AIAA Guidance, Navigation and Control Conference and Exhibit
18 - 21 August 2008, Honolulu, Hawaii

AIAA 2008-6796

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 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 the one-on-one air combat maneuvering problem and an approximate dynamic programming approach to computing an efficient approximation of the optimal policy. 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 which demonstrate the robustness of the method against an opponent beginning from both offensive and defensive situations. Flight results are also presented using micro-UAS flown at MITs Real-time indoor Autonomous Vehicle test ENvironment (RAVEN).

I. Introduction

Despite 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 them remotely, as was first accomplished by an MQ-1 Predator UAS in 2002[1]. However, this approach requires a one-to-one pilot-to-aircraft ratio, which does not fully leverage the strengths of combat UAS. If a UAS is ever going to fulfill the air combat missions performed by these manned aircraft we posit

*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, larry.bush@csail.mit.edu
§Professor of Aeronautics and Astronautics, williams@mit.edu.
¶Assistant Professor of Aeronautics and Astronautics, nickroy@mit.edu.

that the ability to fly BFM will be a requirement.[a] By automating some air combat decisions, an operator could potentially maximize vehicle performance while managing multiple UAs in combat.

The purpose of this research is to develop an on-line solution technique for computing near-optimal UAS BFM decisions. Computing near-optimal maneuvering decisions requires a long planning horizon. 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 complex online computations are not possible with current techniques.

These issues were addressed by applying approximate dynamic programming (ADP) to the air combat domain. On a simplified simulated air combat problem, we demonstrate a significant 18.7% improvement over the current state of the art as well as a 6.9% improvement over expert human performance. Additionally, actual micro-UAS flight results are presented using Real-time indoor Autonomous Vehicle test ENvironment (RAVEN)[2,3].

I.A. Approach Summary

The goal of air combat is to maneuver your aircraft into a position of advantage on the other aircraft, from either an offensive or defensive starting position, while minimizing risk to your own aircraft. This goal is achieved by selecting control actions (e.g., desired roll rate), given vehicle dynamics and assumed adversary strategy. Our research objective was 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 would allow switching from pursuit to evasion roles during an engagement. Dynamic programming[4] (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 solution is capable of producing good results in a finite time. The contribution of this paper is the application of approximate dynamic programming (ADP) to air combat. To accomplish this, we applied extensive feature development, trajectory sampling, reward shaping and an improved policy extraction technique using rollout. Finally, to facilitate real-time operation, we utilized a neural net classifier to model the adversary aircraft maneuvering policy.

I.B. Literature Review

Air combat has been the subject of previous research. The optimal solution to a general pursuer-evader game was first defined in [5]. 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. For example, Virtanen et al.[6] modeled air combat using an influence diagram, which could be solved using dynamic programming. However, they used a limited planning horizon to mitigate the computational complexity. Nevertheless, they demonstrated sensible control choices in real-time.

Other approaches include limited search, rule-based methods and nonlinear model predictive control. Austin et al.[7,8] demonstrated simulated real-time air combat maneuver selection using a game theoretic recursive search over a short planning horizon. The maneuver selection was again only optimal 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 in [9], which was successful in simulated air combat against human adversaries. Unfortunately, the method was time consuming to implement and improve due to hard-coded preferences from experienced

[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.

Table 1. Symbols used for ADP architecture.

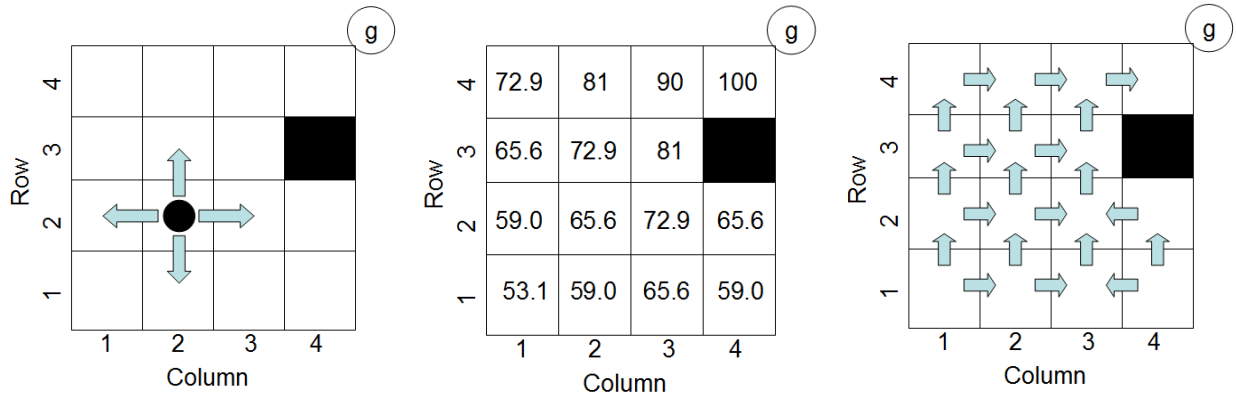
Nomenclature			
x	state vector	$\hat{J}(X)$	$[\hat{J}(x^1), \hat{J}(x^2) \dots \hat{J}(x^n)]^T$
x_i	state at time or time-step i	$J_{approx}(x)$	function approximation form of $J(x)$
x^n	n^{th} state vector in X	$\hat{J}(x)$	scalar result of Bellman backup on x
x_{term}	special terminal state	$S(x_b)$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

pilots and the manual evaluation and adjustment of maneuver selection parameters. Furthermore, the development process needed to be repeated in order to be applied to vehicles with different performance characteristics. Nevertheless, 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. Lastly, [10,11] presented a nonlinear model predictive tracking control (NMPTC) which presents a real-time implementation of an evasion controller game involving fixed wing aircraft. The authors comment on the need to encode proven aircraft maneuvering tactics from [12] into the cost functions used for the optimization in order to encourage these behaviors, as the method itself did not produce such maneuvers. The algorithm demonstrated did not have the ability to switch between pursuit and evasion roles.

While the aforementioned approaches achieved some success, we aim to improve upon them in terms of optimality, level of expert human involvement and flexibility. We achieve our objective using approximate dynamic programming.

II. Approximate Dynamic Programming Method

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, eliminating the need for extensive on-line computation. Although optimal, a DP policy is intractable to compute for large problems, because of the exponential growth of the discrete state space size with the number of state space variables. This section introduces approximate dynamic programming (ADP) using an example problem and motivates the need for an approximate solution. The section concludes with a detailed explanation of how ADP is applied to air combat.



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

II.A. Dynamic Programming Example

The shortest path DP problem shown in Figure 1 will be used to define terminology (Table 1) and methods used in future sections. The problem involves a robot 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 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 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).

The objective is to determine a movement strategy that results in the optimal 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 is accessible only from square (4,4). The reward for success defined by the function $g(x)$:

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

A function $J(x)$ is defined at each state representing the future reward value of that state. The optimal future reward function, $J^*(x)$ can be computed by repeatedly performing a Bellman backup[13] on each state. This optimal *value* will be referred to as $J^*(x)$. An optimal policy, π^* can then be computed from $J^*(x)$ using Equation 3. 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. 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 a random 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).

J^* can then be used to derive the optimal policy π^*. Where the optimal action at time-step i

is defined as:

$$\begin{aligned} u_i &= \pi^*(x_i) = u \in \{up,\ down,\ left,\ right\} \\ &= \arg\max_u \left[g(x_i) + \gamma J^*(f(x_i, u))\right] \end{aligned} \tag{3}$$

The 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[14]. A continuous function approximator eliminates the need to represent and compute the future reward for every discrete state. The advantage of using a function approximator to represent the value function is that we can represent continuous state spaces. Additionally, a function approximator requires many fewer parameters to represent the value function of a high-dimensional state space than would be required in a table lookup representation. By reducing the number of parameters, we also reduce the amount of time required to compute the optimal parameter values. The simple shortest path problem 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 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 $v\Delta t$ 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 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 $\hat{J}^{k+1}(X)$:

$$\hat{J}^{k+1}(X) = TJ^k_{approx}(X) \tag{4}$$

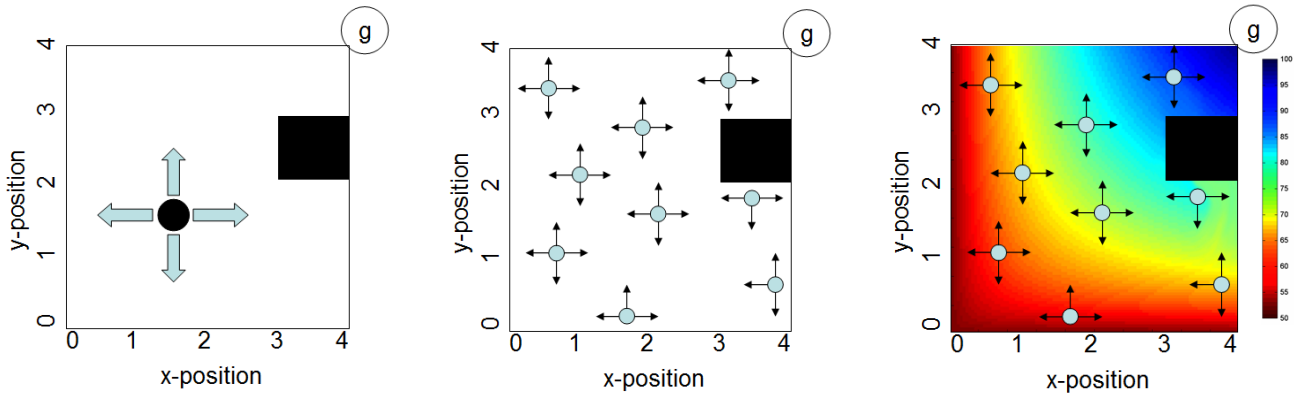
where $\hat{J}^{k+1}(X)$ refers to the set of values produced by a Bellman backup on X. We use then compute $J^{k+1}_{approx}(X)$, which is an approximation to the optimal value function using a function approximator fit to the backed-up values $\hat{J}^{k+1}(X)$. There are many choices of parametric function approximators. The technique selected was the use of least-squares to fit a hyper-plane to $\hat{J}^{k+1}(X)$, essentially using a linear function approximator. The values of any other state x can be computed as the value of the hyperplane at x.

In using function approximation to represent the value function, we are explicitly relying on the function approximator to generalize from the value of the states in X to the other states x. The assumption is that the function approximator has some knowledge of how similar x is to the basis states in X, and can perform the value interpolation appropriately. One possible measure of similarity is simply Euclidean distance between states, but the domain knowledge can be used to choose features of the states that match our intuition of how similar states are. We can represent a state x as a set of features $\phi(x)$. For a set of states X, we similarly compute a feature set Φ. The feature set is computed for all state samples $x^n \in X$ and stored in Φ so that:

$$\Phi(X) = \left[\phi(x^1)\ \phi(x^2)\ \ldots\ \phi(x^n)\right]^T \tag{5}$$

The new $J_{approx}(x)$ can now be computed using standard least squares estimation as follows[14]:

$$\beta^{k+1} = (\Phi^T\Phi)^{-1}\Phi^T\hat{J}^{k+1}(X) \tag{6}$$



(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, $\pi^*(x)$.

J_{approx} is computed as:

$$J^{k+1}_{approx}(x) \equiv \phi(x)\beta \tag{7}$$

where β are the value function parameters computed in Equation 6. 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 [14].

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 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.

This method for solving an ADP 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" [4] causes in classical DP techniques. The remainder of this section explains how approximate dynamic programming was applied to the air combat game.

III. ADP Applied to Air Combat

In this section ADP is applied to the air combat game. We first describe our system states, goal, control inputs and dynamics. We then discuss control policy learning followed by policy extraction.

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

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

$$x = \left[x_b^{pos}, y_b^{pos}, \psi_b, \phi_b, x_r^{pos}, y_r^{pos}, \psi_r, \phi_r\right]^T \tag{8}$$

The position variables 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 previously specified limits of $[-180, 180]$.

The goal of the blue aircraft is to attain and maintain a *position of advantage* behind the red aircraft. A specific goal zone (depicted in Figure 8) defines the *position of advantage* 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 1.

A simulation was developed which models micro-UA vehicle dynamics. The dynamics are captured by the state transition function $f(x, u_b, u_r)$, which takes both red and blue control actions as input 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 $u\epsilon\{L, S, R\}$. Thus, the aircraft maintains control action u_i for Δt, then executes u_{i+1} for Δt, etc. The pseudo-code in Algorithm 2 defines the operation of the state transition function.

An assumption was made regarding the red aircraft maneuvering strategy, based on [8], 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 12 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 [15]. 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 act as a good training platform. 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 intelligence.

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). Some expected adversary maneuvering policy is assumed and produces 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 our problem definition, Algorithm 3 can be used to produce value function J_{approx}^N and blue maneuvering strategy:

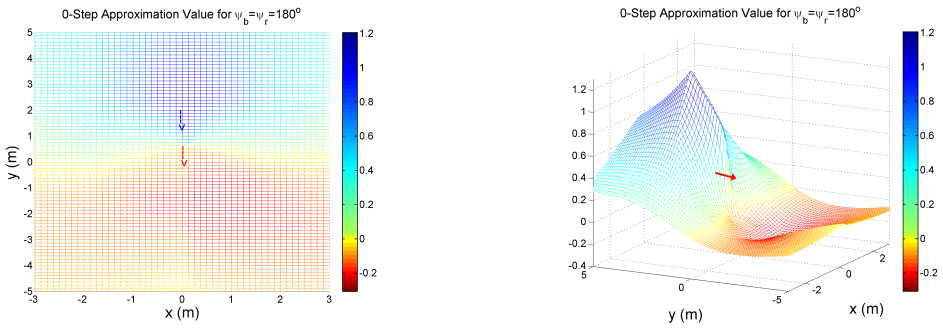
$$u_b = \pi_{approx}^N(x_i) \equiv \arg\max_{u_b} \left[g(x_i) + \gamma J_{approx}^N(f(x, u_b, \pi_r^{nom}(x_i))) \right] \tag{9}$$

to select the blue control action given any game state.

However, effective ADP requires an approximation architecture that estimates the function well. Good features are the key to a good architecture. We discuss below our extensive feature development process.

ADP iteratively approximates the value function by performing Bellman backups and regression with respect to a set of state space training samples (X). Due to the large space, state sample selection is important and challenging. This problem was addressed using trajectory sampling, discussed in Section III.A

The ADP process gradually moves toward a good value function approximation. However, the *position of advantage* reward function ($g_{pa}(x)$) is highly discontinuous. Consequently, it is difficult for the architecture to approximate intermediate value functions during the ADP process. We address this issue using reward shaping, discussed below.



(a) (b)

Figure 3. 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 red and blue heading and bank angle are fixed. The color represents the relative value (blue=offensive, red=defensive) given to blue aircraft positions surrounding the red aircraft.

Table 2. 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	$\dot{\mathtt{ATA}}_{int}$	$10 - \lvert\dot{\mathtt{AA}}\rvert$
$\lvert\lvert v_{rel} \rvert\rvert$	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
$\dot{\mathtt{AA}}_{int}$	$10 - \lvert\dot{\mathtt{AA}}\rvert$	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

III.C. Feature Development

The approximation architecture used 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 (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 2). Decisions made during BFM are primarily based on relative aircraft position and orientation[b]. Typically pilots consider R, AA, ATA, $\dot{\mathtt{AA}}$, and $\dot{\mathtt{ATA}}$ to be the most critical pieces of information during an engagement. We will briefly describe these below.

Range (R) is clearly an important tool for assessing the tactical situation. Range coupled with AA, ATA and HCA (see Figure 10) provides complete information about the current state. For reference, a graphical representation of AA is shown in Figure 4. However, the current state change rate is also relevant. $\dot{\mathtt{AA}}$ represents the rotation rate of the red aircraft from the perspective of

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

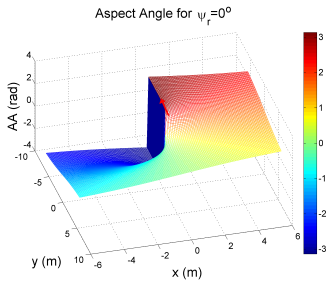


Figure 4. Plot of inter-aircraft geometry feature AA, given red aircraft indicated position and 0 degree heading, for various blue aircraft locations.

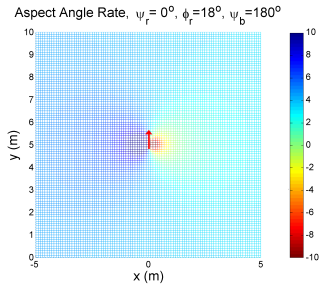


Figure 5. Plot shows $\dot{\mathtt{AA}}$ perceived by the blue aircraft at various locations, given red aircraft position (shown), 18 degree bank angle and corresponding turn rate.

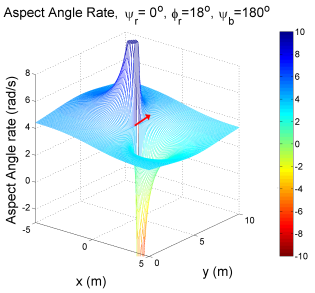


Figure 6. Rotated view of $\dot{\mathtt{AA}}$, where $\dot{\mathtt{AA}} = 0$ rad/s corresponds to the red aircraft's current turn circle.

the blue aircraft. $\dot{\mathtt{AA}}$ incorporates the adversary's bank angle and turn rate, range and own-ship velocity into one piece of information. $\dot{\mathtt{AA}}$ is typically determined visually by a human pilot and is used as an initial indication of an impending aggressive maneuver by the adversary. (See Figures 5 and 6 for a graphical representation of $\dot{\mathtt{AA}}$.) $\dot{\mathtt{ATA}}$ is also known as 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 2^{nd} 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:

$$\phi(x) = \left\{A(x), B(x), C(x), A^2(x), A(x)B(x), A(x)C(x), B^2(x), B(x)C(x), C^2(x)\right\} \tag{10}$$

The polynomial expansion successfully produced 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.

The forward–backward algorithm[15] 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 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:

$$\{|\mathtt{AA}|,\ R,\ \mathtt{AA}^{+},\ \mathtt{ATA}^{-},\ S_A,\ S_R,\ |\mathtt{HCA}|,\ \dot{\mathtt{AA}}_{int},\ \dot{\mathtt{ATA}},\ \dot{\mathtt{ATA}}_{int},\ \theta_c,\ \phi_r,\ \phi_b\} \tag{11}$$

All of the features are derived from the eight state (x) components. Consequently, 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.

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

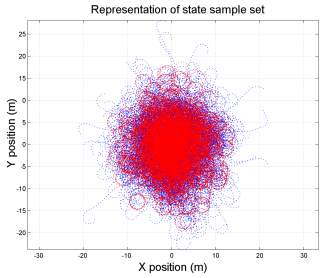


Figure 7. Set of 10^5 state space samples generated by combat simulations. The blue & red aircraft locations are shown.

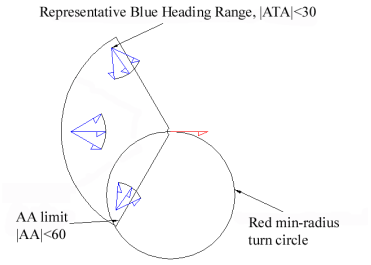


Figure 8. The blue aircraft is rewarded for maneuvering into the goal zone / *position of advantage* (shown) behind the red aircraft.

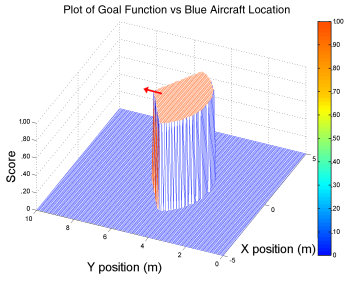


Figure 9. Plot of reward function for flight within Goal Zone (g_{pa}).

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 that the areas most likely to be seen during combat were sampled sufficiently, points were selected using trajectory sampling. 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 section III.E 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 the state samples, X, is shown in Figure 7. 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 7 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.

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 8. 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 combination of the two functions $g_{pa}(x)$ and S were used by the ADP to reinforce good behavior.

Scoring Function The scoring function is an expert developed heuristic, which reasonably captures the relative merit of every possible state in our adversarial game[7,8]. The scoring function,

S, computed as shown in Equation 12, considers relative aircraft orientation and range.

$$S = \left(\frac{\left[\left(1 - \frac{\texttt{AA}}{180^\circ}\right) + \left(1 - \frac{\texttt{ATA}}{180^\circ}\right) \right]}{2} \right) \left(e^{-\left(\frac{|R - R_d|}{180^\circ k} \right)} \right) \tag{12}$$

Each aircraft has its own symmetric representation of the relative position of the other vehicle. Without loss of generality we will describe the geometry from the perspective of the blue aircraft. The aspect angle (`AA`) and antenna train angle (`ATA`) are defined in Figure 10. `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 ($\texttt{AA} = \texttt{ATA} = \pm 180^\circ$, $\texttt{R} = 2$).

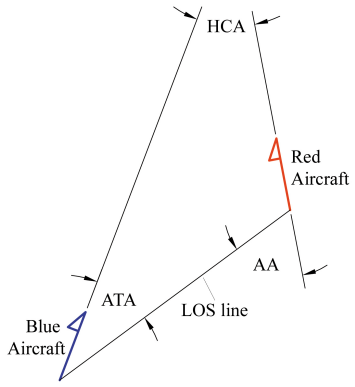


Figure 10. Aircraft relative geometry showing Aspect Angle (`AA`), Antenna Train Angle (`ATA`) and Heading Crossing Angle (`HCA`).

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

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

The scoring function ($S(x)$) defined above was originally implemented as the red policy minimax heuristic. Due to the continuous properties of $S(x)$, we combined it with g_{pa} to create $g(x)$, used in the ADP learning algorithm described in Section III.C:

$$g(x) = w_g g_{pa} + (1 - w_g) S \tag{13}$$

where weighting value $w_g \in [0, 1]$ was determined experimentally.

The goal function $g(x)$ is used in Bellman backup operation (Equation 14) similar to Equation 4. The goal function $g(x_i)$ is evaluated at $x_{i+1} = f(x, u)$ for all states in set X.

$$\hat{J}^{k+1}(X) \equiv T J^k_{approx}(X) = \max_u [\gamma J^k(f(X, u)) + g(f(X, u))] \tag{14}$$

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, we 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, we can think of S as providing a reasonable initial value function, which we improve via ADP.

Algorithm 2 State Transition function $f(x_i, u_b, u_r)$ (Air Combat Problem)

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

III.F. On-line Policy Extraction

By using effective feature selection, sampling and reward shaping, we were able to generate a good value function $J^N_{approx}(x)$. 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*[14]. 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 produces better control actions than a one-step look-ahead Bellman backup operator. However, it 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`[16]. 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 $U_r = \pi^{nom}_r(X)$ (computed using the minimax algorithm). Using the same architecture described in Section III.C, a forward–backward algorithm was used to search for a feature set that produced the highest correct percentage of red policy classification.

A plot of the classifier performance during the search process is shown in Figure 11. 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 11 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 $\{\texttt{AA},\ R,\ S,\ x^{pos}_{rel}\ v_{rel}\}$.

This neural net used to generate the red policy helped to increase the operating speed of the

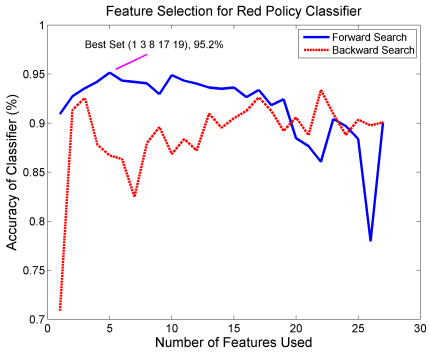


Figure 11. A neural-net learned the 6-step minimax red-policy. The plot shows generalized classification error versus the number of features, throughout the forward-backward feature search process.

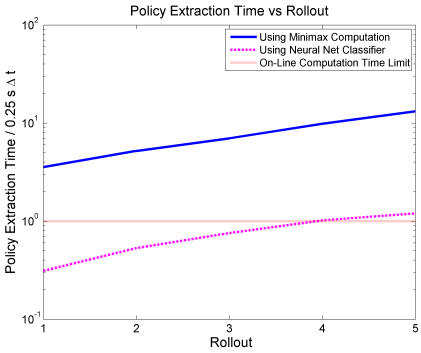


Figure 12. This plot shows the decrease in policy extraction time enjoyed via a red policy classifier; replacing the minimax search during the rollout process.

blue policy extraction algorithm by an order of magnitude. Figure 12 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.

Algorithm 3 Combat Policy Learning

Initialize $J^1_{approx}(x) \equiv S(x)$
Initialize N: desired iterations
for $k = 1 : N$ **do**
  $\overline{f} = f(X, u_b, \pi_r^{nom}(X))$
  $\hat{J}^{k+1}(X) = \max_{u_b}[\gamma J^k_{approx}(\overline{f}) + g(\overline{f})]$
  $\Phi(X) = [\phi(x) \forall\ x \in \{X\}]$
  $\beta^{k+1} = (\Phi^T\Phi)^{-1}\Phi^T\hat{J}^{k+1}(X)$
  $J^k_{approx}(x) \equiv \phi(x)\beta^{k+1}$
end for
Output: $(J^N_{approx}(x))$

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}

IV. Simulation and Flight Tests

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

IV.A. Combat Simulation

Our 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 $\pi^{40}_{0.8}$.

Table 3. 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°	$-18°$

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 $g \approx 9.81 m/s^2$.

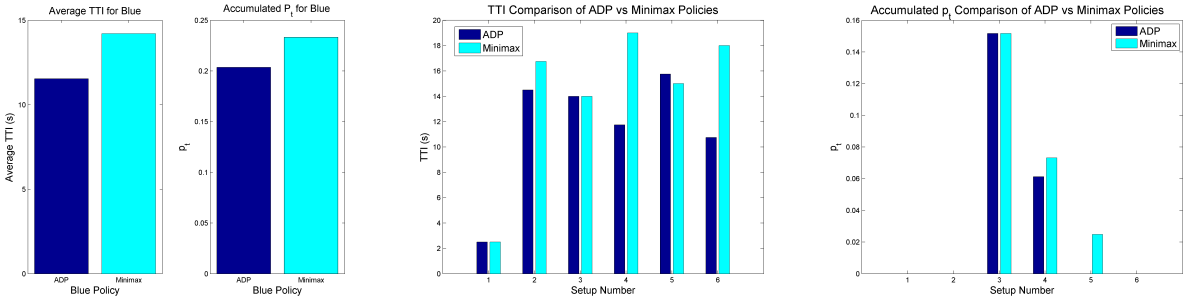
The aircraft were initialized at the specific starting points defined in Table 3. These initial conditions are called "setups" in fighter pilot terms, and will be referred to as such here. The simulation accepts a control action, u, from both aircraft, then progresses the state forward $\Delta t =$ 0.25 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 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 six initial states in Table 3 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 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.

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 our



(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.

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

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. This performance was also 6.9% better than the pilot reference result[c] in TTI and 12.5% 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.

Figure 18 shows a typical perch setup 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; this allowed continued flight in the goal zone, without a flight path overshoot. This is excellent behavior with respect to traditional BFM techniques.

Complete engagement drawings are shown 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.

During Setup #2 (Figure 14) the $\pi_{0.8}^{40}$ policy does better than the π_b^{nom} policy. In Figure 14(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 14(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

[c]The pilot reference results were produced by the lead author using manual human control of the blue aircraft in simulation.

Table 4. 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

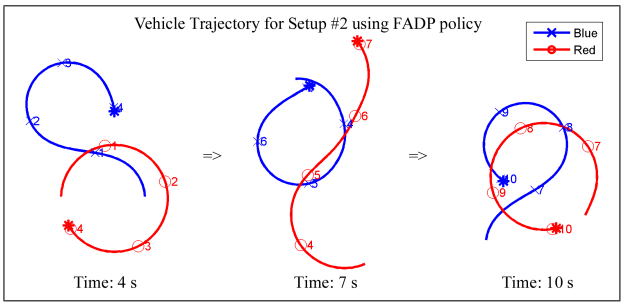
reverse and quickly wins. Note that these simulations are deterministic, therefore any deviation on the part of red is due to some difference in the blue maneuvering. Red is reacting to something different than blue did. In essence $\pi_{0.8}^{40}$ was capable of "faking-out" red by presenting a maneuver 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 mimicing 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 tactics.

Setup #4 in Figure 15, 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[d] 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 ways 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.

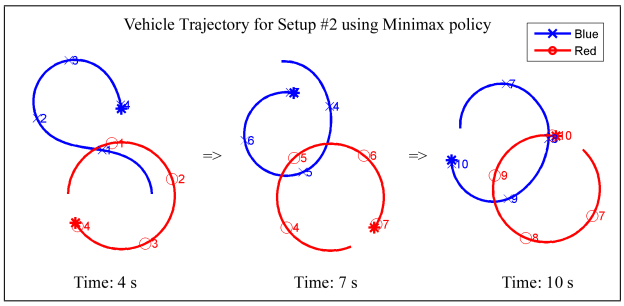
It appears that based on accepted methods of basic fighter maneuvering, $\pi_{0.8}^{40}$ continued to make good maneuver selections. Once the two different maneuvering policies deviate it is difficult to make direct comparisons. However, $\pi_{0.8}^{40}$ appears to be thinking further ahead and therefore completes the intercepts in less time and with less accumulated risk.

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 results appear promising. Table 4 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.

[d] A 2-circle fight occurs when the aircraft are flying on separate turn circles as in Figure 15(a) at 4 s. For comparison, an example of a 1-circle fight can be seen in Figure 15(b) at 4 s.

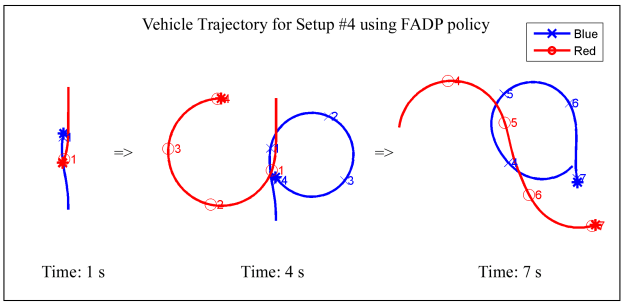


(a) Policy $\pi_{0.8}^{40}$

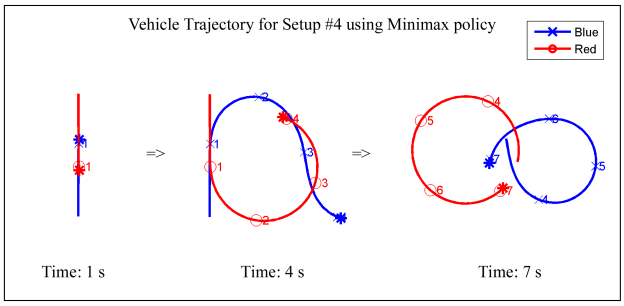


(b) Policy π_b^{nom}

Figure 14. Simulation results from Setup 2 demonstrating the improvement of Policy $\pi_{0.8}^{40}$ over Policy π_b^{nom}.



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



(b) Policy π_b^{nom}

Figure 15. Simulation results from Setup 4 demonstrating the improvement of Policy $\pi_{0.8}^{40}$ over Policy π_b^{nom}.

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, using the red policy classifier we were able to execute that policy in real-time. This section completes the results by demonstrating the policy using flight tests on a real micro-UA in RAVEN.

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, 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. For complete details on vehicle development and control see [17].

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 achieve a *position of advantage* using pursuit curves and basic maneuvers such as high and low yo-yos[12]. 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 17 in MIT's RAVEN.

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

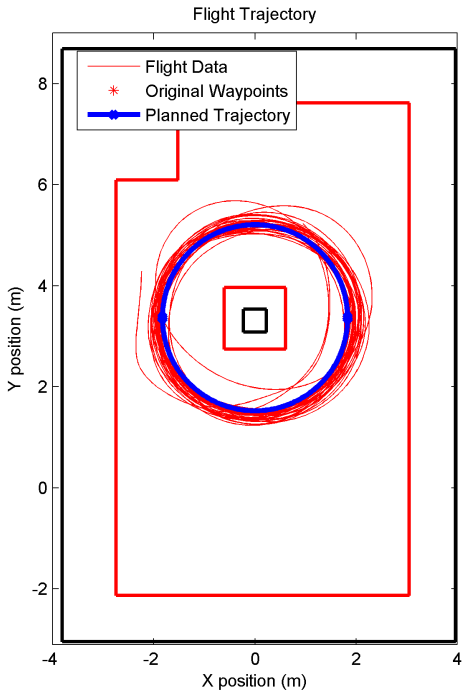


Figure 16. Flight path of micro-UA in left hand circular orbit. This stable platform was used as a target aircraft during flight tests.

Figure 17. 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.

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 3). 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, thus, requiring 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 16). However, the light weight aircraft used (see Figure 17) are sensitive to disturbances created by other aircraft. Figure 19 demonstrates these deviations and the associated corrections. For example, in the simulated trajectory (Figure 19(b)), red makes a perfect left hand turn. Yet, in the actual flight test (Figure 19(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 20 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.

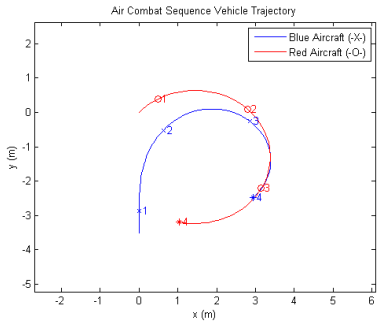
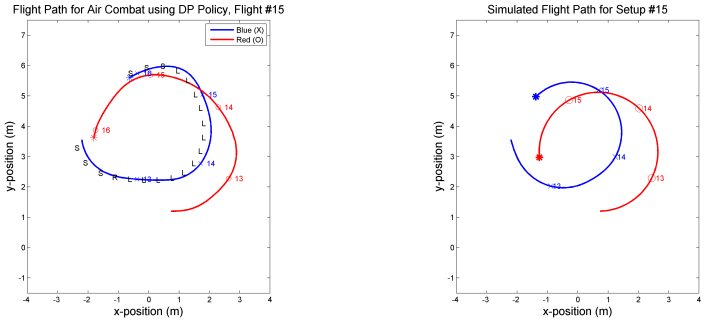


Figure 18. ADP policy simulation results demonstrating effective performance in a perch BFM setup. The numbers along each trajectory represent time in seconds.



(a) Flight Trajectory. (b) Simulated Trajectory.

Figure 19. 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.

In the first flight, blue aggressively lead pursuit in the first frame (7.1 s). Blue then 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 our RAVEN platform) using a long planning horizon (achieved via off-line ADP policy learning and on-line rollout). Our flexible method is capable of switching roles from defender to offender during an engagement. We achieved the above goals while reducing expert human involvement to the setting of high level goals and identifying features of air combat geometry.

In addition to meeting the above objectives, our ADP approach achieved an overall TTI improvement of 18.7%. Our simulations show intuitive examples of subtle strategy refinements, which lead to improved performance. Overall, we have contributed a method which handles a complex air-combat problem. Our ADP method combined extensive feature development, trajectory sampling, and reward shaping. Furthermore, we developed a novel (adversary policy classifier) method for real-time rollout based policy extraction.

We restricted our work to air-combat in the horizontal plane with fixed velocity. However, ADP is appropriate for even more complex (high-dimensional) problems, which require long planning horizons. In summary, future work should focus on extending our approach to 3-D problems with less restrictive vehicle dynamics.

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.

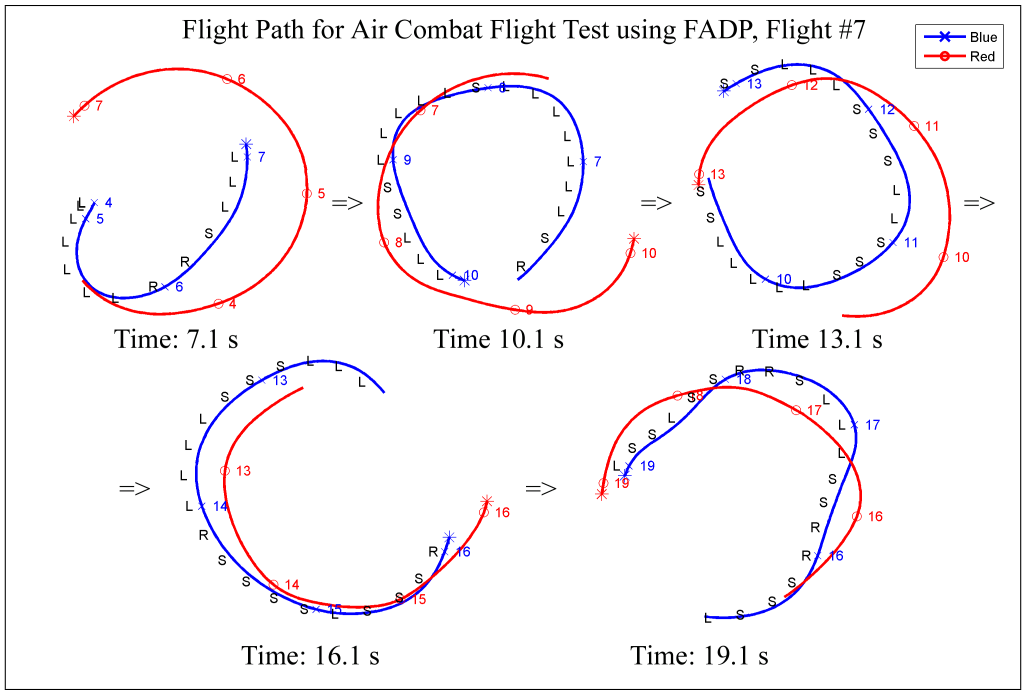


Figure 20. Test flight #7 using policy $\pi_{0.8}^{40}$ against a left turning red aircraft. The red and blue numbers along the respective flight numbers represent seconds. The black letters L, S, and R represent the current blue maneuver selection, which are left, straight, or right, respectively.

References

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

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

[3] J. How, B. Bethke, A. Frank, D. Dale, and J. Vian, "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., Proc. Nat. Acad. Sci., 1952.

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

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

[7] 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.

[8] 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.

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

[10] 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 Conf. on Decis. and Control*, Dec. 2004.

[11] 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 2005 American Control Conference*, Vol. 3, June 2005, pp. 1509–1514.

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

[13] 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.

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

[15] Russell, S. and Norvig, P., *Artificial Intelligence: A Modern Approach (2nd Edition)*, Prentice Hall, Dec. 2002.

[16] The Math Works, "Neural Network Toolbox *newpnn* article," http://www.mathworks.com, 2008.

[17] 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.