (shown) and red aircraft 18 degree bank angle. In certain combat situations $\dot{AA}$ is a key indication of where the blue aircraft is located with respect to the red aircraft's extended turn circle. The $\dot{AA} = 0$ rad/s corresponds to the red aircraft's current turn circle. Additionally, sudden changes in $\dot{AA}$ can indicate an impending aggressive maneuver by the adversary.

III.D. Trajectory Sampling

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. A uniform grid would be unnecessarily fine in areas of little importance and inappropriately coarse in more likely states. To ensure that the areas most likely to be seen during combat were sampled sufficiently, points were selected using trajectory sampling. 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 $\pi^*(x)$. Red and blue starting positions were selected from a Gaussian distribution with $\sigma = 7$ m. This distribution was selected based on the speed and turning performance of the aircraft. Ranges beyond

approximately 20 m result in a high aspect initial merge, so there is little benefit to exploring larger ranges. The initial aircraft headings and bank angles were selected from a uniform distribution. A combat simulation was run from this initial state using the 6-step minimax policy for each aircraft's actions, see Section III.A for further description of the simulation and policy. 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. Each state, x^n, consists of the location and orientation of both aircraft. Additionally, the red control action, $u_r(x)$, at each point is recorded. The precomputed $u_r(x)$ are subsequently used by the ADP to generate a blue policy, π_b, which counters the red maneuvers.

III.E. Reward Shaping

The goal of the blue aircraft is to attain and maintain an offensive position behind the red aircraft. The function $g_{pa}(x)$, which rewards the blue aircraft each time step it is in the goal zone, is depicted in Figure 4. By rewarding states in the goal zone, the ADP should learn a $J_{approx}(x)$ that will guide the blue aircraft toward the defined *position of advantage*. However, the discontinuous nature of $g_{pa}(x)$ made this difficult. Therefore, an alternative continuous *scoring function* S was defined. A weighted linear combination of the two functions $g_{pa}(x)$ and S were used by the ADP to generate a smoother reinforcement signal. A subset of features was explicitly chosen to generate a smooth function that provided gradient information for the case when $g_{pa}(x)$ provided only limited information about how to improve.

The scoring function is an expert developed heuristic, which reasonably captures the relative merit of every possible state in the adversarial game[8,9]. The scoring function, S, considers relative aircraft orientation and range:

$$S = \left(\frac{\left[\left(1 - \frac{\texttt{AA}}{180^\circ}\right) + \left(1 - \frac{\texttt{ATA}}{180^\circ}\right) \right]}{2} \right) \texttt{exp} \left(\frac{-\left| R - R_d \right|}{180^\circ k} \right) \tag{14}$$

Each aircraft has its own symmetric representation of the relative position of the other vehicle. Without loss of generality the following describes the geometry from the perspective of the blue aircraft. The aspect angle (`AA`) and antenna train angle (`ATA`) are defined in Figure 3. `AA` and `ATA` are limited to a maximum magnitude of 180° by definition. R and R_d are the range and desired range in meters between the aircraft, respectively. 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 and 2 m for R_d. The function returns 1.0 for a

completely offensive position ($\texttt{AA} = \texttt{ATA} = 0^\circ,\ \texttt{R} = 2$) and 0.0 for a completely defensive position ($\texttt{AA} = \texttt{ATA} = \pm 180^\circ,\ \texttt{R} = 2$).

The scoring function ($S(x)$) in Equation 14 is implemented as the red policy minimax heuristic. Due to the continuous properties of $S(x)$, it is combined with g_{pa} to create $g(x)$, used in the ADP learning algorithm.

$$g(x) = w_g g_{pa} + (1 - w_g)S \tag{15}$$

where weighting value $w_g \in [0, 1]$ was determined experimentally. The value that produced the best results was $w_g = 0.8$; this value is used for the results shown in subsequent sections.

The goal function $g(x)$ is used in Bellman backup operation (Equation 16) similar to Equation 3. The goal function $g(x_i)$ is evaluated at $x_{i+1} = f(x, u)$ for all states in set X.

$$\hat{J}^{k+1}(X) \equiv T J^k_{approx}(X) = \max_u [\gamma J^k(f(X, u)) + g(f(X, u))] \tag{16}$$

Thus, the g_{pa} reward component has influence only when the resulting system state is within the goal zone. However, the S reward component has influence over the entire state-space and tends to be higher near the goal zone. Thus, S helps to guide the ADP process in the right direction. Intuitively, one can think of S as a form of reward shaping, providing intermediate rewards, to help ADP solve sub-problems of the overall air combat problem. Alternatively, one can think of S as providing a reasonable initial value function, which is improved via ADP.

III.F. On-line Policy Extraction

By using effective feature selection, sampling and reward shaping, a good value function ($J^N_{approx}(x)$) was generated. However, $J^N_{approx}(x)$ is still not a perfect representation of the true $J^*(x)$. To minimize the effect this difference has on the resulting policy, a policy extraction method using rollout was employed.

Rollout extracts a policy from $J^N_{approx}(x)$ that more closely approximates the optimal policy $\pi^*(x)$ than $\pi^N_{approx}(x_i)$ 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*[5]. The policy resulting from rollout is referred to as $\bar{\pi}^N_{approx}(x_i)$. Algorithm 4 shows the procedure used to determine $\bar{\pi}^N_{approx}(x_i)$ on-line in both simulation and flight tests.

Rollout produced better control actions in the experiments than a one-step look-ahead Bellman backup operator. Rollout can be used to magnify the effectiveness of any given heuristic algorithm[24]. However, rollout requires more real-time computation because, as shown in Algorithm 4, the assumed red maneuvering policy must be evaluated multiple

times during rollout-based policy extraction. For example, a 3-step rollout requires the red policy to be evaluated 30 times. In generating training data to produce the blue policy, the red policy was generated by a minimax search, which is relatively time consuming to compute. In order to accomplish the policy extraction process in real-time, a faster method was required to determine the assumed red control action. The minimax search was therefore replaced during rollout with the probabilistic neural-network classifier available in the Matlab® `Neural Net Toolbox`[25]. This is similar to a method described in Ref. [26]. The neural network 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 forward-backward search architecture described in Section III.C, a forward–backward algorithm was used to search for a feature vector that produced the highest correct percentage of red policy classification. It is worth noting that by using a neural network for the red aircraft, the code is now learning both the red and blue aircraft policies. The critical difference is that the red aircraft uses a much simpler policy and reward function. In contrast, the blue aircraft uses a more complex policy which requires the use of approximate dynamic programming.

Algorithm 4 Policy Extraction, $\bar{\pi}^N_{approx}(x_i)$

Input: x_i , **Initialize:** $J_{Best} = -\infty$
for $u_b = \{L, S, R\}$ **do**
  $x_{temp} = f(x_i, u_b, \pi_r^{nom}(x_i))$
  for $j = \{1 : N_{rolls}\}$ **do**
    $x_{temp} = f(x_{temp}, \pi^N_{approx}(x_{temp}), \pi_r^{nom}(x_{temp})$
  end for
  $J_{Current} = [\gamma J^N_{approx}(x_{temp}) + g(x_{temp})]$
  if $J_{Current} > J_{Best}$ **then**
    $u_{best} = u_b$, $J_{Best} = J_{Current}$
  end if
end for
Output: u_{best}

A plot of the classifier performance during the search process is shown in Figure 8. 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 8 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 {`AA`, R, S, x_{rel}^{pos} v_{rel}}.

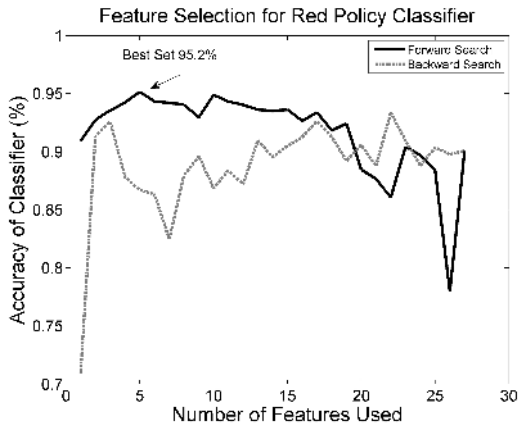


Figure 8. A neural-net learned the 6-step minimax red-policy. The plot shows generalized classification error versus the number of features, throughout the forward-backward feature search process.

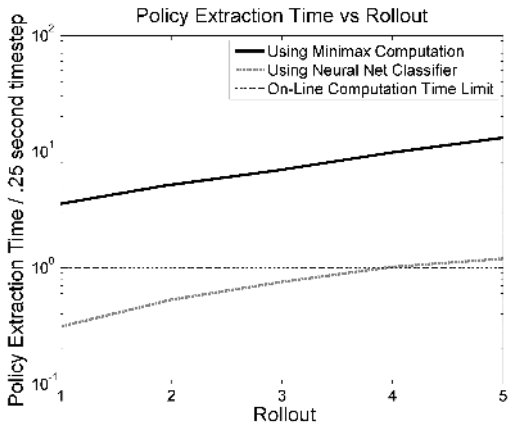


Figure 9. Plot shows the decrease in policy extraction time enjoyed via a red policy classifier; replacing the minimax search during the rollout process.

This neural net used to generate the red policy helped to increase the operating speed of the blue policy extraction algorithm by an order of magnitude. Figure 9 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 red-policy neural net classifier mimics the 6-step minimax policy and was used in the simulation and flight tests discussed in the next section.

IV. Simulation and Flight Tests

The process outlined in Section III generated successful air combat maneuvering policies. The policies were tested using a computer simulation as well as micro-UAS flight tests. Subsections IV.A and IV.B describe the simulation and test results. Subsections IV.C and IV.D describe the flight testbed and results, which demonstrate real-time air combat maneuvering on a micro-UAS aircraft.

IV.A. Combat Simulation

The policy naming convention is: π^k_{wg}, produced after k iterations, using a goal weight value of w_g. Through numerous policy learning calibration experiments, w_g=0.8 was chosen as the goal weighting value and 40 as the number of learning iterations, resulting in policy $\pi^{40}_{0.8}$. Further discussion and results from the calibration process can be found in Ref. [27].

The policy was tested in air combat using a simulation based on the state transition function described in Algorithm 2. Both aircraft are restricted to level flight, thus $A_{lat} = g\tan(\phi)$ defines the lateral acceleration for a given bank angle where $g \approx 9.81 m/s^2$.

The aircraft were initialized at the specific starting points defined in Table 2. These

Table 2. Six initial states (referred to as "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°	$-23°$	0 m	0 m	0°	18°
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°	23°	0.25 m	-0.25 m	$-45°$	0°
6	2–circle	0 m	0.1 m	270°	$-23°$	0 m	-0.1 m	90°	$-18°$

initial conditions are called "setups" in fighter pilot terms, and will be referred to as such here. The simulation accepts control actions from both aircraft, u_r and u_b, then advances the state forward by $\Delta t = 0.25$ s using $x_{t+1} = f(x_k, u_b, u_r)$ given by Algorithm 2. 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 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 for the student is a common technique used in actual BFM training to amplify the gains made by making appropriate maneuvering decisions, thus clarifying the difference between good and bad maneuvers. The performance advantage also facilitates assessment of a student's improvement from engagement to engagement. In the simulation, the intent is to assess the blue aircraft's performance using various maneuvering policies. The time required to complete the intercept (TTI) was selected as the primary measure of the effectiveness of a particular maneuvering policy. It is difficult to assess TTI 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 difference yields shorter engagements, allowing blue policy TTI comparisons.

The six initial states in Table 2 were chosen to evaluate a range of specific maneuvering tasks. The specific setups were designed to assist in easy evaluation of maneuvering performance. They put the blue aircraft in a range of positions, including offensive, neutral and defensive situations. For example, Setup #1 is an offensive setup for the blue aircraft. The blue aircraft was 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 was failing to produce reasonable decisions. Setup #3 is a defensive setup, where the blue aircraft must first maneuver defensively before becoming neutral, and then offensive, prior to achieving the goal zone.

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: 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 ($\phi_{\max}^{blue} > \phi_{\max}^{red}$). The probability of termination was used as a metric to evaluate the risk exposure (i.e., from adversary weapons). The value of p_t was computed by assigning probabilities for each time-step spent in specified weapon engagement zones (in front of the adversary).The p_t was accumulated over the course of an engagement to produce a total probability of termination for the entire engagement. A minimum amount of risk was desirable. The primary goal was to minimize TTI, a secondary goal was that of minimizing p_{t_total}.

A nominal blue aircraft maneuvering strategy (π_b^{nom}) was used as a basis for comparing the learned policy. As explained in Section III.A, the red aircraft used a minimax search with the scoring function to produce u_r. π_b^{nom} was generated using the same technique. While both aircraft had equivalent strategies, the blue aircraft consistently won the engagements due to the available performance advantage.

IV.B. Simulation Results

As an example of a maneuver in the air combat game, Figure 10 shows a simulation flown by an ADP policy. Upon initial setup, the blue aircraft was positioned behind the red aircraft, who was showing a +40 degree `AA`. At the initiation of the simulation, the red aircraft began a maximum performance right turn. The blue aircraft drove ahead then initiated a break turn which concluded with flight in the goal zone behind the red aircraft. At the termination of the break turn, the blue aircraft's flight path was aligned with the red aircraft's flight path, thus allowing continued flight in the goal zone, without a flight path overshoot. This is excellent behavior with respect to traditional BFM techniques. A more greedy behavior of immediately turning to point a the red aircraft would have resulted in a high aspect pass and a subsequent longer engagement.

Complete engagement drawings are shown in Figures 11 and 12 from selected setups 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 $\pi_{0.8}^{40}$ policy that result in considerable improvements.

The $\pi_{0.8}^{40}$ policy does better than the π_b^{nom} policy. In Figure 11(a) one can see that the red aircraft chose to reverse the turn to the left at approximately 5 s into engagement, while in

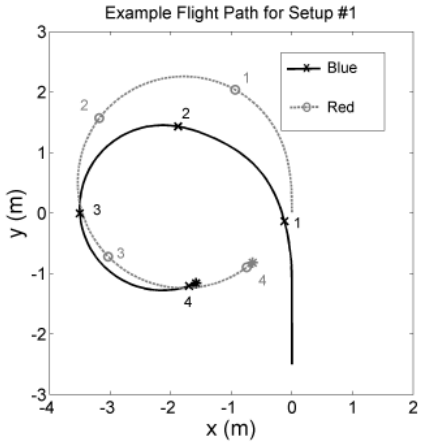
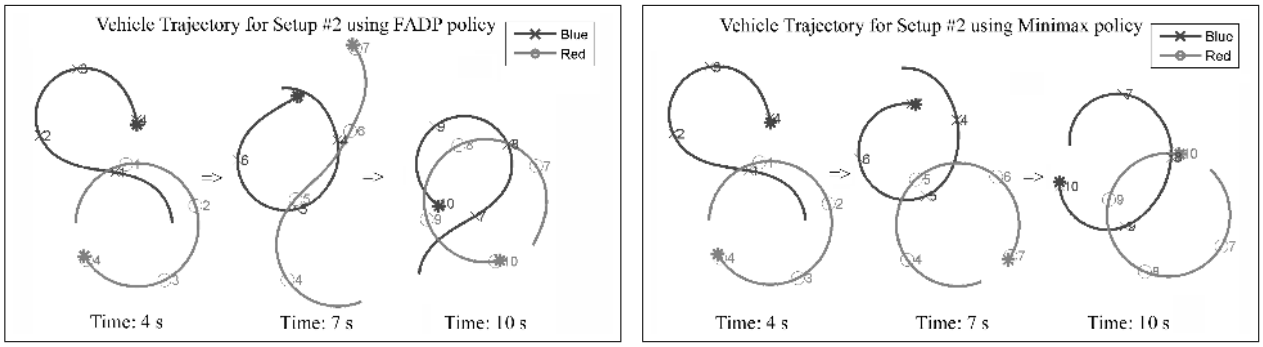


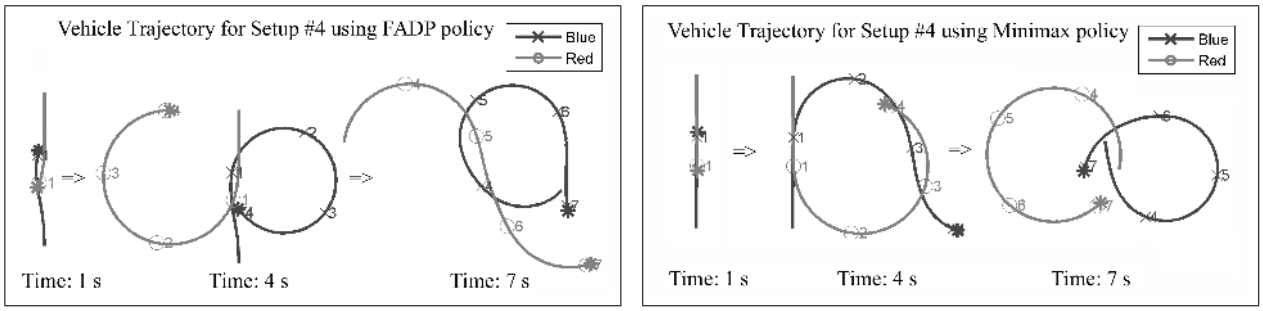
Figure 10. ADP policy simulation results demonstrating effective performance in a perch BFM setup. The numbers along each trajectory represent time in seconds.



(a) Policy $\pi_{0.8}^{40}$ (b) Policy π_b^{nom}

Figure 11. Simulation results from Setup 2 demonstrating the improvement of Policy $\pi_{0.8}^{40}$ over Policy π_b^{nom}. The red aircraft chose to reverse the turn to the left at approximately 5 s into engagement, while in Figure 11(b) the red aircraft continued to the right. There is no noticeable difference in the first frame (through 4 s), however, close inspection of the lines at 5 s shows a small difference. In the last frame (through 10 s), $\pi_{0.8}^{40}$ took advantage of the red aircraft's decision to reverse and quickly wins.

Figure 11(b) the red aircraft continued to the right. There is no noticeable difference in the first frame (through 4 s), however, close inspection of the lines at 5 s shows a small difference. In the last frame (through 10 s), $\pi_{0.8}^{40}$ took advantage of the red aircraft's decision to reverse and quickly won. Note that these simulations are deterministic, therefore any deviation on the part of red is due to some difference in the blue maneuvering. The red aircraft π_r^{nom} policy is reacting to something that $\pi_{0.8}^{40}$ did different from π_b^{nom}. In essence $\pi_{0.8}^{40}$ was capable of "faking-out" red by presenting a maneuver that appeared attractive to red, but blue was capable of exploiting in the long term. The $\pi_{0.8}^{40}$ 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. Note that in this case the red policy was generated using a neural network mimicking a minimax search, and the ADP was successful in learning a policy to exploit it. However, any technique could be used to model the adversary behavior based on available information of red maneuvering



(a) Policy $\pi_{0.8}^{40}$ (b) Policy π_b^{nom}

Figure 12. Simulation results from Setup 4 demonstrating the improvement of Policy $\pi_{0.8}^{40}$ over Policy π_b^{nom}. In the first frame (1 s) the $\pi_{0.8}^{40}$ policy made a small check turn to the left, then immediately initiated a right-hand lead-turn. This allowed the red aircraft to have a slight advantage at the initial merge while forcing a 2-circle fight which allowed blue to make the most of the turning rate advantage. The small advantage given to red is quickly regained in the following frame. At 4 s, it is clear that $\pi_{0.8}^{40}$ was extremely offensive, while the π_b^{nom} was practically neutral. In the last frame at 7 s, $\pi_{0.8}^{40}$ was seconds from winning, while π_b^{nom} still has a long way to go to complete the engagement.

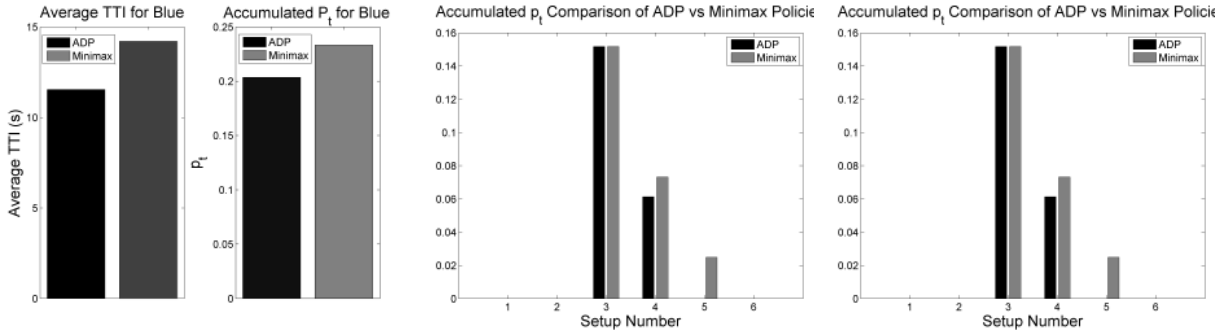
tactics.

Setup #4 in Figure 12, demonstrates learning behavior very similar to that in setup #2. In the first frame (1 s) the $\pi_{0.8}^{40}$ policy made a small check turn to the left, then immediately initiated a right-hand lead-turn. This allowed the red aircraft to have a slight advantage at the initial merge while forcing a 2-circle fight[e] which allowed blue to make the most of the turning rate advantage. The small advantage given to red is quickly regained in the following frame. At 4 s, it is clear that $\pi_{0.8}^{40}$ was extremely offensive, while the π_b^{nom} was practically neutral. In the last frame at 7 s, $\pi_{0.8}^{40}$ was seconds from winning, while π_b^{nom} still has a long way to go to complete the engagement. The ADP learning process was able to learn that a near-term suboptimal maneuver could force behavior from the red adversary which would have a large benefit in the long-term.

The performance of the $\pi_{0.8}^{40}$ policy as compared to the baseline blue policy, π_b^{nom}, is shown in Figure 13. In Figure 13(a) the average TTI per engagement and accumulated probability of termination (p_t) is shown for both the $\pi_{0.8}^{40}$ (left column in each figure) and π_b^{nom}. The $\pi_{0.8}^{40}$ policy was approximately 18.7% faster in achieving the *position of advantage* and did so with a 12.7% decrease in p_t. Figure 13(b) and 13(c) show the results of the individual setups. Setup #5 (reversal) is the one engagement where the π_b^{nom} policy managed a shorter TTI. The difference was small, approximately 1 s, and the improvements in the other setups are comparatively large. $\pi_{0.8}^{40}$ accumulated an equal or lower p_t than π_b^{nom} for all setups.

The $\pi_{0.8}^{40}$ policy was tested against policies other than the π_r^{nom} policy that it was trained against. This demonstrates the ability to maneuver successfully against an adversary that does not do what is expected, which is an important attribute of any combat system. The

[e]A 2-circle fight occurs when the aircraft are flying on separate turn circles as in Figure 12(a) at 4 s. For comparison, an example of a 1-circle fight can be seen in Figure 12(b) at 4 s.



(a) Compare overall performance. (b) TTI of each setup. (c) p_t of each setup.

Figure 13. Simulation performance of best maneuvering policy ($\pi_{0.8}^{40}$) 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 minimax baseline π_b^{nom} policy.

Table 3. Blue maneuvering policies were tested against various red policies. Blue policy $\pi_{0.8}^{40}$ was trained against a 6-step minimax red maneuvering policy (π_r^{nom}). Here the $\pi_{0.8}^{40}$ shows it is still more effective in combat than π_b^{nom} against policies other than the one it was trained on.

	Average TTI (s)					Accumulated p_t				
Policy	π_r^{nom}	π_r^{10mm}	π_r^{PP}	π_r^{R}	π_r^{L}	π_r^{nom}	π_r^{10mm}	π_r^{PP}	π_r^{R}	π_r^{L}
π_b^{nom}	14.21	29.54	16.46	15.86	15.04	0.233	0.204	0.233	0.085	0.073
π_b^{40}	11.54	25.63	13.75	12.50	9.79	0.203	0.173	0.204	0.061	0.085
% Improv.	18.7	13.2	16.5	21.3	33.1	12.7	15.2	12.7	27.6	-15.9

results appear promising. Table 3 presents the performance of $\pi_{0.8}^{40}$ and π_b^{nom} policies in combat versus five different red policies. The policies were π_r^{nom} (which was used in training), a 10-step minimax search (π_r^{10mm}), a pure-pursuit policy (π_r^{PP}), a left turning policy (π_r^{L}) and a right turning policy (π_r^{R}). For example, note the considerable additional average time required against π_r^{10mm}, as compared to π_r^{nom}. 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 $\pi_{0.8}^{40}$. The average TTI and accumulated p_t vary between the adversarial policies, but $\pi_{0.8}^{40}$ still manages to complete the intercept in less time than the minimax policy (π_b^{nom}) in each case and (in all but one case) with less risk.

IV.C. Flight Testbed

Section IV.B 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, use of the red policy classifier allowed for execution of that policy in real-time. This section completes the results by demonstrating the policy using flight tests on a real micro-UA in RAVEN.

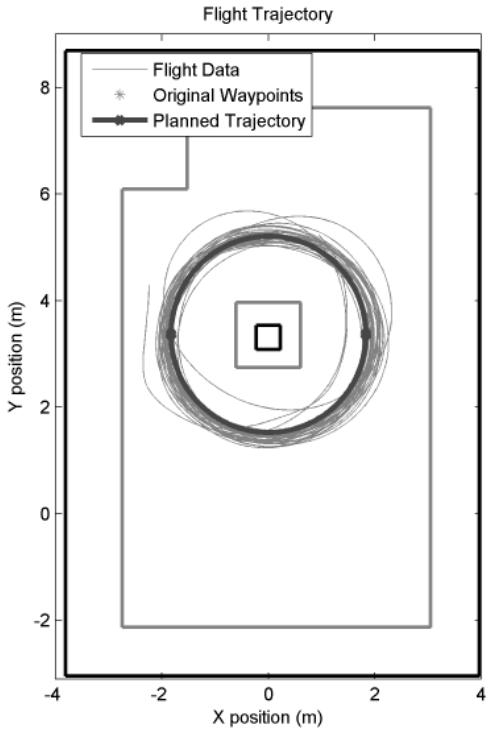


Figure 14. Flight path of micro-UA in left hand circular orbit. This stable platform was used as a target aircraft during flight tests.

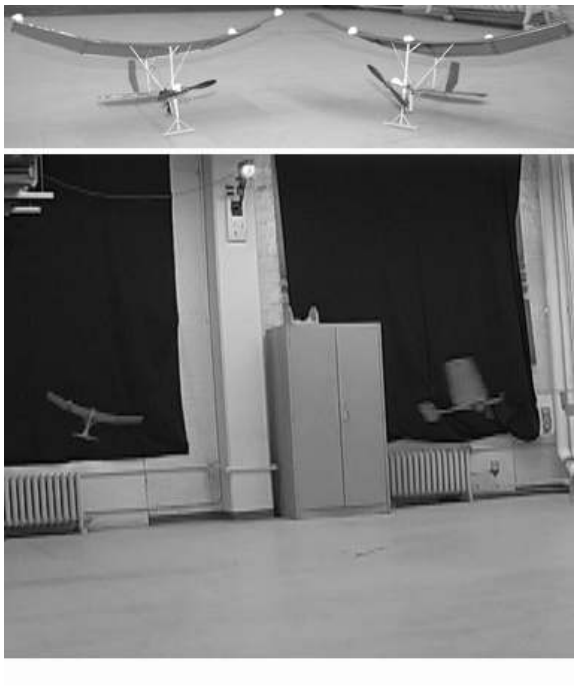
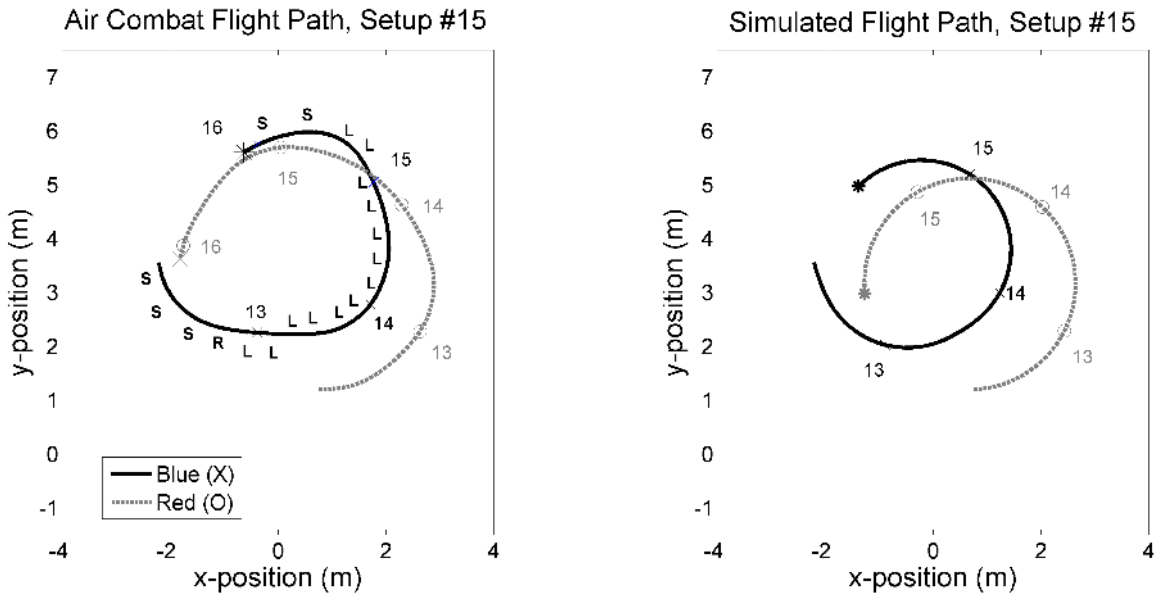


Figure 15. Above: Micro-UAS designed for Real-time indoor Autonomous Vehicle test Environment (RAVEN). Below: Micro-UAs engaged in Basic Fighter Maneuvering (BFM) during flight test.

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. Subsequently, a PID flight controller was designed and tested[27] and a trajectory follower algorithm was implemented[28,29] to achieve autonomous flight. Finally, the combat planner software was integrated into RAVEN to complete actual air combat experiments (see Refs. [3,27] for details).

For the air combat flight tests, the red aircraft was commanded to take off and fly in a continuous left hand circle, maintaining approximately $\phi_{\max} = 18^\circ$ 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 to achieve a *position of advantage* using pursuit curves and basic maneuvers such as high and low yo-yos[13]. 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 15 in MIT's RAVEN.

The $\pi^{40}_{0.8}$ policy was tested using micro-UA aircraft. The policy extraction algorithm



(a) Flight Trajectory.

(b) Simulated Trajectory.

Figure 16. 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. Lines are labeled with time in seconds since the beginning of the engagement. The blue aircraft flight line shows the maneuvering decision being executed at 0.25 s increments.

(Algorithm 4) 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 used an extended perch setup (similar to Setup #1 in Table 2). 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 demonstrated 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, which therefore requires additional lead pursuit maneuvers as well as real-world corrections.

IV.D. Flight Results

The aircraft designed to fly in RAVEN do an excellent job of following a prescribed trajectory when flown alone (see Figure 14). However, the light weight aircraft used (see Figure 15) are

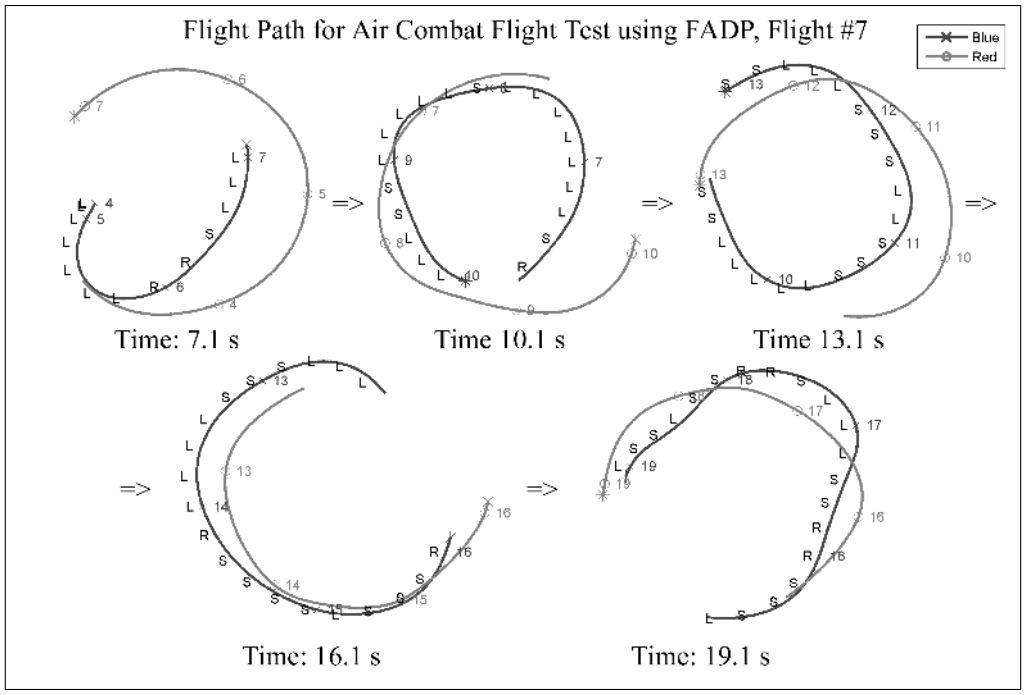


Figure 17. Test flight #7 using policy $\pi_{0.8}^{40}$ against a left turning red aircraft. The red and blue numbers along the respective flight numbers represent seconds. The black letters L, S, and R represent the current blue maneuver selection, which are left, straight, or right, respectively.

sensitive to disturbances created by other aircraft. Figure 16 demonstrates these deviations and the associated corrections. For example, in the simulated trajectory (Figure 16(b)), red makes a perfect left hand turn. Yet, in the actual flight test (Figure 16(a)) red experiences turbulence caused by blue's presence, resulting in an imperfect circle. After the disturbance, red corrects in order to track the prescribed circle, and thus sometimes exceeds the bank limit imposed in the simulation.

Figure 17 demonstrates a fight started from the extended perch setup. The blue aircraft's actions can be tracked 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 in the first frame (7.1 s). Blue eased to accommodate red's elongated turbulence induced turn in the second frame (10.1 s), then continued lead pursuit in the third frame (13.1 s). By 14 s, blue had attained the goal zone position and maintained it until a disturbance sets the aircraft off course. Blue quickly recovered and reattained the goal zone positions.

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

V. Conclusions

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. These objectives include real-time decision making (demonstrated on the RAVEN platform) using a long planning horizon (achieved via off-line ADP policy learning and on-line rollout). This flexible method is capable of producing maneuvering decisions for an aircraft which is positioned in defensive, neutral, or offensive situations in a simplified combat game in which the blue aircraft had a slight performance advantage. This capability is achieved while reducing expert human involvement. Human involvement is limited to setting high level goals and identifying air combat geometry features.

In addition to meeting the above objectives, the ADP approach achieved an overall TTI improvement of 18.7% over the minimax policy on the simplified air combat game. The simulations show intuitive examples of subtle strategy refinements, which lead to improved performance. In addition, the computed maneuvering policy performs well against an adversary that is different than the used d training. Overall, the contribution is a method which handles this simplified air-combat problem and could be extended to a more complex air-combat or other similar problem. The ADP method combined extensive feature development, trajectory sampling, and reward shaping. Furthermore, a novel (adversary policy classifier) method was developed for real-time rollout based policy extraction.

The overall process was validated on a simplified air-combat game in the horizontal plane with fixed velocity. This is a decidedly simplified setting, however, ADP is appropriate for even more complex (high-dimensional) problems that require long planning horizons. While the maneuvering policy computed and evaluated in this research was tested against a variety of adversary policies, it could not be evaluated against an equal or more capable red bandit. This is due to the limitations placed on the red aircraft when generating the sample states in Section III.D. These states were chosen to represent the full range of expected states. Thus, the approximated policy is only valid within those limits. Simulation against a more capable bandit would require a new policy to be computed. This would require generation of a new set of sample states including the higher limits and then performing the rest of the procedure described to produce a new maneuvering policy. This is specifically a limitation of the 2-D combat game used in this research. A properly formulated 3-D game could allow a full range of roll, pitch, yaw, altitude, and velocity for each aircraft. In such a formulation, a policy generated for combat against an aircraft with specific flight characteristics (e.g. maximum thrust, maximum G) should be capable of providing maneuvering solutions in combat against a variety of different aircraft and maneuvering policies. Future research could investigate the success a given policy may have against various adversaries.

The logical next step for future work would be to extend the problem to 3-D maneuvering with less restrictive vehicle dynamics (e.g. variable velocity, non-level flight). A 3-D formulation would certainly increase the number of state variables and the number of control actions. Fortunately, a good approximation architecture should be able to compactly represent the associated higher dimensional value function. Furthermore, the rollout based policy extraction operation scales linearly with the number of control actions. Consequently, the method should be extensible to this larger problem, however, careful selection of sample states and efficient programming would be required to generate the policy in a reasonable amount of time and extract the maneuvering actions in real-time.

Beyond an extension to 3-D maneuvering, it is possible that the concepts presented in this paper could be scaled to full-size, real-world combat aircraft. Certainly, the vehicle dynamics for a modern fighter aircraft is much more complex than the flight models used. Additionally, the complex flight control systems and aircraft features, such as thrust vectoring, increase the number of possible control actions. These differences lead to a larger state space and more complex maneuvering policies for both red and blue aircraft. Potentially, an Approximate Dynamic Programming formulation similar to the one presented here could be a candidate for solving such a complex problem.

Acknowledgments

Research supported in part by AFOSR # FA9550-08-1-0086 with DURIP grant # FA9550-07-1-0321 and by the American Society of Engineering Education (ASEE) through a National Defense Science and Engineering Graduate Fellowship for the lead author.

References

[1] Tiron, R., "Can UAVs Dogfight?" *Association for Unmanned Vehicle Systems International: Unmanned Systems*, Vol. 24, No. 5, Nov-Dec 2006, pp. 39–42.

[2] M. Valenti, B. Bethke, G. Fiore, J. How, and E. Feron, "Indoor multi-vehicle flight testbed for fault detection, isolation, and recovery," *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, Keystone, CO, August 2006, AIAA 2006-6200.

[3] How, J., Bethke, B., Frank A., Dale, D., and Vian, J., "Real-Time Indoor Autonomous Vehicle Test Environment," *Control Systems Magazine*, Vol. 28, No. 2, April 2008, pp. 51–64.

[4] Bellman, R., "On the Theory of Dynamic Programming," Tech. rep., Proceedings National Academia of Science, 1952.

[5] Bertsekas, D. and Tsitsiklis, J., *Neuro-Dynamic Programming*, Athena Scientific, Belmont, Massachusetts, 1996, pp. 60–70, 255-375.

[6] Isaacs, R., "Games of Pursuit," Tech. rep., The Rand Corporation, P-257, Santa Monica, CA, November 1951.

[7] Virtanen, K. and Karelahti, J. and Raivio, T., "Modeling Air Combat by a Moving Horizon Influence Diagram Game," *Journal of Guidance, Control and Dynamics*, Vol. 29, No. 5, Sep-Oct 2006, pp. 1080–1091.

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

[9] Austin, F., Carbone, G., Falco, M., and Hinz, H., "Game Theory for Automated Maneuvering During Air-to-Air Combat," *Journal of Guidance, Control and Dynamics*, Vol. 13, No. 6, Nov-Dec 1990, pp. 1143–1149.

[10] Burgin, G. and Sidor, L., "Rule-Based Air Combat Simulation," Tech. rep., NASA, CR-4160, 1988.

[11] Sprinkle, J., Eklund, J., Kim, H., and Sastry, S., "Encoding Aerial Pursuit/Evasion Games with Fixed Wing Aircraft into a Nonlinear Model Predictive Tracking Controller," *IEEE Conference on Decision and Control*, Dec. 2004, pp. 2609–2614.

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

[13] Shaw, R., *Fighter Combat Tactics and Maneuvering*, Naval Institute Press, Annapolis, Maryland, 1985, pp. 62–97.

[14] Tesauro, G., "Temporal difference learning and TD-Gammon," *Communications of the ACM*, Vol. 38, No. 3, 1995, pp. 58–68.

[15] Bertsekas, D., Tsitsiklis, J., and Wu, C., "Rollout algorithms for combinatorial optimization," *Journal of Heuristics*, Vol. 3, No. 3, 1997, pp. 245–262.

[16] Secomandi, N., "Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands," *Computers and Operations Research*, Vol. 27, No. 11-12, 2000, pp. 1201–1225.

[17] Tsitsiklis, J. and Roy, B., "Feature-based methods for large scale dynamic programming," *Machine Learning*, Vol. 22, No. 1, 1996, pp. 59–94.

[18] Powell, W., *Approximate Dynamic Programming: Solving the curses of dimensionality*, Wiley-Interscience, 2007, pp. 225–262.

[19] Bethke, B., How, J., and Ozdaglar, A., "Approximate dynamic programming using support vector regression," *Proceedings of the 2008 IEEE Conference on Decision and Control, Cancun, Mexico*, 2008, pp. 3811–3816.

[20] Bush, L., Williams, B., and Roy, N., "Computing Exploration Policies via Closed-form Least-Squares Value Iteration," *International Conference on Planning and Scheduling*, 2008, `http://groups.csail.mit.edu/mers/papers/Lawrence_Bush_ICAPS_08_ExplorationPolicies.pdf`.

[21] Keller, P., Mannor, S., and Precup, D., "Automatic basis function construction for approximate dynamic programming and reinforcement learning," *ICML '06: Proceedings of the 23rd*

international conference on Machine learning, ACM, New York, NY, USA, 2006, pp. 449–456.

[22] Russell, S. and Norvig, P., *Artificial Intelligence: A Modern Approach (2nd Edition)*, Prentice Hall, Dec. 2002, pp. 189–199, 574–575.

[23] Vapnik, V., *Statistical learning theory*, Wiley New York, 1998, pp. 524–530.

[24] Bertsekas, D. P., Tsitsiklis, J. N., and Wu, C., "Rollout Algorithms For Combinatorial Optimization," *Journal of Heuristics*, Vol. 3, 1997, pp. 245–262.

[25] The Math Works, "Neural Network Toolbox 6^{TM}," pp. 8.10–8.12, Available at http://www.mathworks.com/access/helpdesk/help/pdf_doc/nnet/nnet.pdf/ [retrieved 7 April 2010].

[26] Rodin, E. and Massoud Amin, S., "Maneuver prediction in air combat via artificial neural networks," *Computers & mathematics with applications(1987)*, Vol. 24, No. 3, 1992, pp. 95–112.

[27] McGrew, J., *Real-Time Maneuvering Decisions for Autonomous Air Combat*, S.M. thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, June 2008.

[28] Frank, A., J. McGrew, M. V., Levine, D., and How, J., "Hover, Transition, and Level Flight Control Design for a Single-Propeller Indoor Airplane," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, August 2007, AIAA-2007-6318.

[29] Park, S., Deyst, J., and How, J. P., "Performance and Lyapunov Stability of a Nonlinear Path Following Guidance Method," *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 6, Nov 2007, pp. 1718–1728.