Development of an Integrated Intelligent Multi-Objective Framework for UAV Trajectory Generation

Jennifer Nicole Wilburn
Fault-Tolerant Trajectory Tracking of Unmanned Aerial Vehicles Using Immunity-Based Model Reference Adaptive Control

by
Brenton K. Wilburn

Dissertation submitted to the Benjamin M. Statler College of Engineering and Mineral Resources at West Virginia University In partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Mechanical Engineering

Larry Banta, Ph.D.
Mridul Gautam, Ph.D.
Wade Huebsch, Ph.D.
Powsiri Klinkhachorn, Ph.D.
Victor Mucino, Ph.D.
Mario Perhinschi, Ph.D., Chair

Department of Mechanical and Aerospace Engineering Morgantown, West Virginia December 5, 2014
FAULT-TOLERANT TRAJECTORY TRACKING OF UNMANNED AERIAL VEHICLES USING IMMUNITY-BASED MODEL REFERENCE ADAPTIVE CONTROL

Brenton K. Wilburn

This dissertation presents the design, development, and simulation testing of an adaptive trajectory tracking algorithm capable of compensating for various aircraft subsystem failures and upset conditions. A comprehensive adaptive control framework, here within referred to as the immune model reference adaptive control (IMRAC) algorithm, is developed by synergistically merging core concepts from the biologically-inspired artificial immune system (AIS) paradigm with more traditional optimal and adaptive control techniques. In particular, a model reference adaptive control (MRAC) algorithm is enhanced with the detection and learning capabilities of a novel, artificial neural network augmented AIS scheme. With the given modifications, the MRAC scheme is capable of detecting and identifying a given failure or upset condition, learning how to adapt to the problem, responding in a manner specific to the given failure condition, and retaining the learning parameters for quicker adaptation to subsequent failures of the same nature.

The IMRAC algorithm developed in this dissertation is applicable to a wide range of control problems. However, the proposed methodology is demonstrated in simulation for an unmanned aerial vehicle. The results presented show that the IMRAC algorithm is an effective and valuable extension to traditional optimal and adaptive control techniques. The implementation of this methodology can potentially have significant impacts on the operational safety of many complex systems.
DEDICATION

To my wife, Jennifer, thank you for always being there for me through all of the
tough times.
ACKNOWLEDGEMENTS

I would like to thank all of the sponsors which have made this research possible. This research was initially supported by a grant from the Army Research Laboratory. Later it was supported by the NASA West Virginia Space Grant Consortium Graduate Fellowship Program.
# TABLE OF CONTENTS

ABSTRACT ............................................................................................................. II

DEDICATION ........................................................................................................... III

ACKNOWLEDGEMENTS ......................................................................................... IV

LIST OF FIGURES .................................................................................................. VIII

LIST OF TABLES ...................................................................................................... XII

NOMENCLATURE ...................................................................................................... XIII

1 INTRODUCTION .................................................................................................. 1

1.1 Motivation ......................................................................................................... 1

1.2 Research Objectives ......................................................................................... 5

1.3 Thesis Overview ............................................................................................... 7

2 LITERATURE REVIEW ....................................................................................... 8

2.1 Trajectory Tracking ......................................................................................... 8

2.2 Adaptive Control .............................................................................................. 9

2.2.1 Gain Scheduling ......................................................................................... 11

2.2.2 Model Reference Adaptive Control .............................................................. 13

2.2.3 Self-Tuning Regulators ............................................................................ 15

2.2.4 Dual Control ............................................................................................. 16

2.2.5 \( \mathcal{L}_1 \) Adaptive Control ........................................................................ 17

2.2.6 Neural Networks ...................................................................................... 19

2.3 Genetic Algorithm ............................................................................................ 20

2.4 Artificial Immune System .............................................................................. 21

3 Artificial Immune System .................................................................................. 23

3.1 Biological Immune System ............................................................................. 23
3.2 Artificial Immune Feedback Suppression Mechanism ........................................... 26
3.3 Artificial Immune System Negative Selection Strategy .................................. 28

4 TESTING ENVIRONMENT .................................................................................. 31

4.1 WVU UAV Simulation Environment ............................................................... 31
  4.1.1 Architecture ............................................................................................... 31
  4.1.2 Aircraft Models .......................................................................................... 36
  4.1.3 Failure Models ........................................................................................... 38

4.2 Performance Indices ....................................................................................... 42

4.3 Batch Simulations ........................................................................................... 46

5 TRAJECTORY TRACKING CONTROLLERS ...................................................... 50

5.1 Fixed-Parameter Controllers .......................................................................... 50
  5.1.1 Position PID ............................................................................................... 50
  5.1.2 Outer-Loop NLDI ....................................................................................... 52
  5.1.3 Extended NLDI ......................................................................................... 53
  5.1.4 LQR with Integral Action ......................................................................... 55

5.2 Adaptive Control ............................................................................................. 60
  5.2.1 AIS Suppression Mechanism .............................................................. 61
  5.2.2 ℋ1 .............................................................................................................. 62
  5.2.3 MRAC ....................................................................................................... 63
    5.2.3.1 Dead-Zone Modification ................................................................. 71
    5.2.3.2 Projection Operator ............................................................................ 73
    5.2.3.3 Reference Model Modification ....................................................... 77
    5.2.3.4 Application to UAV Trajectory Tracking ........................................ 77
  5.2.4 IMRAC ..................................................................................................... 78
    5.2.4.1 Artificial Immune System Detection Scheme ................................. 80
    5.2.4.2 Generation of Detectors ................................................................. 86
    5.2.4.3 Artificial Neural Network State Predictor ...................................... 88
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Performance versus robustness trade-off</td>
<td>10</td>
</tr>
<tr>
<td>2-2</td>
<td>Overview of gain scheduling</td>
<td>12</td>
</tr>
<tr>
<td>2-3</td>
<td>Overview schematic of model reference adaptive control</td>
<td>14</td>
</tr>
<tr>
<td>2-4</td>
<td>Self-tuning regulator overview</td>
<td>15</td>
</tr>
<tr>
<td>3-1</td>
<td>Overview of the biological immune feedback response</td>
<td>26</td>
</tr>
<tr>
<td>3-2</td>
<td>Nonlinear suppression function</td>
<td>27</td>
</tr>
<tr>
<td>3-3</td>
<td>Biological immune system intruder detection</td>
<td>29</td>
</tr>
<tr>
<td>3-4</td>
<td>Example structured identification detector set</td>
<td>30</td>
</tr>
<tr>
<td>4-1</td>
<td>WVU UAV Simulation Environment architecture</td>
<td>32</td>
</tr>
<tr>
<td>4-2</td>
<td>WVU UAV Simulation Environment version 3.0 Simulink model</td>
<td>33</td>
</tr>
<tr>
<td>4-3</td>
<td>FlightGear visualization software displaying the YF-22 aircraft model</td>
<td>34</td>
</tr>
<tr>
<td>4-4</td>
<td>WVU UAVDashboard software package</td>
<td>35</td>
</tr>
<tr>
<td>4-5</td>
<td>WVU YF-22 research aircraft</td>
<td>36</td>
</tr>
<tr>
<td>4-6</td>
<td>Pioneer FlightGear visualization</td>
<td>38</td>
</tr>
<tr>
<td>4-7</td>
<td>TigerShark FlightGear visualization</td>
<td>38</td>
</tr>
<tr>
<td>4-8</td>
<td>Control surface failure configuration GUI</td>
<td>41</td>
</tr>
<tr>
<td>4-9</td>
<td>Control surface actuator/damage model</td>
<td>41</td>
</tr>
<tr>
<td>4-10</td>
<td>Collect batch results GUI</td>
<td>48</td>
</tr>
<tr>
<td>4-11</td>
<td>Performance indices display</td>
<td>49</td>
</tr>
<tr>
<td>5-1</td>
<td>Trajectory tracking formation geometry</td>
<td>51</td>
</tr>
<tr>
<td>5-2</td>
<td>Overview of the position PID architecture</td>
<td>52</td>
</tr>
<tr>
<td>5-3</td>
<td>Extended NLDI control architecture</td>
<td>54</td>
</tr>
<tr>
<td>5-4</td>
<td>State-feedback architecture</td>
<td>56</td>
</tr>
<tr>
<td>5-5</td>
<td>State-feedback with integral action</td>
<td>57</td>
</tr>
<tr>
<td>5-6</td>
<td>MRAC architecture</td>
<td>69</td>
</tr>
</tbody>
</table>
Figure 5-7. MRAC tracking performance ............................................................................. 70
Figure 5-8. MRAC parameter estimates ............................................................................. 71
Figure 5-9. μ operator ........................................................................................................ 73
Figure 5-10. Projection operator within transition region .................................................... 74
Figure 5-11. Projection operator at parameter bounds ........................................................ 74
Figure 5-12. MRAC tracking performance with projection bounds ..................................... 76
Figure 5-13. MRAC parameter estimates with projection bounds ...................................... 76
Figure 5-14. Overview of the AIS MRAC mechanism ......................................................... 80
Figure 5-15. IMRAC preparation flowchart ....................................................................... 84
Figure 5-16. IMRAC operation flowchart ......................................................................... 85
Figure 5-17. Flowchart of detector generation .................................................................. 88
Figure 5-18. Artificial neuron structure .............................................................................. 89
Figure 5-19. Back propagation ANN .................................................................................. 90
Figure 5-20. Neural network angular rate estimation for figure-8 trajectory ...................... 94
Figure 5-21. Neural network angular rate estimation error ............................................... 95
Figure 6-1. Overview of the evolution process .................................................................. 99
Figure 6-2. Generating initial population .......................................................................... 101
Figure 6-3. Mutation operation ......................................................................................... 103
Figure 6-4. Mutation flowchart ........................................................................................ 103
Figure 6-5. Crossover operation ......................................................................................... 104
Figure 6-6. Crossover flowchart ......................................................................................... 105
Figure 6-7. Performance index normalization .................................................................... 108
Figure 6-8. Selection flowchart ........................................................................................ 109
Figure 6-9. Optimization results for the position PID controller ....................................... 110
Figure 6-10. Optimization results for the outer-loop NLDI controller ............................... 110
Figure 6-11. Optimization results for the extended NLDI controller ................................ 110
Figure 6-12. Optimization results for the adaptive position PID controller ...................... 110
Figure 6-13. GA comparison of extra normalization step for the position PID controller
112

Figure 6-14. GA comparison of extra normalization step for the outer-loop NLDI controller ................................................................. 112

Figure 6-15. GA comparison of extra normalization step for the extended NLDI controller .................................................................................. 112

Figure 6-16. GA comparison of extra normalization step for the adaptive position PID controller .................................................................................. 112

Figure 6-17. Tracking error for a non-optimized position PID controller for 3D s-turns trajectory .......................................................................................................................................................................................... 113

Figure 6-18. Tracking error for a GA optimized position PID controller for 3D s-turns trajectory .......................................................................................................................................................................................... 113

Figure 6-19. Robust optimization results for the LQR with integral action controller .. 114

Figure 7-1. Figure-8 ........................................................................................................ 116

Figure 7-2. Oval .................................................................................................................. 116

Figure 7-3. S-turns – horizontal projection........................................................................ 116

Figure 7-4. S-turns – vertical projection ............................................................................ 116

Figure 7-5. Obstacle avoidance ........................................................................................ 117

Figure 7-6. Average global performance index comparison for different trajectories... 120

Figure 7-7. Average global performance index for different failures.......................... 120

Figure 7-8. Average performance index by path for position PID versus AIS-augmented position PID .......................................................................................................................... 121

Figure 7-9. Average total performance for L1 adaptive control.................................... 123

Figure 7-10. Average total performance for MRAC derived controllers ....................... 125

Figure 7-11. Parametric uncertainty estimate for RMRAC with high adaptation rate for an elevator failure .................................................................................................................. 127
Figure 7-12. Parametric uncertainty estimate for RMRAC with low adaptation rate for an elevator failure ................................................................. 127
Figure 7-13. Parametric uncertainty estimate for IMRAC with low adaptation rate for an elevator failure ................................................................. 130
Figure 7-14. LQR with integrator subjected to 16° right stabilator failure ............... 133
Figure 7-15. “Vaccinated” IMRAC algorithm subjected to 16° right stabilator failure . 133
Figure 7-16. Total performance by path for IMRAC .................................................. 134
LIST OF TABLES

Table 6-1. Optimization results for coupled outer/inner-loop tuning ................................ 110
Table 7-1. Description of upset condition severities ................................................................. 118
Table 7-2. Performance index weights and normalization cut-offs ........................................ 118
Table 7-3. Nominal PI for MRAC controllers ........................................................................ 126
Table 7-4. IMRAC detection rate ............................................................................................ 129
Table 7-5. Convergence of the IMRAC algorithm for a control matrix uncertainty .......... 131
Table 7-6. Convergence of the IMRAC algorithm for a stuck rudder failure ....................... 131
Table 7-7. Convergence of IMRAC algorithm for uncertain moments of inertia .............. 132
Table 7-8. Example of IMRAC vaccination ............................................................................. 133
# NOMENCLATURE

## VARIABLES

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>English</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A$</td>
<td>State matrix</td>
<td>-</td>
</tr>
<tr>
<td>$B$</td>
<td>Control matrix</td>
<td>-</td>
</tr>
<tr>
<td>$C$</td>
<td>Output matrix</td>
<td>-</td>
</tr>
<tr>
<td>$C$</td>
<td>Cluster</td>
<td>-</td>
</tr>
<tr>
<td>$D$</td>
<td>Feedthrough matrix</td>
<td>-</td>
</tr>
<tr>
<td>$D$</td>
<td>Detector</td>
<td>-</td>
</tr>
<tr>
<td>$e$</td>
<td>Error</td>
<td>-</td>
</tr>
<tr>
<td>$f$</td>
<td>Forward error</td>
<td>m</td>
</tr>
<tr>
<td>$h$</td>
<td>Horizontal error</td>
<td>m</td>
</tr>
<tr>
<td>$K$</td>
<td>Gain</td>
<td>-</td>
</tr>
<tr>
<td>$l$</td>
<td>Chromosome length</td>
<td>-</td>
</tr>
<tr>
<td>$l$</td>
<td>Lateral error</td>
<td>m</td>
</tr>
<tr>
<td>$n$</td>
<td>Population size</td>
<td>-</td>
</tr>
<tr>
<td>$p$</td>
<td>Roll rate</td>
<td>rad/s</td>
</tr>
<tr>
<td>$P$</td>
<td>Population</td>
<td>-</td>
</tr>
<tr>
<td>$P$</td>
<td>Ricatti equation solution</td>
<td>-</td>
</tr>
<tr>
<td>$q$</td>
<td>Pitch rate</td>
<td>rad/s</td>
</tr>
<tr>
<td>$Q$</td>
<td>LQR performance parameter</td>
<td>-</td>
</tr>
<tr>
<td>$r$</td>
<td>Yaw rate</td>
<td>rad/s</td>
</tr>
<tr>
<td>$R$</td>
<td>LQR control effort parameter</td>
<td>-</td>
</tr>
<tr>
<td>$S$</td>
<td>Self</td>
<td>-</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td>Unit</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>$T$</td>
<td>Time</td>
<td>sec</td>
</tr>
<tr>
<td>$u$</td>
<td>Control input</td>
<td>-</td>
</tr>
<tr>
<td>$U$</td>
<td>Universe</td>
<td>-</td>
</tr>
<tr>
<td>$V$</td>
<td>Velocity</td>
<td>m/s</td>
</tr>
<tr>
<td>$w$</td>
<td>Weight</td>
<td>-</td>
</tr>
<tr>
<td>$x$</td>
<td>System state</td>
<td>-</td>
</tr>
<tr>
<td>$X$</td>
<td>Feature state</td>
<td>-</td>
</tr>
<tr>
<td>$y$</td>
<td>System output</td>
<td>-</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Angle of attack</td>
<td>rad</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Sideslip angle</td>
<td>rad</td>
</tr>
<tr>
<td>$\Gamma$</td>
<td>Modification decision matrix</td>
<td>-</td>
</tr>
<tr>
<td>$\delta$</td>
<td>Deflection of control surface</td>
<td>rad</td>
</tr>
<tr>
<td>$\eta$</td>
<td>AIS effectiveness scaling factor</td>
<td>-</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Pitch angle</td>
<td>rad</td>
</tr>
<tr>
<td>$\Theta$</td>
<td>Parametric uncertainty vector</td>
<td>-</td>
</tr>
<tr>
<td>$K$</td>
<td>Potential modification matrix</td>
<td>-</td>
</tr>
<tr>
<td>$\Lambda$</td>
<td>Random decision matrix</td>
<td>-</td>
</tr>
<tr>
<td>$\Lambda$</td>
<td>Uncertainty matrix</td>
<td>-</td>
</tr>
<tr>
<td>$\mu$</td>
<td>Mutation rate</td>
<td>-</td>
</tr>
<tr>
<td>$\mu$</td>
<td>Dead-zone operator</td>
<td>-</td>
</tr>
<tr>
<td>$\nu$</td>
<td>Crossover rate</td>
<td>-</td>
</tr>
<tr>
<td>$\bar{\sigma}$</td>
<td>Chromosomal modification limit</td>
<td>-</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>Chromosomal modification limit</td>
<td>-</td>
</tr>
</tbody>
</table>

Greek symbols:
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi )</td>
<td>Bank angle</td>
<td>rad</td>
</tr>
<tr>
<td>( \Phi )</td>
<td>MRAC regressor vector</td>
<td>-</td>
</tr>
<tr>
<td>( \psi )</td>
<td>Heading angle</td>
<td>rad</td>
</tr>
</tbody>
</table>

**Subscripts**

<table>
<thead>
<tr>
<th>Subscript</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>Aileron</td>
<td>-</td>
</tr>
<tr>
<td>( ad )</td>
<td>Adaptive</td>
<td>-</td>
</tr>
<tr>
<td>( B )</td>
<td>B-cells</td>
<td>-</td>
</tr>
<tr>
<td>( bl )</td>
<td>Baseline</td>
<td>-</td>
</tr>
<tr>
<td>( d )</td>
<td>Desired</td>
<td>-</td>
</tr>
<tr>
<td>( e )</td>
<td>Elevator</td>
<td>-</td>
</tr>
<tr>
<td>( h )</td>
<td>Helper</td>
<td>-</td>
</tr>
<tr>
<td>( i )</td>
<td>Individual in population</td>
<td>-</td>
</tr>
<tr>
<td>( j )</td>
<td>Generation number</td>
<td>-</td>
</tr>
<tr>
<td>( r )</td>
<td>Rudder</td>
<td>-</td>
</tr>
<tr>
<td>( ref )</td>
<td>Reference</td>
<td>-</td>
</tr>
<tr>
<td>( s )</td>
<td>Suppressor</td>
<td>-</td>
</tr>
<tr>
<td>( t )</td>
<td>Thrust</td>
<td>-</td>
</tr>
<tr>
<td>( T )</td>
<td>T-cells</td>
<td>-</td>
</tr>
<tr>
<td>( XY )</td>
<td>Horizontal plane</td>
<td>-</td>
</tr>
<tr>
<td>( XYZ )</td>
<td>Combined coordinate system</td>
<td>-</td>
</tr>
<tr>
<td>( Z )</td>
<td>Altitude</td>
<td>-</td>
</tr>
</tbody>
</table>
## Acronyms

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIS</td>
<td>Artificial Immune System</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>CA</td>
<td>Control Activity</td>
</tr>
<tr>
<td>DNA</td>
<td>Deoxyribonucleic Acid</td>
</tr>
<tr>
<td>FDIE</td>
<td>Fault Detection, Identification, and Evaluation</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>GTM</td>
<td>Generic Transport Model</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IMRAC</td>
<td>Immune Model Reference Adaptive Control</td>
</tr>
<tr>
<td>LQR</td>
<td>Linear-Quadratic Regulator</td>
</tr>
<tr>
<td>MRAC</td>
<td>Model Reference Adaptive Control</td>
</tr>
<tr>
<td>NLDI</td>
<td>Nonlinear Dynamic Inversion</td>
</tr>
<tr>
<td>PI</td>
<td>Performance Index</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional Integral Derivative</td>
</tr>
<tr>
<td>PV</td>
<td>Performance Vector</td>
</tr>
<tr>
<td>RBF</td>
<td>Radial Basis Function</td>
</tr>
<tr>
<td>RMRAC</td>
<td>Robust Model Reference Adaptive Control</td>
</tr>
<tr>
<td>STR</td>
<td>Self-Tuning Regulator</td>
</tr>
<tr>
<td>TT</td>
<td>Trajectory Tracking</td>
</tr>
<tr>
<td>UAV</td>
<td>Unmanned Aerial Vehicle</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>WVU</td>
<td>West Virginia University</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

1.1 MOTIVATION

As robotics and automation become ever more entwined into the fabric of modern society, it is paramount that high-performance, fault-tolerant and adaptive control laws be developed to power these devices. As such, unmanned aerial vehicles (UAVs) are particularly difficult systems to control due to uncertainties that may occur in the modeling process, nonlinear equations of motion, and the wide range of failure or upset conditions that may occur. This makes them exceptionally well-suited for the development and testing of sophisticated adaptive control techniques.

The popularity and utility of unmanned aerial vehicles (UAVs) has increased dramatically in recent years. Military operations currently utilize UAVs for a wide range of reconnaissance and combat missions in order to keep human pilots out of harm’s way[1, 2]. In addition to military uses, UAVs have also found their way into many commercial and scientific uses. Popular examples are aerial surveillance, mineral exploration, transportation, search and rescue, forest fire detection, livestock monitoring, highway patrol, and atmospheric research.

Many of the current utilizations of UAVs are reliant on human operators or pilots to control the aircraft. However, the tasks given to UAVs are often very tedious for human operators. This can lead to pilots becoming fatigued and compromising the hardware and mission integrity. As such there is currently a strong push to create UAV systems capable of higher levels of autonomy[3]. Ultimately, greater autonomy will lead to increased efficiency and robustness with lower operating costs.

There are many components involved in making an autonomous UAV. Two of the core considerations in autonomous flight are path planning and trajectory tracking. The purpose of path planning is to produce a flyable path for the aircraft. Given an initial
coordinate, desired final coordinate, and possibly waypoints or obstacles, the path planner must be capable of creating a discretized set of spatial configurations that the aircraft will attempt to navigate using a trajectory tracking algorithm or controller. The problem of path planning has many well-established solutions for producing flyable paths that safely navigate to points of interest while avoiding obstacles. A thorough investigation of the most popular path planning methods can be found in Wilburn[4].

The other primary challenge in developing an autonomous aerial vehicle is actually satisfying the spatial configurations commanded by the path planner; this is the purpose of trajectory tracking. The specifics of various common trajectory tracking approaches will be covered in the next chapter, but for now it is important to note that there are many factors that make control of aircraft exceptionally difficult. In general, a controller must take into account vehicle kinematics and as well as nonlinear vehicle dynamics. Aircraft plant models are typically obtained through computer modeling[5], wind tunnel testing, or from parameter/system identification[6] methods using flight data. Consequently, the obtained model is only an approximation of the actual system. This results in a number of uncertainties that need to be accounted for in order to achieve good trajectory tracking performance. In addition to modeling uncertainties, internal and external forces can have a large impact on the accuracy of a given model. For example, the aerodynamic properties of the aircraft change based on factors such as altitude, Mach number, thermal effects, fuel usage, and atmospheric pressure[7]. Additionally, external forces such as atmospheric turbulence affect the model. An aircraft contains many subsystems that must all function properly. In addition to typical uncertainties, the model may change due to actual faults or failures among the subsystems. The problem of failed hardware is often much more significant on UAVs since they typically do not feature the same levels of redundancy as full-size commercial or military aircraft. Attempting to control an uncertain model with traditional controllers often leads to unpredictable results, and thus poor, to sometimes
catastrophic, performance. As such, the field of adaptive control has risen to meet the challenge of maintaining good performance in a system even in the presence of large uncertainties. Adaptive controllers modify their parameters or structure to maintain a desired level of performance even in the presence of large uncertainties, disturbances, or subsystem failures\[^8\].

Adaptive control is a relatively new field of study. The field emerged in the 1950s when engineers began pushing the capabilities of military aircraft. One of the first developed, and logically simplest, adaptive controllers developed was gain scheduling\[^8\]. This approach consists of scheduling controller gains depending upon the operating conditions of the plant. This basic adaptive control method led to the development of many novel control techniques. Methodologies in the family of adaptive controllers referred to as model reference adaptive control (MRAC) aim to coerce a given system into responding like a desirable reference system\[^9,10\]. Another popular group of adaptive controllers, referred to as self-tuning regulators (STR), attempt to estimate the state of the system online during operation and use the estimated model to update the control parameters. Adaptive controllers under the broad title of dual control introduce probing signals to the model in an attempt to gain information about the model that can be used to adjust the controller parameters.

Adaptive control allows the controller to be refined in real-time to ideally assure good performance even in the presence of large uncertainties or failures. In general, there are two families of adaptive controllers: direct and indirect. Indirect adaptive control methods first estimate new plant parameters; then adaptive gains (controller parameters) can be computed based on the updated plant model. With direct adaptive control methods, controller gains are estimated directly without needing to first update the plant model. In addition to direct versus indirect methods, adaptive controllers can also be classified as feedback or feedforward. A feedforward adaptive controller uses correlations between the disturbances and system performance to adjust compensatory
control action\[8\]. An adaptive feedback controller uses plant measurements to update plant estimates and create new compensatory action\[8\].

One should remark on the relationship between robust control and adaptive control. With robust control, the control system is tuned for a range of expected system parameters instead of a fixed set. As such, a robust controller is expected to perform well as long as the system does not deviate from the design range. Robust control methods use fixed controller gains that are optimized to perform well within a bounded range of system parameters; on the other hand, adaptive control adjusts gains in real-time to account for insufficiencies in the model or uncompensated external forces. As such, adaptive control is generally for handling larger error in a system. Often the best performance is seen in a blending of the two methodologies; robust control is used to design the initial controller, and adaptive control accounts for errors or faults outside of the design range\[8\].

The adaptive controllers introduced above each can play an important role in creating a system that is robust to failure. In general, however, they serve as only part of the total package for an integrated fault-tolerant system. With a system as complex as an autonomous UAV, there are many complicated subsystems which must work in harmony. This brings up the topic of fault detection, identification, and evaluation (FDIE)\[11\]. Essentially, it is logical that a system can be better controlled if the nature of its operating state and health are fully understood. If, for instance, a fault or other unexpected operating condition has occurred, it is important to be able to detect this fact. Once a failure has been detected, the system should be able to identify what type of fault has occurred, and in which subsystem. With full knowledge of the operating state of the system, a more effective adaptive control technique can be established. Finally, it is important to evaluate the severity of such a fault and what consequences it may have. This is the general concept behind FDIE.
The artificial immune system (AIS) paradigm is a biologically inspired artificial intelligence technique that has shown great promise in solving the problem of FDIE\cite{11-16}. The biological immune system is the human body’s defense mechanism against harmful invading organisms, or pathogens. It is capable of detecting, identifying, and evaluating the nature and severity of pathogens within the human body such that an appropriate defensive response can be initiated. The AIS paradigm draws inspiration from, and aims to emulate, the operation of the biological immune system.

As will be discussed in more detail later in this document, the AIS methodology uses detectors to identify operating states that do not belong to the typical operation of the system. Such a scheme is capable of detecting abnormal conditions. Test data can then be used to correlate individual detectors with specific failure classes and severities. When working with highly complex systems such as an aircraft, high dimensionality of detector space presents a problem in creating optimal sets of detectors. One approach to this problem developed a hierarchical structure for classifying abnormalities using lower dimensionality detector sets\cite{11}. In another approach, a genetic algorithm was used to optimize detector sets for better coverage of the non-self and to have less overlap with other detectors\cite{17}.

1.2 Research Objectives

The objective of this research is to create a more comprehensive, adaptive control architecture by combining concepts from the artificial immune system paradigm with concepts from classical adaptive controllers. The developed methodology is applied to the highly-complex and nonlinear problem of aircraft trajectory tracking. There are many aspects of the biological immune system that are not fully understood. As such, principles of classical optimal and adaptive control will be used to further the development of the AIS architecture. A thorough performance analysis will be
performed in a numerical simulation environment in order to demonstrate the fault-tolerant capabilities of the proposed adaptive control algorithm.

In addition to improving upon the existing classical adaptive control and AIS methodologies, a genetic algorithm (GA) will be developed in order to provide a systematic approach for tuning automatic controllers within the WVU UAV simulation environment. This will insure that performance improvements obtained can be attributed to the methodology and not controller tuning quality. To improve the efficiency of the genetic algorithm, a matrix-based approach is used within the genetic operations and evaluations.

The performance of the methodologies developed in this research is evaluated in simulation. This requires a sufficient simulation environment. The basis of such an environment was partially in place at the onset of this research; however, significant modifications were needed to facilitate the research at hand. Emphasis has been placed on updating its structure for modularity and efficiency. Certain modifications have also been made to facilitate the integration of the proposed AIS architecture and the GA.

The contributions of this research can be summarized as:

- Develop and test a novel, comprehensive adaptive control framework based on principles of artificial immune systems and model reference adaptive control algorithms for UAV trajectory tracking control.
- Analyze the effectiveness and performance of the proposed adaptive control scheme.
- Develop a back-propagation neural network for estimating aircraft angular rate parameters, to supplement the AIS detection scheme.
- Develop a genetic algorithm which ensures optimal tuning of the trajectory tracking algorithms being compared.
- Create a composite index for comparing the performance of a given trajectory tracking controller.
• Update the WVU UAV Simulation Environment with enhanced modularity, scenario configuration, and batch result collection procedures.

1.3 Thesis Overview

This document is organized as follows. Chapter 2 will present a literature review of the most prominent trajectory tracking algorithms, fault-tolerant control methodologies, and usages of the artificial immune system. Chapter 3 will provide an overview of the biological immune system followed by a discussion of the various methodologies of the artificial immune system paradigm. Chapter 4 provides an overview of the UAV simulation environment used to design and test the trajectory tracking controller. Chapter 5 presents an overview of the trajectory tracking architecture that was used and a few traditional trajectory tracking controllers, which were used as a basis of comparison for the artificial immune system methodology developed in this dissertation. Next, Chapter 6 will provide an overview of the genetic algorithm, which was developed to ensure that a given trajectory tracking controller has optimal gains. In Chapter 7, the approach for testing the performance of the trajectory tracking controllers will be presented along with a thorough comparison of the proposed methodology and existing controller architectures. Finally, conclusions and recommendations for future research will be presented in Chapter 8.
2 LITERATURE REVIEW

The creation of fault-tolerant, adaptive controllers for UAV trajectory tracking involves a total system approach. UAV’s are mathematically complex, highly-nonlinear systems to control. When faults and abnormal operating conditions are considered, the control problem becomes even more cumbersome. This chapter aims to outline the progress that has been made in several key areas that are important for trajectory control of UAVs.

2.1 TRAJECTORY TRACKING

Trajectory tracking controllers are responsible for making an unmanned aerial vehicle (UAV) follow a given path or trajectory. A plethora of controller architectures exists to handle the task of aircraft trajectory tracking.

A common approach for any complicated control system is to break the problem down into more manageable subsystems; with aircraft, it is common to use an inner-loop/outer-loop structure in which vehicle kinematic equations are used in the outer-loop to calculate the necessary angular rates or attitudes to track a given trajectory, while an inner-loop controller commands the aircraft aerodynamic control surfaces to satisfy the inputs from the outer-loop [18, 19]. Many approaches to aircraft trajectory tracking use classic or modified proportional-integral-derivative (PID) style controllers for control of the inner and outer control loops [20-23]. The structure of the inner-loop and outer-loop controllers varies depending on the individual approach and trajectory representation. One such approach uses a combination of PID controllers for reaction-based control of heading angle and altitude [24-26]. A more sophisticated approach which is often implemented is to solve for the required attitude states of the aircraft using the dynamic model of the aircraft through the process of nonlinear dynamic inversion (NLDI) [27-35]. Controllers based around NLDI are in general capable of cancelling out
non-linear dynamics encountered in the aircraft mathematical model. In many cases, NLDI based control is limited to control of the outer-loop, while more traditional control techniques command the inner-loop\[^{[30-35]}\]. This is due to the fact that the outer loop generally controls vehicle kinematics and is typically simpler to implement. More advanced techniques aim to extend the scope of NLDI based control to the inner loop as well, which must take into account the more complex vehicle dynamics\[^{[36]}\].

In addition to the conventional methods which have just been described, some aircraft trajectory controllers make use of optimal control techniques in order to provide mathematically guaranteed optimal performance characteristics. One of the most common optimal control techniques is the linear-quadratic regulator (LQR). This class of controllers utilizes a quadratic cost function to minimize the error and the control input of the system\[^{[37-40]}\]. This technique has been implemented for trajectory and stabilizing control for many UAV applications \[^{[41-44]}\]. Another popular optimal control technique is the linear-quadratic Gaussian (LQG) controller. The LQG controller is simply an LQR optimal regulator coupled with an optimal estimator, typically a Kalman filter, for state estimation\[^{[38, 39]}\]. It is useful for systems in which the full state is not available, requiring the output measurement to be used instead, and in situations when the output measurement is subject to stochastic disturbances. LQG are often applied on aircraft controllers since system noise can be significant and since the full state is often not available for measurement\[^{[45, 46]}\]. A final common optimal control technique is $H_\infty$ control. This control technique allows engineers to incorporate robustness criteria into the controller design process\[^{[37, 38]}\]. This method has been heavily used in the aircraft control field\[^{[45, 47, 48]}\].

### 2.2 Adaptive Control

Traditional control algorithms, such as PID and NLDI controllers, are not generally designed to be robust to uncertainties. For this reason, robust control techniques have
been developed to make controllers that are less sensitive to uncertainties and disturbances in the process. Such controllers do in fact help in many situations. However, robustness techniques often come with a tradeoff in performance; as robustness increases, performance generally decreases. For this reason, adaptive control techniques have been developed. Generally speaking, adaptive controllers should be capable of maintaining high levels of performance and robustness. An overview of this relationship is provided below in Figure 2-1.

![Figure 2-1. Performance versus robustness trade-off](image)

Adaptive control allows the controller to be refined in real-time to ideally ensure good performance even in the presence of large uncertainties or failures. In general, there are two families of adaptive controllers, namely direct and indirect control. Indirect adaptive control methods first estimate new plant parameters; then adaptive gains (controller parameters) can be computed based on the updated plant model. With direct adaptive control methods, controller gains are estimated directly without the need for explicitly updating the plant model. Besides direct versus indirect methods, adaptive controllers can also be classified as feedback or feedforward. A feedforward adaptive controller uses correlations between the disturbances and system performance
to adjust compensatory control action. An adaptive feedback controller uses plant measurements to update plant estimates and create new compensatory action. Some of the most commonly applied adaptive control techniques will be discussed in the following subsections.

2.2.1 Gain Scheduling

As with fixed-parameter controllers there are also many classifications of adaptive controllers. One of the simplest and earliest adaptive control methods developed was gain scheduling. In gain scheduling, multiple sets of fixed controller gains are developed to provide satisfactory performance at different operating points of a nonlinear system. The set of gains used by the controller depends upon observations of variables known to have an effect on the plant model. These observed variables are known as scheduling variables\(^{(49)}\). Relating this to aircraft control, altitude and Mach number are often used as scheduling variables since they have large effects on the plant model. Different operating condition combinations, such as Mach number and altitude, are commonly referred to as points in the flight envelope. When the aircraft is operating at a given point in the flight envelope, a corresponding linearized plant model and set of controller gains are used. When the aircraft goes to another point in the flight envelope, the model and gains are switched to another set. A schematic of gain scheduling can be seen below in Figure 2-2.
While gain scheduling at its core is a relatively simple adaptive control technique, the final controller generally uses more complicated schemes for the system state measurement or identification. Lee et al.\cite{50} implemented a fuzzy-logic-augmented gain scheduling scheme on vertical take-off and landing UAVs. The fuzzy logic augmentation served to better cover the range of operating conditions. Sadeghzadeh et al.\cite{51} implemented an active fault tolerant gain scheduling scheme for a quadrotor UAV in which the gain schedule selection was determined by a separate FDIE scheme. Chengfu et al.\cite{52} implemented a linear parameter-varying system-based gain scheduling scheme that they applied to a morphing-wing UAV in order to change the wing angle to accommodate for mission requirements.

The concept of gain scheduling is very primitive as compared to other adaptive control techniques. It is logical that modifying the gains of an automatic controller can alter the performance of the controller. As such, when the operating conditions of the system are altered, the old gains can be replaced with new ones in order to maintain satisfactory performance. The problem arises in that the controls designer must have very extensive knowledge about every operating condition that could be experienced, i.e., the designer must determine which variables correlate to changes in the model behavior and appropriate gains for each condition. Developing effective gain scheduling...
controllers can be a very time- and resource-intensive process. Even with such thorough design, it is still likely that if the operating conditions leave the range of design specifications, unsatisfactory performance will be experienced.

2.2.2 Model Reference Adaptive Control

Model reference adaptive control (MRAC) is a commonly used method for adaptive control\[^8, 37, 53, 54\]. Unlike the gain-scheduling paradigm, MRAC does not heavily rely on having accurate plant models. Instead, the actual system is coerced into responding like a reference model which embodies the desired response characteristics. The output of the actual system is compared with the output of the desired reference model. The error between the two is used to adapt the controller parameters such that ideally the output of the actual system will converge to the output of the reference system model. This method can be used in either a direct or indirect manner. Indirect adaptive control using MRAC is discussed in this dissertation.

MRAC was originally developed for aerospace applications. It was first used to shape the dynamic handling characteristics of a human piloted aircraft to match the handling preferences of the pilots. Another use is for eliminating uncertainties in the modeling process. An approximate model is obtained through testing and verification. However, sizable errors may still be present in the process. The anticipated model, or a desired one, is used as the reference model in the MRAC process; then the system can adapt to respond like the reference model. One key feature of MRAC for use in aerospace applications, is that depending on the formulation, rigorous stability proofs have been developed\[^9, 37\].

In general, an MRAC mechanism is added to a traditional state feedback controller. The traditional state feedback loop serves as an inner-loop controller. The outer-loop consists of a reference model and an adjustment mechanism for adapting the inner-loop
control parameters. A generalized block diagram of this architecture can be seen in Figure 2-3.

![Figure 2-3: Overview schematic of model reference adaptive control](image)

The adjustment mechanism of an MRAC controller is responsible for driving the error between the reference model and the actual model to zero. This mechanism can be computed in several ways. The two most common methods are applying a stability criterion (Lyapunov rule) or by using a gradient-descent method (MIT rule).

Schreier et al.\cite{55} used traditional MRAC control for altitude stabilization of a quadrotor. This was used in conjunction with a model identification adaptive controller for self-tuning of its controllers. Orsag et al.\cite{56} implemented a Lyapunov-based MRAC controller on a quadrotor with a multi-DOF manipulator. The MRAC control served to ensure dynamic stability of the aircraft while operations were performed using the manipulator. Solomon\cite{57} applied a traditional MRAC scheme to the brushless DC motor serving as the propulsion system for a UAV system. This resulted in steadier speed control, and thus more consistent propulsion for the aircraft. MRAC will be discussed in more rigorous detail later in this document during the development of this research’s core contribution.
2.2.3 **Self-Tuning Regulators**

A self-tuning regulator (STR) is another common type of control scheme capable of dealing with uncertainties in a given system model. A self-tuning regulator combines the design of a control system from a known plant model and identification of system parameters for an unknown system. Well-established approaches exist for each of these two components. With this method, system parameters are estimated in real-time, and then the estimated system parameters are used to update the controller parameters\(^8\). The estimated system parameters are used as though they are correct; this is often referred to as the *Certainty Equivalence Principle*. An overview of this method can be seen below in Figure 2-4.

Self-tuning regulators can operate either as direct or indirect. With an indirect approach, an estimator is used for the system model and the controller design. This has the advantage of being simple, but often results in too much computational overhead. This is typically referred to as explicit self-tuning control. With a direct approach, the system model is parameterized in terms of the controller, thus combining the two steps.
This saves computational time and effort. This is referred to as implicit self-tuning control.

Like MRAC, self-tuning regulators also consist of two primary loops. The inner loop acts as a typical feedback controller, while the outer loop provides adaptation for the inner-loop controller. Unlike MRAC, self-tuning regulators do not use a reference model; instead, the actual model is estimated using parameter identification techniques (note that in some texts, STRs are regarded as a type of MRAC depending on structure and nomenclature). Any number of parameter identification techniques can be used to estimate the system model. Likewise, the possibilities for the controller design are practically unlimited so long as they can be computed fast enough to work online and take into account any nonlinear behavior. This structure can also be regarded as a form of gain scheduling with the distinction that the model does not need to be determined \textit{a priori}.

Benjanarasuth\textsuperscript{[58]} implemented direct self-tuning regulators on a model helicopter for the purpose of controlling the pitch of the vehicle. The STR implemented in this application made use of a weighted recursive least squares algorithm for the system estimation and traditional pole-placement for the controller design. Chowdary and Johnson\textsuperscript{[10]}, proposed a least squares method for identification within a STR.

\textbf{2.2.4 Dual Control}

As summarized by Sevcik \textsuperscript{[59]}, dual control refers to a class of controllers which have two primary goals. First, the controller should control the system as well as possible. In addition to controlling the system, the controller should try to gain more information about the system through use of a \textit{probing} signal, i.e., a test maneuver. Additional information gained by the probing signal can subsequently be used to improve the control of the system. This step is counter-intuitive to previous controllers in that it must actively disturb the system to obtain more information, and the control goal is
typically to eliminate disturbances. The two-fold compromise between control and probing is where the method takes the name dual control. This technique is often used when initial estimates of the system are poor and the time horizon is short. Dual control methods take the accuracy of their plant estimations into account, and provide excitation to speed up estimation convergence, thus speeding up adaptation.

Adaptive control methods that both control a process and perturb it to accelerate the parameter identification process are referred to as dual control methods. Feldbaum\cite{60, 61} proposed a solution to the dual control problem in the form of a functional equation commonly referred to as the Bellman equation. Solving this requires the use of dynamic programming.

Dual control schemes are classified as either optimal or suboptimal. Optimal dual controllers are the type originally proposed by Feldbaum in 1961\cite{60, 61}. Suboptimal dual controllers arose to solve difficulties in calculating optimal dual controllers.

Liu et al.\cite{62} implemented dual control on a supersonic missile platform. This method was a hybrid approach which mixed traditional PID control with a backstepping method. Yung et al.\cite{63} implemented dual control on a precision stepper motor servo.

2.2.5 $L_1$ Adaptive Control

Model reference adaptive control (MRAC) has been shown to be a valuable adaptive control technique. As stated before, the goal of MRAC is to force a given plant model to behave like a desired reference model. MRAC control can be used in several different configurations such as direct or indirect, with several variations on the format of the adaptation mechanism, and with variations in the representation of the reference model.

While MRAC is a powerful adaptive control architecture, the traditional implementation does possess drawbacks. Several approaches have been developed for adapting the adjustment mechanism of MRAC controllers; however, generalized
systematic approaches to tuning its parameters, such as rate of adaptation, are lacking. Additionally, much effort has been placed on proving asymptotic stability using the methods proposed by Lyapunov. The overall stability of the method has been proved. However, the asymptotic stability of the parametric estimation errors is not guaranteed; the parametric estimation errors are only guaranteed to be bounded. Proving the stability of such systems is very important. However, the transient response and overall robustness are often neglected. This highlights one of the key issues with MRAC control, as well as other adaptive controllers; as adaptation rate increases to rapidly respond to errors in the system, the phase margin, and thus overall robustness decreases. The classical MRAC method exhibits a poor response to time delays; this is only exacerbated as the adaptation rate increases.

Hovakimyan and Cao\cite{64} proposed a novel version of MRAC, called $L_1$ adaptive control, to allow for rapid adaption, while maintaining high robustness. The $L_1$ adaptive control algorithm derives its name from the use of the $L_1$ norm, also known as the taxicab or Manhattan norm, in the bounded input, bounded output stability proof. While the full formulation and stability proofs of the $L_1$ are much more rigorous, in essence, an $L_1$ controller is a state predictor MRAC augmented with a low-pass filter. This feature allows for the adaptation gain to be scaled arbitrarily high (as limited by hardware) to improve the performance of the model tracking while minimizing robustness issues generally encountered from the increased adaptation gain; thus, $L_1$ separates performance and robustness in the design. Consequently, the typical performance/robustness trade-off of adaptive controllers is lessened. In addition to this more robust nature, the $L_1$ adaptive controller also provides a well-defined mechanism for tuning its parameters. This greatly simplifies design effort required. For these reasons, the $L_1$ adaptive control methodology is given its own classification over MRAC since its operating characteristics are improved and unique.
As will be discussed later, $L_1$ adaptive controller is not without its faults. For one, the introduction of the low-pass filter also reshapes the response of the system to not exactly follow the reference model; while transient response is improved in many situations, it does mildly invalidate the assumption of reference model following. Additionally, the method introduces more stringent sample rate restraints that are often not possible in some systems due to hardware restraints. Furthermore, like MRAC, $L_1$ adaptive control still requires the tuning of certain parameters to achieve the desired performance; the transient response of the system is directly related to the choice of low-pass filter. Finally, similar robustness and performance gains can be obtained through modification of classical MRAC. Similar theoretical concerns regarding $L_1$ adaptive control have been expressed by Jafari et al.$^{[65]}$

Regardless of the above-described criticisms, $L_1$ adaptive control is currently given significant attention within the research community. The $L_1$ adaptive controller can be used in most applications where a normal MRAC or other adaptive controller would be used. It was used by Cao and Hovakimyan$^{[66]}$ for UAV trajectory tracking. In this application, the onboard autopilot system was augmented with an $L_1$ adaptive controller. This application also presented a lengthy stability proof of the $L_1$ architecture using the principles of Lyapunov. In addition to this application, the $L_1$ architecture has also been implemented on the NASA generic transport model (GTM) in a nonlinear dynamic inversion-based trajectory tracking controller$^{[67]}$. In addition to the trajectory tracking UAV applications discussed, the $L_1$ architecture has been implemented in missile guidance$^{[68]}$ and camera gimbal tracking$^{[69]}$.

### 2.2.6 Neural Networks

In addition to the adaptive control methodologies presented thus far, there are many more adaptive control techniques. One adaptive method that has shown much success is artificial neural networks (ANNs). An ANN is a biologically inspired,
artificial intelligence technique capable of a specific type of adaptation referred to as learning. As such, they can be used to replace various modules of the traditional adaptive controllers or as independent schemes. One of the most common uses of neural networks in adaptive control is in the estimation of the process model. In this approach, training data is used to teach the ANN general knowledge about the structure of the model, then when uncertainties arise, the network adapts accordingly. While promising, computational overhead, coupled with the necessity for large amounts of training/validation data, have kept their actual uses limited. In addition, ANNs consist of a complicated structure of neuron models that is often treated as a black box process, i.e., it is difficult to accurately prove or predict the inner workings of the controller. As such, researchers have had little success proving the stability of controllers developed using such methods, which is of utmost importance for any such controller to get certification for use in commercial applications.

2.3 Genetic Algorithm

One problem with practically any of the trajectory tracking controllers and adaptive control techniques listed above is that they all contain gains or other control parameters, the selection of which have a profound effect on the controller’s functionality and performance. Different mathematical and optimal control techniques exist for tuning a set of controller gains. However, the highly-complex nature of aircraft often makes such deterministic tuning approaches exceedingly difficult or impossible depending on the method. For this reason, another approach is needed for developing a suitable set of gains for a controller. One common approach is to heuristically guess-and-check controller gains offline in simulation until a suitable solution is found. This approach may lead to an acceptable controller; however, such an approach can be very time-consuming, i.e., costly, and it offers no guarantees of optimal performance. As such, a simple, systematic method for the optimization of controller gains is needed. This is
important for the actual operation of the controller in the system – *optimal* control performance is almost universally desired. Additionally, it is critical when drawing conclusions about the benefits and disadvantages of a given novel controller as compared to historic approaches that all controllers in the study operate at their highest potential; this imperative step is often conveniently neglected during literature regarding the development and analysis of novel control algorithms. One such methodology capable of such requisite optimization is a genetic algorithm (GA).

Genetic algorithms are an artificial intelligence technique often used for optimizing complex systems. They are based on the principles of biological evolution such as natural selection. In one application, Austin et al.[70] used a GA for hypersonic flight control. In another application Guo et al.[71] designed an automatic steering controller for unmanned ground vehicles using GAs. Kim et. al.[72] used a GA augmented with fuzzy logic for trajectory tracking. Darrah et al.[73] used GAs for UAV mission planning. Perhinschi[74,77] implemented various GAs in the tuning of helicopter control laws. The particular GA developed in this research effort will be discussed in further detail in Chapter 6.

### 2.4 Artificial Immune System

Artificial intelligence (AI) techniques have shown great promise in adaptive control applications. The biological immune system (BIS) has given much inspiration to the field, yielding several distinct AI paradigms, all broadly referred to as the artificial immune system (AIS). One of these methods aims to emulate the body’s highly robust ability to detect, identify, and eliminate invading pathogens without harming its own cells[12]. Artificial immune systems were first used for detection and identification of system abnormalities and failure conditions. Example traditional detection and identification uses of AIS include pattern recognition[78,79], computer security[13,14,80], and
data mining\textsuperscript{[81]}. Such techniques do not directly adapt automatic controllers, but instead alert the user/system to issues with the system health and performance.

More recently, the AIS paradigm has been used for adaptive control. There are two dominant approaches to using AIS for control purposes. The first method gets its inspiration from the way in which the biological immune system suppresses its own reaction to invading antigens; the innate response of the immune system acts swiftly to neutralize threats, then a suppression mechanism steps in to stabilize the system and prevent a toxic reaction\textsuperscript{[82]}. This direct adaptive control technique has been used in many applications such as servo response stabilization\textsuperscript{[82]}, control of tank system liquid fill rates\textsuperscript{[83]}, and super-heated turbine control\textsuperscript{[84]}. The second method is a detector-based solution, which is an extension of the methodology used for AIS fault detection and identification. Similar to the detection and identification uses for AIS, detector sets are used to identify operating conditions in a given system, and compensator action is extracted from a self database\textsuperscript{[85, 86]}. Within this architecture, the compensatory action extracted from the self-database must be trained with numerous flight-test data.
3 ARTIFICIAL IMMUNE SYSTEM

3.1 BIOLOGICAL IMMUNE SYSTEM

The biological immune system is a naturally occurring adaptive control system which protects the human body from invading pathogens, which are microorganisms, such as bacteria or viruses, that are capable of causing disease. The biological immune system has given inspiration to adaptive immune system control techniques. Within the human body, the biological immune system is used to protect the body from dangerous pathogens. The immune system comprises two primary components: the innate immune system and the specific immune system.

The innate immune system protects the body from pathogens in a non-specific manner, i.e., it does not require prior knowledge about the invading pathogen. There are four primary functions within the innate immune system: anatomical barriers, the cellular response, the inflammation response, and the complement system. Anatomical barriers are the body’s first line of defense. They are barriers which physically inhibit pathogens from entering the body. Examples of such barriers are skin and the various mucous-lined passages of the respiratory and gastrointestinal systems. If a pathogen gets past the body’s anatomical barriers, it will encounter other cells and chemicals designed to neutralize it. Leukocytes, commonly referred to as white blood cells, are specialized defensive cells typically formed in the bone marrow that play an integral role in this immune response. The body contains five different broad classes of leukocyte: neutrophils, eosinophils, basophils, monocytes, and lymphocytes. The first four types are the primary cellular components of the innate immune response. Generally speaking, these cells capture invading pathogen, devour, or phagocytize, them, and present pieces of them, referred to as antigens, on their surface using major histocompatibility complexes (MHCs). This mechanism fights off infection directly by
destroying pathogens, and it also serves to trigger the adaptive immune system through antigen presentation. As will be discussed later, the last type of white blood cell, lymphocytes, are the primary component of the adaptive immune system. The inflammation response is another mechanism that the innate immune system uses to eliminate pathogens from the body. This mechanism is stimulated by chemicals released by injured body cells. This response releases chemicals that promotes swelling which in turn creates additional physical barriers at the location being attacked. Additionally, chemicals are excreted that attract white blood cells to the site of the attack to aid in removing dead or damaged cells. Finally, the complement system helps the cellular response to rid the body of pathogens. This mechanism is responsible for the synthesis of several proteins that aid in the tagging of pathogens for removal, destruction of pathogens by forming holes in their plasma membranes, removal of dead pathogens, and triggering of the inflammation response.

The second primary component of the immune system is the specific or adaptive immune system. When the body encounters a new pathogen, the adaptive immune system develops a memory to the specific pathogen such that subsequent encounters can be eliminated more efficiently. Since the response mounted by this system is specific to a given threat, this mechanism is sometimes referred to as the specific immune system or acquired immunity.

The immunological memory formed by the adaptive immune system is the premise behind vaccination. During vaccination, a small number of weakened pathogens for a given disease are introduced into the body such that the adaptive immune system can learn to recognize and neutralize them in a low-risk situation. Consequentially, if a real threat is encountered in the future, the body will already know how to deal with the threat.

The adaptive immune system comprises lymphocytes and antibodies. Lymphocytes originate through a pseudo-random process from multipotent hematopoietic stem cells
in the bone marrow. A portion of these cells travel to, and mature in, the thymus while others initially remain in the bone marrow. The cells which remain in the bone marrow are referred to as B-cells and the cells which mature in the thymus are commonly referred to as T-cells.

The B-cells, which form the *humoral response*, circulate throughout the bloodstream and have a special receptor on their surface to help recognize and bind to specific free-floating antigens and neutralize them. A B-cell becomes *activated* once it is bound to a specific antigen. Once activated, the B-cells generate antibodies corresponding to that pathogen. These antibodies are Y-shaped proteins with MHC molecules on their ends that travel through the blood plasma and bind to other pathogens with matching receptors. Numerous antibodies may bind to an individual pathogen which in turn hinders its movement and functionality, in addition to marking it for removal by phagocytotic cells. After initiating a response to an antigen, some of the B-cells are retained as *memory B-cells*.

The T-cells, which form the *cell-mediated response*, are responsible for killing body cells which have become infected by pathogens. As such, T-cells must be able to distinguish between healthy and virally infected or cancerous body cells. Several different types of T-cells are matured in the thymus. During the maturation process, genetic processes *teach* the T-cells to recognize body, or *self*, cells from infected, or *non-self*, cells. *Cytotoxic T-cells*, Tc cells, are responsible for the actual destruction of compromised, non-self cells. *Helper T-cells*, or Th-cells, do not directly kill non-self cells, but instead assist in the adaptive immune response. Once a helper T-cell has become activated by binding with an antigen, it releases *cytokines* that stimulate or regulate the response of other adaptive immune system cells. *Regulatory*, or *suppressor*, T-cells are crucial for maintaining a balance in the immune response. The T-cells are vital to regulating the production of B-cells, and thus the production of antibodies. If a large number of antigens is detected, more Th-cells and fewer Ts-cells are produced, resulting
in more B-cell production. As the number of antigens present in the body decreases, the number of suppressing Ts-cells is increased, and the number of Th-cells is reduced, which ultimately leads to fewer B-cells and antibodies. Eventually, this process equalizes and the immune response is complete. An overview of this process is provided below, in Figure 3-1.

Figure 3-1. Overview of the biological immune feedback response

3.2 **Artificial Immune Feedback Suppression Mechanism**

As formulated by Takahashi et al., an immunity feedback control law was developed based on the above described biological feedback suppression mechanism. When applied to control systems, error in the system is analogous to invading pathogens. In this control scheme, a higher than usual control gain serves to quickly react to errors, then the suppression mechanism steps in to stabilize the otherwise likely unstable response. In the biological immune system, the number of B-cells is related to the number of helper Th-cells and suppression Ts-cells as follows:

\[ B(k) = T_h(k) - T_s(k) \]

The helper Th-cells can be analogized as a simple proportional controller:
\[ T_h(k) = k_1 \varepsilon(k) \]

where \( k_1 \) is the stimulation factor, and \( \varepsilon(k) \) is the error in the system. The suppression cells react proportionally to a nonlinear function:

\[ T_s(k) = k_2 f[\Delta B(k)] \varepsilon(k) \]

where \( k_2 \) is the suppression factor, \( \Delta B(k) \) is the concentration change of the B-cells, and \( f[\ast] \) is a nonlinear function. From this, the number of B-cells in the system can be expressed as:

\[ B(k) = K (1 - \eta f[\Delta B(k)] \varepsilon(k)) \]

where \( K = k_1 \), and \( \eta = \frac{k_2}{k_1} \). In the control input to the aircraft, \( u(k) \) corresponds to the stimulation received by \( B(k) \), then the feedback control can be re-formulated as:

\[ u(k) = K(1 - \eta f(\Delta u(k))) \varepsilon(k) \]

This adaptive gain can be utilized within controllers such as a discrete conventional proportional-integral-derivative (PID) controller. This can be expressed as:

\[ u(k) = k_p \left( 1 - \eta f(\Delta u(k)) \right) \left( 1 + \frac{k_i}{z - 1} + k_d \frac{z - 1}{z} \right) \varepsilon(k) \]

where:

\[ f(\Delta u(k)) = 1 - \exp \left( - \frac{(\Delta u(k))^2}{a} \right) \]

This function can be seen below in Figure 3-2.

![Figure 3-2: Nonlinear suppression function](82)
As will be shown in Chapter 5, several baseline, fixed-parameter controllers have been augmented with this adaptive AIS suppression mechanism. This mechanism has shown some limited success in accommodating for faults and abnormal conditions for UAVs. Additionally, as will be discussed in more detail in Chapter 8, one of the main contributions of this research is in expanding the original formulation of this mechanism and incorporating features from more traditional adaptive control techniques.

3.3 Artificial Immune System Negative Selection Strategy

The other common immunity-based adaptive control mechanism is inspired from the biological immune system’s ability to identify and eliminate pathogens invading the body. The immune system learns to detect and eliminate threats from previous exposure to invading antigens. This process relies on specialized cells called phagocytes, which act as an initial defense system. These are generated using a negative-selection process which bonds to biochemical markers specific to the particular invading pathogen. These markers transfer this information to lymph nodes where lymphocytes are generated. As mentioned above, the two primary types of lymphocytes are T-cells and B-cells. These cells are pseudo-randomly generated in the thymus; this ensures high variability. The antibodies identify invading pathogens and mark them for destruction by cytotoxic T-cells. To improve future responses, memory B-cells and T-cells are generated. The negative-selection process by which this occurs produces biological agents capable of identifying non-self entities in the body and ignoring the body’s own cells. A generic overview of this process is provided below in Figure 3-3.
For the AIS paradigm detection process, detectors are created to identify *self*, or normal operation, or *non-self*, abnormal operation. This is created by collecting vast amounts of operational data, e.g., flight tests at nominal conditions. This self data is clustered into $n$-dimensional hyperspaces. During operation, data points not falling within a self detector are considered to be non-self or abnormal. This process requires much offline pre-processing of the data to optimize the data for quick and thorough evaluation. The identification data is collected to generate detectors corresponding to particular failures. This data can then be arranged into a structured self/non-self within the *hierarchical multi-self* strategy\[11\], which is capable of classifying particular failures quickly. This is a brief generalization for the processes of detection and identification. An example 2D projection of a structured identification detector set can be seen below in Figure 3-4.
Aside from failure detection and identification, the detector-based AIS methodology can also be used for adaptive control. This method considers the control action in the definition of the self. Both feature sets, control action, and controlled variables are stored and mapped to detectors. This creates a mapping between control variables and compensatory commands. As with detection and identification, these compensatory commands are clustered into hyper-features that may later be recalled by the central point or other parameters\textsuperscript{[86]}. 

![Figure 3-4: Example structured identification detector set\textsuperscript{[85]}](image_url)
4 TESTING ENVIRONMENT

The fault-tolerant adaptive control architecture developed in this dissertation was tested in a numerical simulation environment. This chapter describes the capabilities and features of this simulation environment. First, the overall architecture of the environment and validation of the aircraft model is described. Next, a set of performance metrics is defined in order to quantify the relative performance of the trajectory tracking controllers being evaluated. Finally, the batch process used to collect the results will be described.

4.1 WVU UAV SIMULATION ENVIRONMENT

A MATLAB/Simulink simulation environment was developed at WVU to facilitate the design and analysis of unmanned aerial systems\cite{7,88}. The simulation environment aims to provide a highly modular tool for developing and testing intelligent fault-tolerant trajectory tracking controllers and path planners. A simplified simulation environment was already present at the onset of this research; however, a sizeable portion of this research effort has been devoted to updating this environment to be more modular, and to incorporate the next generation capabilities of MATLAB/Simulink and modern visualization software.

4.1.1 ARCHITECTURE

The WVU UAV simulation environment consists of three primary components. These are the Simulink simulation, FlightGear, and a custom graphical mission monitoring and configuration tool called the UAVDashboard. An overview of the environment relationships can be seen below in Figure 4-1.
A highly modular, singular Simulink model was designed such that it can be quickly reconfigured with any number of different aircraft models, trajectory planners, trajectory tracking controllers, failure models, and atmospheric conditions. Configurable subsystem blocks are used with libraries to provide a means of quickly selecting or interchanging the various models and subsystems. Model callbacks are often utilized for purposes such as switching from real-time to accelerated time, plotting historical records of state variables or performance histories, and loading pertinent model variables and parameters when the simulation loads or a module is interchanged. Relevant modules are masked with a graphical user interface (GUI) to further facilitate the configuration of system parameters such as failure magnitudes and times. This will be discussed in greater detail in the upcoming sections.

Asides from the rapid reconfiguration properties of the Simulink model, it also communicates with FlightGear and the UAVDashboard via local user datagram
protocol (UDP) network sockets to relay display information about the state of the aircraft. In addition to the aforementioned capabilities, the Simulink model also interfaces with a hardware joystick connected to the computer for the purpose of manual flight when no autonomous modes of operation are selected. The Simulink model can be seen below in Figure 4-2.

Figure 4-2. WVU UAV Simulation Environment version 3.0 Simulink model

The second component in the WVU UAV simulation environment architecture is the open-source aircraft visualization software, FlightGear. This software package is used for 3D visualization of the given aircraft in realistic environment. FlightGear receives aircraft state information over a UDP connection and displays a realization of the aircraft flying in the specified region. While purely a visual refinement, the aircraft model used by FlightGear updates according to the model being used in the simulation environment. A typical visualization obtained from FlightGear can be seen below in Figure 4-3.
Mission scenarios pose different objectives and constraints for aircraft path planners. A common criterion is that the produced path must avoid obstacles within some margin. Obstacles can be physical structures such as buildings or other aircraft which must be avoided to prevent possible catastrophic damage. In addition to actual obstacles, some planners also aim to minimize exposure to threats such as enemy radar. Another common objective of path planners is to achieve a final pose $P(x, y, z, \theta, \psi)$ where $(x, y, z)$ are desired location coordinates, and $(\theta, \psi)$ are the attitude angles pitch and heading. For the purpose of intelligent control and mission planning, many considerations are needed. As such, a tool for specifying such mission configuration was created. The software package, UAVDashboard, is a mission configuration and visualization tool that was developed in Microsoft Visual C#. Its primary goal is to allow for the specification of the aircraft positions, altitudes, and orientations. In
addition to aircraft, the software also allows for placing and sizing threat zones on an aerial map of the given environment. A threat zone can be either an obstacle that would likely cause a crash if contacted, or it could be simply an area of elevated risk. The various trajectory planners use this information to create paths with minimal threat exposure. The aircraft and threat zone specifications are saved to a configuration file in the simulation working directory and are used during the initialization scripts. The second goal of the UAVDashboard software is to provide a means for 2D visualization. An aerial map of the given airport/region used by FlightGear is preloaded into the UAVDashboard software. During a given simulation, aircraft state data, as well as desired and actual trajectory data, are sent to the UAVDashboard via UDP connection and displayed on the given map. A typical view of the software can be seen below in Figure 4-4.

Figure 4-4. WVU UAVDashboard software package
4.1.2 AIRCRAFT MODELS

When developing adaptive control algorithms for aircraft, it is important to test their functionality on several different platforms to assess their robustness. For this reason, the WVU UAV simulation environment currently incorporates five different UAV models. The simulation environment makes use of modular configurable subsystems such that the simulation can quickly be reconfigured to utilize another aircraft.

The most commonly used aircraft model in the simulation environment is WVU YF-22 research aircraft. The WVU YF-22 aircraft is inspired from the U.S. Air Force F-22 fighter aircraft. The YF-22 is scaled down to approximately 15% of the actual aircraft dimensions. It is powered by a miniature jet engine, and carries enough fuel for missions of approximately 12 minutes. This 6’6” research aircraft is capable of carrying a maximum of 12 lbs payload for a maximum take-off weight of 50 lbs. The physical YF-22 research aircraft is depicted below in Figure 4-5.
Numerous flight tests have been performed using the YF-22 research aircraft. The flight data from these tests were used with parameter identification techniques to develop a non-linear mathematical model of the aircraft which was later implemented in Simulink. The non-linear model consists of a set of differential equations which incorporate basic physical variables; aerodynamic coefficients such as $C_D$, $C_Y$, $C_L$, etc. are used to compute the aerodynamic forces and moments acting on the aircraft. In addition to the non-linear model, which is used for computing the aircraft state in the simulation, a linearized model was also developed around one operating condition. This linearized model, which will be discussed in more detail in Chapter 5, is used for design purposes in many of the trajectory tracking control laws. A complete description of this aircraft and the formulation of the aerodynamic model is given in Napolitano et al.\cite{6}. The simulation model for the YF-22 is depicted above in Section 4.1.1.

In addition to the YF-22 research aircraft, four other aircraft models are implemented in the WVU UAV simulation environment. Three military aircraft, the RQ-2 Pioneer, the TigerShark UAV, and the OX UAV are implemented in the environment. A more thorough description of these three aircraft, along with the methodologies used in developing their accompanying mathematical and Simulink models can be found in Karas\cite{5}. The FlightGear visualizations for the Pioneer and the TigerShark UAVs can be found in Figure 4-6 and Figure 4-7 respectively.
The final aircraft implemented in the WVU UAV simulation environment is the NASA Generic Transport Model\[^89\] (GTM). The NASA GTM is a 5.5% aerodynamically scaled down implementation of the Boeing 757 commercial airliner. The use of this model is being phased out in the updated version of the simulation environment due to the vastly different structure of the model from the other aircraft and thus increased difficulty of incorporating the UAV into the modular environment design.

The highest fidelity model available at the time of performing this research was the YF-22. As such, the analysis performed here within is conducted using that aircraft.

### 4.1.3 Failure Models

Several abnormal conditions, such as aircraft sub-system failures, and environmental upset conditions, such as strong wind or excessive turbulence, have been modeled and implemented within the WVU UAV simulation environment. This allows for the trajectory tracking controllers to be evaluated for robustness to uncertainties and failure.

The simplest failure incorporated is a failure on the actuation system for a control surface. The two damage models implemented are a *frozen* surface in which a given
control surface is locked in its current position at a specified time and an imposed surface deflection in which the control surface is locked at a specified position at a specified time. In addition to these failures, the control surfaces have been modeled with response time and physical limitations in mind.

Another abnormal condition model incorporated into the simulation environment is strong wind or excessive turbulence. The direction and magnitude of a constant wind can be set directly in the Simulink model. The severity of atmospheric turbulence, based on the Dryden model, is defined in terms of the square of the standard deviation of air velocity in m/s within the three Earth-based coordinate axes.

In addition to control surface failures and atmospheric upset conditions, failures have also been considered on the gyroscope sensor channels. Different classes of failures were considered for the gyroscope sensor. One such failure that was modeled represents a bias in the output. This bias can occur instantaneously as a step or linearly over a given time, referred to a drifting bias. Different constant values are used for small or large biases. In addition to a bias, noise can also be injected into the measurement signal.

Another sensor system which was modeled for failures and abnormal conditions is the global position system (GPS). A simplified GPS model was developed for the WVU UAV simulation environment by Al-Sinbol[90]. This model accounts for errors and delays in the measurements and the update rates. The GPS model can be disabled to give the absolute position, functioning within normal operational ranges, or operating outside the expected operating range. Abnormal conditions include biases and excessive noise.

The last adverse condition considered is that of an uncertain mathematical model. Many assumptions, approximations, measurement errors, and simplifications are made during the task of developing a mathematical model of a given aircraft. This can result in a model that does not accurately describe the dynamics of the aircraft. Such failures
manifest themselves in the model as modifications in the aerodynamic coefficient and state-space representations. Consequentially, control schemes are unlikely to behave as expected if they were developed using a mathematical model that deviates significantly from the actual aircraft dynamics. For this reason, such uncertainties are analyzed in this research. Due to the nature and generality of modeling uncertainties, a modular/graphical utility is not provided in the simulation environment for such modifications. Instead, such uncertainties are introduced directly by modification of the mathematical model initialization scripts.

All of these failures, with the exception of model uncertainties, have been implemented with an emphasis on modularity. The failure models are set aside in separate library blocks which may easily be added to any aircraft model following the same modeling conventions. Each failure subsystem module features a Simulink subsystem mask which allows the user to conveniently reconfigure the failure scenario. As an example, the surface failure GUI and model are shown below in Figure 4-8 and Figure 4-9 respectively.
Figure 4-8. Control surface failure configuration GUI

Figure 4-9. Control surface actuator/damage model
4.2 **Performance Indices**

A series of performance metrics was formulated in order to assess the performance of an aircraft tracking a commanded, fixed trajectory. This set of performance metrics is used to grade the aircraft’s performance for two fundamental objectives. These performance metrics are discussed in greater detail in Wilburn *et al.*[91]

The first performance objective is that a given trajectory tracking controller should follow a commanded trajectory with as little error as possible. This is referred to as *tracking accuracy*. In quantifying the error, the three characteristics measured are the maximum, average, and standard deviation of the absolute error. To measure this, the actual and commanded positions are recorded and analyzed with respect to the XY-plane, the Z-axis, and the combined total error in the XYZ coordinate system. Let the error in the XY-plane be defined by Equation 8.

\[ e_{XY} = \sqrt{[x_c(t) - x(t)]^2 + [y_c(t) - y(t)]^2} \]  

where the \( c \) subscript is used to denote the commanded position and no subscript is used to denote the actual position of the aircraft. The Z direction tracking error is defined as:

\[ e_Z(t) = |z_c(t) - z(t)| \]

and finally, the total XYZ tracking error can be defined as:

\[ e_{XYZ}(t) = \sqrt{[x_c(t) - x(t)]^2 + [y_c(t) - y(t)]^2 + [z_c(t) - z(t)]^2} \]

Using the error terms defined above, nine trajectory tracking metrics can be defined as follows:

- Average absolute tracking error in the XY plane:
  \[ \bar{e}_{XY} = \text{mean}(e_{XY}(t)) \]

- Maximum absolute tracking error in the XY plane:
  \[ e_{XY_{\text{max}}} = \text{max}(e_{XY}(t)) \]

- Standard deviation of the absolute tracking error in the XY plane:
\[ \dot{e}_{XY} = STD(e_{XY}(t)) \]

These same three metrics are calculated for vertical and combined Euclidian distance errors, defined by the Z and XYZ subscripts respectively. Using the above defined trajectory tracking metrics, a trajectory tracking specific performance vector can be defined as:

\[ PV_{TT} = [t_i | i = 1, 2, ..., 9] = [e_{XY_{max}} e_{Z_{max}} e_{XYZ_{max}} \bar{e}_{XY} \bar{e}_{Z} \bar{e}_{XYZ} \dot{e}_{XY} \dot{e}_{Z} \dot{e}_{XYZ}]^T \]

It is important to note that while using this evaluation scheme, trajectory tracking is dependent on time; thus even if an aircraft perfectly follows a path, it will still be penalized if it does not adhere to the commanded timing schedule.

The second goal for the trajectory tracking controllers is to generate commands which the aircraft is capable of following. As part of this constraint, the trajectory tracking controller should supply actuator commands that are gradual and do not cause the control surfaces to saturate for extended periods of time. As a measure of the gradualness of the commanded actuator signals, the integral of the absolute value of the rate of change of the signal is analyzed. The control saturation is measured as a percentage of time steps in which a given control surface is at the maximum extent of its operational range. This portion of the metric is referred to as control activity.

Let \( \delta_x \) be the commanded deflections to a given surface where \( x \) can be \( e, a, r, \) or \( t \) meaning elevator, aileron, rudder, or throttle respectively. The following parameters can be defined as a measure of the control activity quality:

- Integral of surface deflection rate of change:
  \[ I_{\delta_x} = \frac{1}{T} \int_0^T |\delta'_x(t)| dt \]

- Surface saturation index:
  \[ S_{\delta_x} = \frac{100}{T} \int_0^T \delta_x(t) dt \]

where:
\[ \delta_x(t) = \begin{cases} 
0, & \text{if } \delta_x > \delta_{x_{\text{min}}} \land \delta_x < \delta_{x_{\text{max}}} \\
1, & \text{if } \delta_x \leq \delta_{x_{\text{min}}} \lor \delta_x \geq \delta_{x_{\text{max}}} 
\end{cases} \]

Using the above defined control activity metrics, a control activity specific performance vector can be defined as:

\[ PV_{CA} = [c a_i | i = 1, 2, \ldots, 8] = [I \delta_e I \delta_a I \delta_r I \delta_t S \delta_e S a S \delta_r S \delta_t]^T \]

The trajectory tracking accuracy and control activity performance vectors are combined into a simplified and meaningful metric. To accomplish this, a total performance index (PI) is defined as a weighted sum of the above described metrics. In order to achieve this, each of the components must be normalized.

A trajectory tracking PI can be calculated as a weighted average of the trajectory tracking performance vector. Each component is normalized from 0 to 1; with a value of 0 corresponding to poor performance below a heuristically determined threshold, and a value of 1 corresponding to perfect performance. Weights are assigned to each of the parameters based upon the subjective relative importance of each metric. The normalized performance vectors are referred to as \( \overline{PV}_{TT} \) for trajectory tracking and \( \overline{PV}_{CA} \) for control activity. The trajectory tracking accuracy PI is defined as:

\[ PI_{TT} = w_{TT}\overline{PV}_{TT} \]

where \( w_{TT} \) is a 9-element row vector containing the normalization weights corresponding to each element in the trajectory tracking accuracy performance vector.

A control activity PI can also be calculated. In order to accomplish this, a minor modification is needed; the control activity PI is multiplied by the percentage of trajectory points, \( P \), which are within a distance threshold of the commanded path. This is required because the control activity PI loses significance if the goal of following a commanded path is not achieved, i.e. a perfect control activity PI is possible by simply not activating any surfaces, but the aircraft would fall out of the air and the trajectory would not be tracked. The control activity PI is defined as:
\[ P_{I_{CA}} = w_{CA} P \]

where \( w_{CA} \) is an 8-element row vector containing the normalization weights corresponding to each element in the control activity performance vector.

Finally, a global trajectory tracking PI can be defined as a weighted sum of the trajectory tracking PI and the control activity PI:

\[ P_{I_{TOTAL}} = \bar{w}_{TT} P_{I_{TT}} + \bar{w}_{CA} P_{I_{CA}} \]

where \( \bar{w}_{TT} \) and \( \bar{w}_{CA} \) are the heuristic desirability weights for the individual PI components.

This set of performance metrics is used for all of the aircraft models in the simulation environment. The weights are held constant for all configurations to ensure that unbiased comparisons are obtained.

One should note that there are certain benefits and limitations that accompany such a generalized performance index formulation. In many similar studies, more simplified metrics such as average Euclidean distance from the path or control surface saturation percent are used. While such metrics are generally indicative of the performance quality of a given algorithm, they do not provide a very complete assessment of the performance. By incorporating a larger number of metrics into the performance index and assigning meaningful weights for the mission at hand, specific strengths or inadequacies of a given controller can more efficiently be assessed by the designer. The total composite index is typically used in quantifying which controller is better for purposes such as genetic algorithm optimization, but during the design process, all components are taken into account.

Another aspect to note is that the weights are subjective. This is done intentionally as the very definition of what constitutes a better controller may vary significantly depending on the mission at hand. Take for instance the example of military UAVs; hypothetical missions such as bombing passes or aerial refueling have stringent
trajectory accuracy constraints, whereas a long-term surveillance mission would likely want to minimize energy and thus control effort. Additionally, the general framework allows for the flexibility to add or remove metrics as desired. In this research effort, the weights and normalization cutoffs that have been chosen highlight and amplify trajectory tracking accuracy and control effort differences between controllers. As will be shown, and is logical as the metrics are incorporated into the composite index, higher indices correspond to visibly lower error and control activity. Such traditional metrics will also be analyzed in the results section of this document.

Finally, it is important to point out that this metric does not inherently incorporate a measure of robustness. As mentioned earlier, increasing performance is typically associated with decreasing robustness since the performance increases obtained are often the resultant of the inflation of control gains. For the purpose of fault-tolerant control, this is certainly an area of concern. A logical extension to the composite performance index would be to introduce a robustness index and assess a score based on applicable metrics. However, such measures of robustness would either need to use data from multiple flight tests featuring failures, which introduces more subjectivity due to the choice of failures conditions, or the robustness index would have to rely on traditional robust control metrics such as gain and phase margins, which are not always readily available due to the highly non-linear characteristics of aircraft control. Consequently, for the purpose of controller optimization, a modified fitness function incorporating failure condition simulations is introduced to mitigate this issue. This is discussed in more detail in Chapter 6.

4.3 Batch Simulations

When analyzing the performance of the various trajectory tracking controllers or trajectory planners, it is often necessary to run and compile results for numerous different flight scenarios. An efficient method for combining various combinations of
trajectories, abnormal conditions, and failures into one comparison is needed. For this purpose, a batch script, with accompanying GUI, was built into the WVU UAV v3 simulation environment.

This batch process allows for various permutations of trials to be performed based upon the needs of a given researcher. Through use of the GUI shown below in Figure 4-10, the user can specify the aircraft model to be used, the desired trajectory planning methods or pre-recorded trajectories, the trajectory tracking controllers, and any failures or abnormal conditions which may be present. This script then executes the simulation for each possible combination of scenarios. After each simulation, a file is saved with all of the data required to calculate the performance indices described above in Section 4.2. Prefixes for each of the elements, which are defined in the GUI, are concatenated into a single filename which is used to distinguish a particular scenario. The files for a given batch simulation are saved in a common directory. After all of the simulations have been executed, a script is used to compute the various performance metrics for each of the flight data files in the directory. Finally, an Excel worksheet is assembled containing the results which include the filename identifier, the trajectory tracking accuracy performance vector, the control activity performance vector, and the individual trajectory tracking accuracy, control activity, and total combined performance indices. These Excel worksheets are used to create the various performance comparison plots which are featured throughout this document.
In addition to the batch script for calculating performance results, the individual script for calculating performance indices is accessible directly from the simulation environment by opening the *Performance Indices* block. This module calculates the performance indices, sends verbose output pertaining to the individual performance metrics and indices, and displays a bar plot of the performance indices components. An example performance indices plot can be seen in Figure 4-11.
Figure 4-11. Performance indices display
5 Trajectory Tracking Controllers

5.1 Fixed-Parameter Controllers

Several trajectory tracking controller architectures have been implemented in the WVU UAV simulation environment in previous work. The adaptive control techniques being developed in this research are intended to augment existing fixed-parameter, non-adaptive controllers. As such, these existing controllers will serve as the baseline for the AIS-based adaptive control techniques being developed. For this reason, an overview of the existing fixed-parameter controllers is provided in this section. Included in the section will be discussions of the Position PID, Outer-Loop NLDI, Extended NLDI, and LQR with Integral Action algorithms. These controllers are compared to establish a baseline controller performance and select a suitable control architecture to serve as this innate immune response controller in the formulation of the IMRAC.

5.1.1 Position PID

The position PID controller uses linear control laws, namely PID, to minimize the forward, lateral, and vertical distances from a reference trajectory. As such, the algorithm is separated into three distinct controllers: a forward distance controller, a lateral distance controller, and a vertical distance controller.

The position PID controller consists of three stages. The first stage calculates trajectory tracking error relative to a virtual reference aircraft in terms of forward, lateral, and vertical distance. This formation geometry can be seen below in Figure 5-1. In previous work this was used for formation flight of multiple aircraft using non-zero commanded offset distances from the leader to determine the trajectory of the followers\cite{6, 92, 93}. In this controller, the aircraft follows a virtual leader with a zero following-distance.
Figure 5-1. Trajectory tracking formation geometry

The first stage of the algorithm produces a set of geometric trajectory tracking errors. Horizontal-plane errors in the forward direction and lateral direction are defined as $f$ and $l$ respectively, and vertical-plane error is defined as $h$. These errors and their time derivatives are used by an outer-loop controller to produce bank angle, pitch angle, and throttle commands. The desired bank angle command is obtained using the lateral control module described in Equation 22, the throttle command is calculated using the forward control module described in Equation 23, and the desired pitch angle command is obtained by using the vertical controller described in Equation 24. It is important to note that the throttle command is directly controlled by the outer-loop controller, and the other surfaces are controlled by the inner-loop controller; the inner-loop controller serves as a pass-through for the throttle command signal.

$$\phi_d = K_l \dot{l} + K_l l$$  \hspace{1cm} 22
$$\delta_T = K_f \dot{f} + K_f f$$  \hspace{1cm} 23
$$\theta_d = K_z \dot{z} + K_z z$$  \hspace{1cm} 24

The bank angle and pitch angle commanded by the outer-loop controller are achieved by using an inner-loop controller to produce aileron, rudder, and elevator commands. The lateral controller commands for the ailerons and rudder are generated using Equations 25 and 26.
The elevator commands are calculated using Equation 27.

\[
\delta_e = K_q q + K_\theta (\theta - \theta_d)
\]  

5.1.2 **OUTER-LOOP NLDI**

As described in Campa et al.\[35\], the structure of the position PID algorithm was modified to incorporate NLDI-based control of the outer-loop. The inner-loop controller remains unchanged from the position PID controller presented above. The outer-loop NLDI controller uses Equations 28 and 29 to calculate the desired bank angle and throttle command. Equation 24 above is still used to calculate the desired pitch angle\[35\].

\[
\phi_d = \arctan\left\{\frac{1}{g \cos \gamma} \left[ l_d \cos(\chi - \chi_V) + \dot{f}_d \sin(\chi - \chi_V) \right] + \frac{V}{g} \Omega_V \right. \\
+ \left. \left[ l \sin(\chi - \chi_V) - \dot{f} \cos(\chi - \chi_V) \frac{ \Omega_V }{ g \cos \gamma } \right] \right\}
\]  

Figure 5-2. Overview of the position PID architecture
\[
\delta_T = \frac{m}{K_T \cos \gamma} \left[ \ddot{l}_d \sin (\chi - \chi_V) - \dot{j}_d \cos (\chi - \chi_V) \right]
+ \frac{1}{K_T} \left( \frac{1}{2} \rho_0 V^2 S (C_D_0 + C_D a_0) + mg \sin \gamma - T_0 \right)
- \frac{m}{K_T \cos \gamma} \Omega_V \left[ \dot{l} \cos (\chi - \chi_V) + \dot{f} \cos (\chi - \chi_V) \right]
\]

where:

\[
\cos (\chi_V) = \frac{V_{V_x}}{\sqrt{V_{V_x}^2 + V_{V_y}^2}}
\]

\[
\sin (\chi_V) = \frac{V_{V_y}}{\sqrt{V_{V_x}^2 + V_{V_y}^2}}
\]

and where the lateral and forward acceleration terms are estimated by linear compensator-style control laws which are given by:

\[
\ddot{l}_d = -K_{ls} \dot{l} - K_l l
\]

\[
\ddot{j}_d = -K_{fs} \dot{f} - K_f f
\]

### 5.1.3 Extended NLDI

The previous method used NLDI-based control for the outer-loop while linear controllers were used in the inner-loop. In this method, as described in more detail by Moncayo et al., the inner-loop is replaced, or extended, with an NLDI-based controller. The inner-loop NLDI controller is divided into two sub-controllers referred to as the slow mode and the fast mode. This structure can be seen below in Figure 5-3.
In the first stage, the slow mode, pitch, roll, and yaw commands are calculated using Equation 34\textsuperscript{[36]}.

\[
\begin{bmatrix}
p_d \\
q_d \\
r_d
\end{bmatrix} =
\begin{bmatrix}
1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\
0 & \cos \phi & -\sin \phi \\
0 & \sin \phi \sec \theta & \cos \theta \sec \theta
\end{bmatrix}^{-1}
\begin{bmatrix}
U_\phi \\
U_\theta \\
U_\psi
\end{bmatrix}
\]

where:

\[
\begin{bmatrix}
U_\phi \\
U_\theta \\
U_\psi
\end{bmatrix} =
\begin{bmatrix}
K_\phi (\phi_d - \phi) \\
K_\theta (\theta_d - \theta) \\
K_\psi (\psi_d - \psi)
\end{bmatrix}
\]

The second stage, the fast mode, uses the angular rate commands produced by the slow mode to provide control surface commands for the elevators, ailerons, and rudders. The angular acceleration compensator is defined by Equation 36.

\[
\begin{bmatrix}
U_p \\
U_q \\
U_r
\end{bmatrix} =
\begin{bmatrix}
K_p (p_d - p) \\
K_q (q_d - q) \\
K_r (r_d - r)
\end{bmatrix}
\]

The surface deflections are calculated by Equations 37, 38, and 39\textsuperscript{[36]}.

\[
\delta_e = \frac{C_m(x, \delta) - C_{m_0} - C_{m_\alpha} \alpha - \frac{\bar{c}}{2V} C_{mq} q}{C_m \delta_e} = \frac{M_{Ad}}{q \delta_e} - \frac{C_{m_0} - C_{m_\alpha} \alpha - \frac{\bar{c}}{2V} C_{mq} q}{C_m \delta_e}
\]

\[
\delta_a = \frac{C_{i_\delta \gamma} b_2 - C_{n_\delta \gamma} b_1}{C_{i_\delta \gamma} C_{n_\delta a} - C_{n_\delta \gamma} C_{i_\delta a}}
\]

\[
\delta_r = \frac{C_{n_\delta a} b_1 - C_{i_\delta a} b_2}{C_{i_\delta \gamma} C_{n_\delta a} - C_{n_\delta \gamma} C_{i_\delta a}}
\]

where:
\[ b_1 = C_t(x, \delta) - C_{l0} - C_{l\beta} \beta - \frac{b}{2V} (C_{l p} p + C_{l r} r) \]
\[ = \frac{L_{Ad}}{qSb} - C_{l0} - C_{l\beta} \beta - \frac{b}{2V} (C_{l p} p + C_{l r} r) \]  \hspace{1cm} (40)

\[ b_2 = C_n(x, \delta) - C_{n0} - C_{n\beta} \beta - \frac{b}{2V} (C_{n p} p + C_{n r} r) \]
\[ = \frac{N_{Ad}}{qSb} - C_{n0} - C_{n\beta} \beta - \frac{b}{2V} (C_{n p} p + C_{n r} r) \]  \hspace{1cm} (41)

5.1.4 LQR with Integral Action

Pole-placement is a common state-space linear controller design technique that is used to \textit{arbitrarily} relocate the poles of a given system using state feedback. With this technique, the transient response of a system can be altered to resemble that of a system with the desired poles. Note that this method does not alter the locations of the zeros, and thus the response may not identically match the desired response.

For this approach, let the system be described by Equation 42.
\[ \dot{x} = Ax + Bu \]
\[ y = Cx + Du \]  \hspace{1cm} (42)

where \( x \in \mathbb{R}^n \) is the system state vector, \( y \in \mathbb{R}^m \) is the system output vector, \( u \in \mathbb{R}^m \) is the control vector, \( A \in \mathbb{R}^{n\times n} \) is the state matrix, \( B \in \mathbb{R}^{n\times m} \) is the input matrix, \( C \in \mathbb{R}^{q\times n} \) is the output matrix, and \( D \in \mathbb{R}^{q\times m} \) is the feedthrough matrix. It is often the case that the feedthrough matrix is equal to zero; this is the case in the control schemes presented in this document.

If the system is completely controllable, and the system state is fully measurable and available for feedback, then state feedback can be used to force the closed-loop system response to desired pole positions. Let the state feedback controller be given by Equation 43.
\[ u = -Kx \]  \hspace{1cm} (43)
where \( \mathbf{K} \in \mathbb{R}^{m \times n} \) is the state feedback gain matrix which must be determined. The block diagram for this configuration is given in Figure 5-4.

![Block diagram](diagram.png)

**Figure 5-4. State-feedback architecture**

With state feedback, the transient response and the stability of the system is yielded by the closed-loop of the system which is described by Equation 44.

\[
\dot{x} = (A - BK)x
\]  

44

The problem of pole-placement via state feedback thus reduces to calculating an appropriate gain matrix, \( \mathbf{K} \), that makes the poles of Equation 42 above match the desired poles. There are several different methods that can be used to accomplish this. For simple systems, it is often possible to simply solve for \( \mathbf{K} \) algebraically with direct substitution. As described in Ogata\(^{[40]}\), for more complicated systems, matrix transformations or Ackerman’s formula can be used to solve for \( \mathbf{K} \).

This configuration represents a regulator system since the input to the system is zero; when perturbed, the system will return to zero with the transient response dictated by the chosen poles. When it is desired to track a given reference signal, \( \mathbf{r}(t) \in \mathbb{R}^m \), this architecture must be altered. This is referred to as a servo system. To make the controller track the given reference input, an output error comparator and a feedforward gain are introduced. This term can be either a proportional or an integral
gain. In the controller implemented on the UAV, an integral gain will be introduced. This architecture can be seen in Figure 5-5.

The desired poles for the controller can be chosen arbitrarily. However, with higher order systems and in systems with dominant zeros, choosing poles that will give the desired transient response can be a difficult task. For this reason, LQR was developed. LQR is an optimal control technique that aims to minimize control effort. In effect, it is a more systematic approach to designing the transient response of the system.

To introduce the integral gain, an error term is calculated based on the system output and the reference input as given in Equation 45 below.

$$\dot{e}_y(t) = r(t) - y(t)$$  \hspace{1cm} \text{45}

The control input for the system is updated to include the integral term. This is given in Equation 46.

$$u = -K_P x + K_I e_y$$  \hspace{1cm} \text{46}

The open-loop dynamics for this system take the form given in Equation 47.
\[
\begin{bmatrix}
\dot{e}_y \\
\dot{x}
\end{bmatrix}
= \begin{bmatrix}
0_{m \times m} & C \\
0_{n \times m} & A
\end{bmatrix}
\begin{bmatrix}
e_y \\
x
\end{bmatrix}
+ \begin{bmatrix}
0_{m \times m} & B \\
0_{n \times m}
\end{bmatrix}
u
+ \begin{bmatrix}
-I_{m \times m} \\
0_{n \times m}
\end{bmatrix}r(t)
\]

\[
y = \begin{bmatrix}
0_{m \times m} & C
\end{bmatrix}
\begin{bmatrix}
e_y \\
x
\end{bmatrix}
\]

Thus \( \bar{x} \in \mathbb{R}^n \) is the new state vector, \( \bar{A} \in \mathbb{R}^{n \times n} \) is the new state matrix, \( \bar{B} \in \mathbb{R}^{n \times m} \) is the new input matrix, and \( \bar{C} \in \mathbb{R}^{q \times n} \). This can be rewritten as:

\[
\dot{x} = \bar{A}x + \bar{B}u + B_{ref}r(t)
\]

\[
y = \bar{C}x
\]

This can be further simplified by defining:

\[
\dot{z} = \bar{A}z + \bar{B}v
\]

where:

\[
z = \bar{x} \text{ and } v = \dot{u}
\]

From this, the LQR quadratic cost function is given by Equation 48 below.

\[
J = \int_0^\infty (z^T Qz + v^T Rv) \, dt
\]

where \( Q \in \mathbb{R}^{n \times n} \) and \( R \in \mathbb{R}^{m \times m} \) are positive-definite Hermitian matrices which are used to shape the performance of the controller. The relative importance of error and control expenditure are given by \( Q \) and \( R \) respectively.

The control input for this system can also be updated with respect to the transformation given above in Equations 48 and 49 as follows:

\[
v = \dot{u} = -R^{-1}B^TPz
\]

where \( K_x^T \in \mathbb{R}^{m \times n} \) is the LQR gain matrix and \( P \in \mathbb{R}^{n \times n} \) is the unique, positive-definite solution to the algebraic Riccati equation:

\[
PA + A^TP - PBR^{-1}B^TP + Q = 0
\]

Finally, the LQR gain matrix can be expressed in terms the proportional (the original state feedback gain) and the integral gain:
\[ u = -K^T \dot{x} = -K_p x - K_f e_y \]

One should note that LQR gives an optimal solution to the control problem with respect to the choice of \( Q \) and \( R \); thus, these two matrices must still be tuned to produce an acceptable response. The benefit of this method is that the tuning of \( Q \) and \( R \) are much more intuitive than arbitrarily choosing desired pole locations.

For the purpose of UAV control, two different LQR controllers, one for longitudinal movement and one for lateral movement, are used to control aircraft angular positions within the inner-loop trajectory-tracking controller. The outer-loop trajectory-tracking controller does not use LQR since it is responsible for the kinematics and an appropriate linearized model is not applicable. Consequently either of the aforementioned outer-loop controllers, PID-based or NLDI-based, can be coupled with the LQR-base inner-loop control.

LQR is a linear control technique, but the dynamics of the UAV are inherently nonlinear. Consequentially, the nonlinear UAV can be linearized around trim conditions for its longitudinal and lateral channels, and as long as the aircraft remains close to these operating conditions, satisfactory performance can be obtained by the LQR controller.

The linearized longitudinal model for the trim conditions \( \alpha = \theta = 3^\circ \) and \( V = 40 m/s \) for the YF-22 is given as follows:

\[
\begin{bmatrix}
\dot{v} \\
\dot{\alpha} \\
\dot{q} \\
\dot{\theta}
\end{bmatrix}
= \begin{bmatrix}
-0.2368 & 9.5576 & 0 & -9.8057 \\
-0.0119 & -4.1172 & 0.7781 & 0 \\
0 & -33.883 & -3.5729 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
v \\
\alpha \\
q \\
\theta
\end{bmatrix}
+ \begin{bmatrix}
1.43 \times 10^{-12} \\
-0.5435 \\
39.0843 \\
0
\end{bmatrix}
\delta_e
\]

and the linearized lateral model for the YF-22 is given as follows:
\[
\begin{bmatrix}
\dot{\beta} \\
\dot{p} \\
\dot{r} \\
\dot{\phi}
\end{bmatrix} =
\begin{bmatrix}
0.4299 & 0.0938 & -1.0300 & 0.2448 \\
-67.333 & -7.9484 & 5.6402 & 0 \\
20.533 & -0.6533 & -1.9955 & 0 \\
0 & 1 & 0.0524 & 0
\end{bmatrix}
\begin{bmatrix}
\beta \\
p \\
r \\
\phi
\end{bmatrix}
\]

\[
\begin{bmatrix}
-0.1363 & 0.7713 \\
50.9218 & -33.4734 \\
3.1304 & 24.3624 \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
\delta_a \\
\delta_r
\end{bmatrix}
\]

For the longitudinal channel of the LQR:
\[
r(t) = \theta_{cmd}(t)
\]
and for the lateral channel of the LQR:
\[
r(t) = [\beta_{cmd}(t) \phi_{cmd}(t)]^T
\]

As will be discussed later, the LQR controller is used as the baseline part of the controller for the AIS-based MRAC control architecture.

5.2 Adaptive Control

Several different adaptive control architectures were investigated in this research effort. The goal of this project was to improve upon the AIS methodology to provide a comprehensive solution to aircraft fault-tolerance. As such, the first three fixed-parameter controllers discussed above were augmented with the AIS suppression mechanism discussed in Section 3.2. As will be shown in Chapter 7, the performance of this initial control architecture was found to be promising although somewhat limited due to the simplified input/output relationships considered for most of its components. For this reason, a comprehensive survey of more traditional adaptive control techniques was performed in order to investigate alternative component input/output relationships. During this study, certain parallels were observed between the AIS architecture and that of $\mathcal{L}_1$ adaptive control. For this reason, the position PID controller was augmented with the $\mathcal{L}_1$ to evaluate its fault tolerant capabilities. As will be discussed in Chapter 7, the $\mathcal{L}_1$ adaptive control architecture shows significant and
promising fault-tolerant capabilities. However, as discussed in Section 2.2.5, there is much room for improvement within this architecture. For this reason, inspiration will be taken from the method and used to build a better algorithm, herein referred to as IMRAC.

5.2.1 AIS Suppression Mechanism

The AIS suppression mechanism, as formulated in Takahashi et al.\[82\], was used to augment the position PID, the outer-loop NLDI, and the extended NLDI controllers. As will be shown in Chapter 7, the performance of this preliminary control architecture was found to be promising although somewhat limited due to the simplified input/output relationships considered for most of its components. In practice, the AIS suppression mechanism analogy can be applied to any PID type controller using the following relationship:

\[
    u(k) = k_p \left(1 - \eta f(\Delta u(k))\right) \left(1 + \frac{k_i}{z - 1} + k_d \frac{z - 1}{z}\right) e(k)
\]

where:

\[
    f(\Delta u(k)) = 1 - \exp\left(-\frac{(\Delta u(k))^2}{a}\right)
\]

where \(a\) is a constant that changes the shape of the activation function and \(\Delta u(k)\) represents the change in the command over some discrete time delay.

In the position PID architecture, the inner-loop controllers that command pitch angle, \(\theta\), and bank angle, \(\phi\), were augmented with gains that are adaptive by the AIS mechanism. The position PID and the outer-loop NLDI share the same inner-loop controllers. Consequently, the outer-loop NLDI implements the AIS adaptive mechanism in the same way as the position PID controller. Finally, the extended NLDI controller implements the AIS mechanism in the roll rate, pitch rate, and yaw rate PID channels of the inner-loop fast-mode controller.
5.2.2 $\mathcal{L}_1$

The position PID controller was augmented with $\mathcal{L}_1$ adaptive control. Essentially, an $\mathcal{L}_1$ control loop can be used to replace any control loop. For the $\mathcal{L}_1$ augmentation of the position PID controller, two separate $\mathcal{L}_1$ controllers were added; one was used to control the pitch angle $\theta$ and the other one to control the bank angle $\phi$ of the aircraft using the control surfaces. As summarized from Kaminer et al.[94], $\mathcal{L}_1$ adaptive control is achieved using three distinct modules: a state predictor, an adaptive law, and a control law.

The state predictor is given by Equation 61, where $A_{m\theta}, b_{m\theta},$ and $c_{m\theta}$ represent the canonical representation of the reference model $M(s)$. These equations are given in terms of the $\theta$ controller, but the formulation is the same for the $\phi$ controller as well.

$$\dot{x}_\theta = A_{m\theta} x_\theta(t) + b_{m\theta} \theta_{ad}(t) + \delta\theta(t)$$

$$\dot{\theta}(t) = c_{m\theta}^T \dot{x}_\theta(t)$$

The adaptive law is based around Lyapunov’s stability criteria. From Lyapunov’s algebraic equation:

$$A_{m\theta}^T P_\theta + P_\theta A_{m\theta} = -Q_\theta$$

where $P_\theta = P^T_\theta > 0$, $Q_\theta = Q^T_\theta > 0$, and:

$$P_\theta = \sqrt{P_\theta^T \sqrt{P_\theta}}$$

Let $D_\theta$ be defined as the null space of the $c_{m\theta}^T (\sqrt{P_\theta})^{-1}$

$$D_\theta \left( c_{m\theta}^T (\sqrt{P_\theta})^{-1} \right)^T = 0$$

From this, define $\Lambda_\theta$ as:

$$\Lambda_\theta = \left[ \begin{array}{c}
 c_{m\theta}^T \\
 D_\theta \sqrt{P_\theta}
 \end{array} \right]$$
Finally, Equations 66 through 69 give the adaptive law. $T_s$ is the discrete sample time and $i = 1, 2, 3, ...$ is the sample number.

$$\Phi_\theta(T_s) = \int_0^{T_s} e^{A_\theta A_m \theta A_\theta^{-1}(T_s - \tau)} \Lambda_\theta d\tau$$

$$\mu_\theta(iT_s) = e^{A_\theta A_m \theta A_\theta^{-1}T_s} A_\theta(iT_s)$$

$$\hat{\sigma}_\theta(t) = \hat{\sigma}_\theta(iT_s), \ t \in [iT_s, (i + 1)T_s)$$

$$\hat{\sigma}_\theta(iT_s) = -\Phi_\theta^{-1}(T_s) \mu_\theta(iT_s)$$

The control law is given by Equation 70 below where $\mathcal{C}_\theta(s)$ is the Laplace transform of the above described low-pass filter. Stability proofs and design criteria are given by Kaminer et al.\cite{94} and Cao et al.\cite{66, 95, 96}

$$\theta_{ad}(s) = \mathcal{C}_\theta(s) r_\theta(s) - \frac{\mathcal{C}_\theta(s)}{\mathcal{M}_\theta(s)} c_m \tau (sI - A_m)^{-1} \sigma_\theta(s)$$

The above equation is used to generate the control law for a given $\mathcal{L}_1$ adaptive control implementation. Note that this serves as the entire control law; no other components such as a traditional PID controller, are needed with this approach.

### 5.2.3 MRAC

As introduced in Section 2.2.2, MRAC is an adaptive control technique that aims to force a system response to behave like that of a reference model. Many modifications have been made to the basic MRAC architecture that was developed in the 1960s. Such modifications have been introduced to address robustness and transient performance issues associated with early variations of MRAC, as well as for allowing for rigorous stability and boundedness proofs. The MRAC structure developed in this research effort combines the LQR with integral action with several historical modifications that have been shown to positively improve the performance and robustness of traditional MRAC\cite{37, 97-101}. A more thorough explanation of the derivations and reasoning used in developing this method can be found in part in Lavretsky and Wise\cite{37}.
The derivation of MRAC begins with the introduction of uncertainty to the state-space plant model. Such uncertainty can be the result of parametric modeling errors, external disturbances, and failures. Thus, a system can be defined as

\[
\begin{align*}
\dot{x} &= Ax + BL(u + f(x)) \\
y &= Cx + DL(u + f(x))
\end{align*}
\]

where \(x \in \mathbb{R}^n\) is the system state vector, \(y \in \mathbb{R}^m\) is the system output vector, \(u \in \mathbb{R}^m\) is the control vector, \(A \in \mathbb{R}^{n \times n}\) is the state matrix, \(B \in \mathbb{R}^{n \times m}\) is the input matrix, \(C \in \mathbb{R}^{q \times n}\) is the output matrix, \(D \in \mathbb{R}^{q \times m}\) is the feedthrough matrix, \(L \in \mathbb{R}^{m \times m}\) is an unknown diagonal matrix with strictly positive elements \(\lambda_i\), and \(f(x)\) is an unknown nonlinear function representing the matched uncertainty in the system. As before with the LQR methodology, the feedthrough matrix for the system in question is negligible.

Uncertainties of this nature are referred to as matched since they enter the system through the control input. In theory, uncertainties of this nature do not affect the controllability of the system so long as \(L\) is invertible. Consequently, an appropriate controller can cancel out the uncertainties\(^{[37, 102]}\). In general, MRAC is designed to be used with uncertainties that are constant in nature; however, they are often used successfully with time-varying or stochastic uncertainty. In contrast, unmatched uncertainties cannot be directly eliminated by control action. There are methods for making systems more robust to unmatched uncertainties. However, these are considered out of the scope of this research.

The unknown nonlinear function listed above in Equation 71 can be rewritten as:

\[
f(x) = \Theta^T \Phi(x)
\]

where \(\Theta \in \mathbb{R}^{N \times m}\) is a constant matrix with unknown coefficients and \(\Phi(x) \in \mathbb{R}^N\) is a known regressor vector with \(N\) locally Lipschitz-continuous basis functions\(^{[37]}\), \(\phi_i\). Using this relationship, Equation 71 can be rewritten as

\[
\dot{x} = Ax + BL(u + \Theta^T \Phi(x))
\]
\[ y = Cx + D\Lambda(u + \Theta^T\Phi(x)) \]

It is desired that the controller tracks a given bounded reference signal \( r(t) \in \mathbb{R}^m \). The error for such a system can be defined as
\[ e_y(t) = y(t) - r(t) \]
and its integral can be defined as
\[ e_{yI}(t) = \int_0^t e_y(\tau)d\tau \iff \dot{e}_{yI}(t) = e_y(t) \]

From here, the above-described uncertainty is introduced into the LQR with integrator structure given above in Equation 47:
\[
\begin{bmatrix} \dot{e}_{yI} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0_{m \times m} & C \\ 0_{n \times m} & A \end{bmatrix} \begin{bmatrix} e_{yI} \\ x \end{bmatrix} + \begin{bmatrix} 0_{m \times m} \\ B \end{bmatrix} (u + \Theta^T\Phi(x)) + \begin{bmatrix} -I_{m \times m} \\ 0_{n \times m} \end{bmatrix} r(t)
\]

or
\[ \dot{x} = \bar{A}x + \bar{B}A(u + \Theta^T\Phi(x)) + B_{ref}r(t) \]

Equally, the output of the system can be expressed as follows:
\[ \hat{y} = \begin{bmatrix} 0_{m \times m} \\ C \end{bmatrix} \begin{bmatrix} e_{yI} \\ x \end{bmatrix} + D\Lambda(u + \Theta^T\Phi(x)) \]

or
\[ \hat{y} = \bar{C}x + Du \]

A baseline controller can be constructed to satisfy the reference signal if no uncertainty is present; this is the case of the traditional LQR with integral control as described in Section 5.1.4. For convenience, the baseline controller is reiterated below:

In order to develop the baseline controller, the uncertainty in the system is neglected and the open-loop dynamics can be expressed as
\[ \dot{x} = \bar{A}x + \bar{B}u + B_{ref}r(t) \]
\[ \hat{y} = \bar{C}x + Du \]

The system model is transformed as

65
where:
\[ z = \dot{x} \text{ and } v = \dot{u} \]

The LQR quadratic cost function is defined as
\[ J = \int_0^\infty (z^T Q z + v^T R v) \, dt \]

where \( Q \in \mathbb{R}^{n \times n} \) and \( R \in \mathbb{R}^{m \times m} \) are positive-definite Hermitian matrices that are used to design the performance of the controller. The relative importance of tracking error and control expenditure are tuned by the choice of \( Q \) and \( R \) respectively.

The baseline LQR servomechanism controller for this system is given as
\[ v = -R^{-1} B^T P z \quad \Leftrightarrow \quad \dot{u} = -R^{-1} B^T P \dot{x} \]

where \( K_x^T \in \mathbb{R}^{m \times n} \) is the LQR gain matrix and \( P \in \mathbb{R}^{n \times n} \) is the unique, positive-definite solution to the algebraic Riccati equation:
\[ P \bar{A} + \bar{A}^T P - P \bar{B} R^{-1} \bar{B}^T P + Q = 0 \]

The controller given in Equation 84 is integrated to give the resulting baseline controller in terms of the proportional and integral gains.

Finally, the LQR gain matrix can be expressed in terms of the proportional (the original state feedback gain) and the integral gain:
\[ u_{bl} = u = -K_x^T \ddot{x} = -K_I e_y - K_P x \]

where
\[ K_x^T = [K_I \quad K_P] \]

The LQR-based, baseline controller described above provides an optimal solution for ideal systems without uncertainty. However, the uncertainty present in real-world systems may cause the performance of such a system to degrade. For this reason, an adaptive component is added to the controller to help cancel the uncertainty described
in the system model given above in Equation 73. The controller with the adaptive component is given as

\[ u = u_{bl} + u_{ad} = -K_x^T\bar{x} + u_{ad} = -K_ie_{y_l} - K_p\bar{x} + u_{ad} \quad \text{88} \]

It is desired that the controller cause the system to respond like a given reference model. The desired reference model dynamics are given by the equation below.

\[ \dot{x}_{ref} = A_{ref}x_{ref} + B_{ref}r(t) \quad \text{89} \]
\[ y_{ref} = C_{ref}x_{ref} \]

where

\[ A_{ref} = \bar{A} - \bar{B}K_x^T \quad \text{90} \]

and

\[ C_{ref} = \bar{C} - DK_x^T \quad \text{91} \]

In general, any model can be used as the reference model. Here, the reference model is specified as the closed-loop system with LQR and integral control. In this way, if the response of the system deviates from that of the expected, nominal plant with LQR, then the adaptive component will be adjusted to bring the system back to the desired response.

The system dynamics can be reformulated by combining Equations 77, 88, 90, and 91:

\[ \dot{x} = A_{ref}\bar{x} + B\Lambda \left( u_{ad} + \frac{\kappa_l^T}{\theta_l^T\Phi(u_{bl},x)} \right) + B_{ref}r(t) \quad \text{92} \]
\[ \dot{y} = C_{ref}\bar{x} + D\Lambda \left( u_{ad} + \frac{\kappa_l^T}{\theta_l^T\Phi(u_{bl},x)} \right) \]

which can be rewritten as:

\[ \dot{x} = A_{ref}\bar{x} + B\Lambda(u_{ad} + \theta_l^T\Phi(u_{bl},x)) + B_{ref}r(t) \quad \text{93} \]
\[ \dot{y} = C_{ref}\bar{x} + D\Lambda(u_{ad} + \theta_l^T\Phi(u_{bl},x)) \]
The unknown coefficients matrix is extended to include baseline controller output

\[ \overline{\Theta}^T = [K_u \ \Theta^T] \]  

and the regressor vector is redefined as

\[ \Phi(u_{bl}, x) = [u_{bl}^T \ \Phi^T(x)]^T \]  

In order to perfectly cancel the undesired uncertainty, \( \Lambda \) and \( \overline{\Theta} \), the adaptive controller must be defined as:

\[ u_{ad} = -\overline{\Theta}^T \Phi(u_{bl}, x) \]  

By substituting Equation 96 back into Equation 93, the original reference model, the LQR closed-loop response, is obtained as in Equation 89.

However, since the exact uncertainty is unknown, an estimate must be used instead:

\[ u_{ad} = -\hat{\overline{\Theta}}^T \Phi(u_{bl}, x) \]  

The estimate of the unknown parameters can be found by choosing the following Lyapunov function given by the adaptive law presented in Equation 98 below. The full derivation of this function can be found in Lavretsky and Wise\cite{37}.

\[ \dot{\hat{\Theta}} = \Gamma_{\hat{\Theta}} \Phi(u_{bl}, x)e^TPB \]  

where \( \Gamma_{\hat{\Theta}} = \Gamma_{\hat{\Theta}}^T > 0 \) is a symmetric positive-definite adaptation gain matrix, and the reference model tracking error, \( e \), is given as

\[ e = x - x_{ref} \]  

and \( P \) is a symmetric positive-definite matrix given as the unique solution to the algebraic Lyapunov equation given in Equation 100 below.

\[ A_{ref}^TP + PA_{ref} = -Q \]  

where \( Q = Q^T > 0 \) is another symmetric positive-definite matrix. The choice of \( \Gamma_{\hat{\Theta}} \) and \( Q \) affect how quickly the MRAC adapts to a given condition.

Using the adaptive law provided above, the total control effort is given as
\[ u = \frac{-K^T x}{u_{pl}} - \hat{\theta}^T \Phi(u_{bl}, x) \]

The regressor vector \( \Phi(u_{bl}, x) \) can incorporate any combination of state variables. In the research presented here, the state vector is used in its entirety for a given channel. The regressor vector is used in conjunction with the unknown coefficients matrix to form a linear combination. The choice of regressor vector should form a sufficient basis such that a maximum number of uncertainties can be represented. The choice of regressor vector has been the focus of several enhancements to the basic MRAC architecture; many of which incorporate neural network features such as radial basis functions (RBFs)\(^{103-105}\).

An overview of the basic MRAC architecture can be found below in Figure 5-6.

An illustration of the adaptive capabilities of MRAC is provided in Figure 5-7 and Figure 5-8 below. In this example, the MRAC with LQR servomechanism controller is used to control a generic second order system (GSOS) with known parametric uncertainty introduced. As seen in Figure 5-7, at first, the actual response deviates from the reference response due to the uncertainty in the system. However, the MRAC scheme learns the parameters required to adapt to the uncertainty and cancels it. In the
end, the response of the system matches that of the desired reference model. As seen in Figure 5-8, the parameter estimates converge to the actual values of the introduced uncertainty. Stability proofs are given in Lavretsky and Wise[37]. They guarantee that the response of the MRAC system with Lyapunov-based adaptation laws is stable and bounded. One should note that while the controller is stable, the uncertainty parameter estimates are not guaranteed to converge. Additionally, as the magnitude of the uncertainty increases, the MRAC algorithm may have more difficulty for it.

Figure 5-7. MRAC tracking performance
The above-described MRAC algorithm is well-equipped to deal with the uncertainty present in a wide variety of systems. However, the technique is not without its drawbacks. First of all, if the assumption of matched uncertainties is broken, then MRAC may not be capable of effectively cancelling the uncertainty. Additionally, there are limitations in how quickly the algorithm can adapt. If the adaptation rate is set too high, it may introduce instability into the system. Conversely, if the adaptation rate is not high enough, the system may not adapt quickly enough to recover from the given disturbances or uncertainties. The transient performance and robustness of the algorithm can be improved by introducing several modifications; the modified MRAC architecture described hereafter is referred to as Robust MRAC, or RMRAC.

5.2.3.1 Dead-Zone Modification

The basic MRAC structure is designed to accommodate for matched uncertainties. When bounded unmatched uncertainties are introduced into a plant, as shown in Equation 102, and as happens with most real-world systems, traditional MRAC often
cannot perfectly cancel all of the uncertainty. Consequently, the controller continues trying to adapt to the unknown parameters, but instead the parameter estimations become unbounded and diverge from the optimal solution. This is referred to as parameter drift, and it is one of the primary complications that may arise in an MRAC application. A thorough derivation of this mechanism is provided in Ioannu and Sun\cite{102} and Lavretsky and Wise\cite{37}.

\[
\dot{x} = Ax + BA(u + \Theta^T \Phi(x)) + \xi(t) \tag{102}
\]

where \(\xi \in \mathbb{R}^n\) is a uniformly bounded, time-dependent disturbance. Note that this disturbance is defined as unmatched since it does not depend on the control input.

There are several different modifications that can be made to the above-described MRAC architecture that can help prevent parameter drift. Such modifications include the dead-zone operator, the \(\sigma\)-modification, and the \(e\)-modification. Of these methods, the simplest and most often used is the dead-zone operator. As such, this technique will be incorporated in the enhanced MRAC algorithm developed in this research.

The dead-zone operator is essentially a piece-wise function that shuts off the parameter adaptation when the reference model error is below a threshold. The original adaptation law expressed above in Equation 98 can be redefined as

\[
\hat{\Theta} = \begin{cases} 
T_0 \Theta(u_{bl}, x)e^T P\tilde{B} & \text{if } \|e\| > e_0 \\
0_{N \times m} & \text{else} 
\end{cases} \tag{103}
\]

where \(e_0\) is the threshold below which adaptation is disabled. This threshold must be chosen such that the system only adapts when the estimation error is large relative to the modeling error\cite{102}.

The dead-zone operator given above will help mitigate parameter drift. However, it is not Lipschitz-continuous, and thus can introduce chattering or other undesirable effects when the error is close to the cutoff threshold, \(e_0\). A Lipschitz-continuous version of the dead-zone operator was introduced by Slotine and Coetsee\cite{100} for this reason. In
order to define this smooth version of the dead-zone operator, the operator, \( \mu \), is introduced as:

\[
\mu(\|e\|) = \max \left( 0, \min \left( 1, \frac{\|e\| - \delta e_0}{(1 - \delta)e_0} \right) \right)
\]

where \( 0 < \delta < 1 \) is a constant which is chosen to shape the transition from no adaptation to full adaptation.

Using this operator, the adaptive law given above in Equation 103 can be redefined as:

\[
\dot{\hat{\Theta}} = \Gamma_{\Theta} \Phi(\mathbf{u}_{bl}, x) \mu(\|e\|) \mathbf{e}^T \mathbf{P} \mathbf{B}
\]

The effectiveness of the dead-zone operator is determined by the choice of cutoff value, \( e_0 \), and transition value, \( \delta \).

**5.2.3.2 Projection Operator**

The projection operator is another useful robustness enhancement for adaptive control. This technique was originally described in Pomet and Praly\[^{99}\], and later implemented and described in many applications\[^{95, 96, 106, 107}\]. When applied to MRAC, the projection operator is used to bound the parameter estimates, and if they begin to grow beyond a given bound, it will smoothly redirect them to stay bounded. The
projection operator when applied to MRAC serves to maintain the negative semi-
definiteness of the Lyapunov equation\cite{projection_operator}. The complete derivation of this operator has
been thoroughly covered in the above referenced literature, and thus will be omitted
here.

The projection operator is given by Equation 106 below.

\[
Proj(\theta, y) = \begin{cases} 
  y - \frac{\nabla f(\nabla f)^T}{(\nabla f)^T y} if \ f > 0 \wedge (y^T \nabla f) > 0 \\
  y, \quad \text{if else}
\end{cases}
\]

where \( f(\theta) \) is a convex function defined by

\[
f(\theta) = \frac{\|\theta\|^2 - (\theta_{\text{max}})^2}{\varepsilon_\theta (\theta_{\text{max}})^2}
\]

where \( \theta_{\text{max}} \) is the parameter bound, and \( \varepsilon_\theta \) is a constant that shapes the steepness of the
parameter bounding. This operator is visualized below in Figure 5-10 and Figure 5-11.

![Figure 5-10. Projection operator within transition region](image)

![Figure 5-11. Projection operator at parameter bounds](image)

Finally, the adaptive law with dead-zone modification given above in Equation 105
can be enhanced further by the addition of the projection operator as follows:

\[
\hat{\theta} = Proj \left( \hat{\theta}, \Gamma_{\theta} \Phi(u_{bl}, x)y ||e|| e^T P \tilde{B} \right)
\]

The effect of this is that so long as the parameter estimates are well within the
defined boundary, adaptation is performed as usual. When the parameter estimates
approach the boundary, they are smoothly redirected by altering their derivative so they cannot exceed the maximum defined bounds. This serves to enforce the boundedness requirements of the controller, and maintain negative semi-definiteness with respect to Lyapunov.

It is important to ensure that the bounds enforced are greater than the estimated magnitude of the uncertainty. However, if the parametric uncertainty is greater than that of the bounds, the performance of the controller will degrade gracefully due to the smoothing nature of the projection operator. For illustration, the example given above in Section 5.2.3 is modified to incorporate the projection operator. When the bounds imposed by the projection operator are greater than the known parametric uncertainty, there is no change in the convergence of the parameter estimates or the performance. If the bounds are set lower than the known parametric uncertainty, only partial adaptation is realized, but the performance of the system degrades gracefully. To illustrate, the example provided in Section 5.2.3 can be repeated with the addition of the projection operator. In this example, the projection bounds are set lower than the known parametric uncertainty. As seen in Figure 5-12 and Figure 5-13, while the system does not achieve full adaptation, it degrades into a stable, bounded response that is still improved from the initial un-adapted state.
Figure 5-12. MRAC tracking performance with projection bounds

Figure 5-13. MRAC parameter estimates with projection bounds
5.2.3.3 Reference Model Modification

The previous modifications serve to improve the robustness of the algorithm. Another primary concern with traditional MRAC is the speed of adaptation and the overall transient response. As shown by Stepanyan and Krishnakumar\cite{97}, the transient response of the MRAC controller can also be improved upon by introducing an open-loop observer to the reference model as follows:

\[
\dot{x}_{ref} = A_{ref}x_{ref} + B_{ref}r(t) + \lambda e(t), \quad x_{ref}(0) = x_0
\]

where \(\lambda \in \mathbb{R}\) is an additional design parameter that serves as a prediction error feedback gain.

As shown in Stepanyan and Krishnakumar\cite{97}, as well as Lavretsky and Wise\cite{37}, the Lyapunov-based stability proofs are still valid with this representation. During the adaptation, the reference model is adjusted proportionally to its error. As a result, the controller may not perfectly track the reference model during the adaptation. However, as the parametric uncertainty estimates converge, the error term goes to zero. This results in the new term in the reference model cancelling out and thus reverting to the desired response. This modification allows for the adaptation gain to be increased, leading to faster convergence of parametric uncertainty. There are some limitations placed on the choice of \(\lambda\); This discussion is provided in the above-listed resources.

5.2.3.4 Application to UAV Trajectory Tracking

The resulting MRAC controller is a combination of the traditional Lyapunov-based MRAC with LQR with integrator action, dead-zone and projection-based adaptation robustness enhancements, and finally improved transient response due to the reference model modifications. For UAV trajectory tracking, two separate MRAC systems are used in the inner-loop controller – one for longitudinal movement and one for lateral movement. The longitudinal controller is used to command the aircraft pitch. The lateral controller is used to command bank angle and a sideslip angle. The pitch angle
and bank angle are calculated by the outer-loop controller, and the sideslip angle is given as a constant zero. This structure is identical to that of the above-described LQR with integral action controller; in fact, if the parameter estimates and the adaptive gain are set to zero, i.e., no adaptive control component, the MRAC algorithm simplifies back into LQR with integral action.

For this system, the reference model is specified as the closed-loop response linearized aircraft model given in Section 5.1.4 under the action of the LQR with integral action. Therefore, when the actual dynamics are the same as the linearized model, no adaptation is necessary. If the response deviates from that of the reference model due to modeling errors, environmental conditions, or subsystem failures, the parameter estimates will automatically update to bring the response back to that of the reference model.

As with the other elements of this simulation, the controller was implemented in Simulink with an emphasis on modularity. During the development of this controller, many subsystems were created. S-functions were created for more complex tasks such as the projection operator. For completeness, the Simulink implementation for the modified MRAC controller is provided in Appendix A.

5.2.4 IMRAC

The robust MRAC algorithm described in the previous section is capable of adapting to uncertainties in a system and restoring the response to that of the reference model. However, even with the modifications for improved transient response, there are still limitations as to how rapidly the controller can adapt to compensate for a given disturbance. In many high-performance applications such as UAV trajectory tracking, MRAC simply cannot adapt quickly enough to restore the system’s performance such that the mission can be salvaged.
By incorporating principles from the biological immune system, significant improvements can be made to the transient response of the above-described robust MRAC mechanism. The methodology developed here will be described as immunity-based model reference adaptive control (IMRAC). In essence, by extending the detector-based AIS approach described in Section 3.3, past experiences of the parametric uncertainty present at a given condition can be stored along with the detector structure and later recalled as an initial state for future adaptation. The first time a new disturbance is encountered, the response will be the same as the robust MRAC algorithm. This will be referred to as the naïve response. When subsequent disturbances of the same nature are met, the parameter estimates from the earlier encounter are used as the starting point for adaptation. Given sufficient exposure to a disturbance or failure, the parameter estimates are assumed converged for that condition. Such a detector will be referred to as matured. Additionally, if no prior experience for a given disturbance is available, the parameter estimates from a neighboring detector can be used as the initial condition for adaptation with the assumption that if the disturbance is sufficiently close in the solution space, then the true unknown parameters are likely to be similar as well. In this manner, the concept of vaccination can be introduced: less severe failure conditions are introduced to provide exposure so that a more appropriate response can be mounted for more severe failure conditions of the same nature in the future. Additionally, detectors can be matured using simulated disturbances or failure situations so that the system has a better chance of survival when encountering a real threat.

Fundamentally, the enhanced MRAC mechanism becomes analogous to that of the innate immune system – pathogens, or error in this analogy, are dealt with in a non-specific, generalized manner. Conversely, the AIS detector-based mechanism added to the MRAC algorithm is analogous to the adaptive immune system – prior experience is used to more efficiently deal with specific pathogens in the future.
The IMRAC mechanism introduces an AIS detection scheme\textsuperscript{[11, 17]}. The detection scheme incorporates \textit{features} from the system state that are capable of capturing the signature of failures. Previous research has shown that very high detection rates can be achieved by incorporating ANN state estimates into the feature sets\textsuperscript{[11, 108]}. The IMRAC architecture can be seen in Figure 5-14 below. The following sections will describe the detection scheme, the ANN state predictor, and the application to UAV trajectory tracking.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5_14.png}
\caption{Overview of the AIS MRAC mechanism}
\end{figure}

\textbf{5.2.4.1 ARTIFICIAL IMMUNE SYSTEM DETECTION SCHEME}

An AIS detection scheme is used to recognize when the abnormalities or disturbances are present in the system through the process of self/non-self discrimination. As described in Section 3.3, and in Moncayo \textsuperscript{[11]} and Perhinschi \textit{et al.}\textsuperscript{[85]}, large amounts of state data are collected from the system during various normal operating conditions. The nominal data should be extensive enough to sufficiently represent all conditions of the self. This data is used to identify subsets of the solution space, or universe, $\mathbb{U}$, corresponding to the \textit{self}, $S$. All conditions, which are not part of the self, are considered abnormal, or \textit{non-self}, $\bar{S}$. This is expressed symbolically as

\begin{equation}
\bar{S} := \mathbb{U} - S
\end{equation}
In theory, the self and non-self completely cover the solution space. However, the self data is collected at discrete points. For computational purposes, these data points are grouped into hyper-dimensional geometrical shapes. In this research, hyper-spheres are the chosen shape, although many different representations are possible. A hyper-sphere with dimensionality of three is simply a sphere. A center point and a radius are used to represent a hyper-sphere. Self and non-self hyper-spheres are referred to as *clusters* and *detectors*, respectively. In practice, it is not possible to entirely cover the solution space without overlap among the detectors. Self clusters may be allowed to overlap each other and non-self detectors may be allowed to overlap each other; however, detectors are never permitted to overlap the self.

The complete state of the system can be large in some applications. However, many numerical problems arise as the dimensionality of the detectors increases. For this reason, the hierarchical multi-self strategy was developed\[^{11}\]. This architecture shows that it is possible to strategically use subsets of the available state data to detect and identify abnormalities without the need for high dimensionality. The subset of the state variables are referred to as *feature sets*. Individual variables are referred to as *features* or *identifiers*. The dimensionality of the hyperspace is equal to the number of features considered in a detector set. Ideally, a feature set is selected such that the variables are capable of capturing the signature of the various failures that may occur in the system.

In the case of UAV trajectory tracking, previous research has shown that the roll, pitch, and yaw attitude rates along with open-loop neural network-based attitude rate estimations are good features for AIS fault detection\[^{11,108}\]. Let a feature set be defined as

\[ F = \{f_1, f_2, \ldots, f_n\} \]

where \( f_i \) are the individual features, or *identifiers*.

The self is made up of clusters. All of the identifiers are normalized such that the data fits within a unit-hypercube plus an extra padding. This will be presented in more
detail in the following section. The self clusters can be expressed mathematically as follows:

\[ S = \{c_1, c_2, ..., c_{N_c}\} \]

where \( N_c \) is the number of self clusters. The clusters are defined as:

\[ c_i = [C_i \ R_{ci}]^T = [C_{i1} \ C_{i2} \ ... \ C_{in} \ R_{ci}] \]

where \( C_i \) is the center of the cluster, and \( R_{ci} \) is its radius.

Likewise, the detectors can be expressed as follows:

\[ \bar{S} = \{d_1, d_2, ..., d_{N_d}\} \]

where \( N_d \) is the number of detectors. The clusters are defined as:

\[ d_j = [D_j \ R_{dj} \ \hat{\Theta}_j]^T = [D_{j1} \ D_{j2} \ ... \ D_{jn} \ R_{dj} \ \hat{\Theta}_j] \]

where \( D_j \) is the center of the detector, \( R_{dj} \) is its radius, and \( \hat{\Theta}_j \) is the parametric uncertainty estimate described above in Section 5.2.3. Note that the parameter estimates are only important for the detectors since they are the criteria that are used for negative selection.

The detection process is binary; either the system is operating as normal or it is not. This can be expressed mathematically as:

\[ Det \in \{0, 1\} \]

where 0 corresponds to nominal conditions and 1 corresponds to an abnormality.

Let the current normalized feature state be defined as:

\[ X_k = [\chi_{k1} \ \chi_{k2} \ ... \ \chi_{kn}] \]

where the \( \chi_i \) is the current system value for a given feature variable.

Detection for a given feature state is calculated based on whether or not the current system feature state falls within any detectors. For hyper-spheres, this is computed using the Euclidean distance from the current point and the detector centers and radii. This is defined as:
\[ Det_k = \begin{cases} 0 & \text{if } \sqrt{(\chi_{k1} - D_{j1})^2 + (\chi_{k2} - D_{j2})^2 + \ldots + (\chi_{kn} - D_{jn})^2} > R_{dj} \text{ for } j = 1, 2, \ldots, N_d \\ 1 & \text{else} \end{cases} \]

It is plausible that a few data points may trigger detection, but an abnormal condition is not actually present. However, if many consecutive points are activated, then an actual fault is likely. Consequently, it is useful to implement some type of additional scheme to help prevent false alarms. In practice this is usually a moving average or a filter with a detection threshold. In this proposed methodology, a moving average is used for this purpose. Absolute detection can be defined as follows:

\[ Det \in \begin{cases} 1 & \text{if } \frac{1}{N_{ma}} \sum_{i=k-N_{ma}}^{k} Det_i \geq \varepsilon_{ma} \\ 0 & \text{else} \end{cases} \]

where \( N_{ma} \) is the size of the moving average window, \( k \) is the current time step, and \( \varepsilon_{ma} \in [0, 1] \) is the threshold for detection. Essentially, detection will be triggered if the percentage of previous data points inside the moving average window is above the specified threshold.

Once an abnormality has been discovered, the parameter estimate portion of the individual detector responsible for triggering detection is used to reinitialize the MRAC parameter estimates. The detector responsible for triggering absolute detection is assumed to be the last one activated. It is possible that multiple detectors can be activated at once, and consequently, multiple detectors could be responsible for absolute detection. In this case, the parameter estimate used to initialize the adaptive law will be assigned to the average of the matured activated detectors. Once the MRAC has converged for a given abnormal condition, the matured parameter estimates are stored back into the original detector or detectors which triggered the adaptation mechanism. For the case of multiple detectors, the same set of matured parameter estimates is stored into each of them.
Using this method, the concept of identification is not necessary. Like the process of AIS-based identification, information is attached to specific detectors. For this controller, it is not important to know what the exact nature of the failure is. Instead, the only important information is the parameter estimates associated with a given detector.

To summarize, before IMRAC can be used, a detector set is needed. For application to UAV trajectory tracking, ANN-based state predictors are used as features. Therefore, these predictors must be initialized and trained before they can be used. As will be discussed in Section 5.2.4.3, the inputs to the ANN algorithm are collected using nominal test data. Next, the ANN is trained to be recognized the expected state of the aircraft by learning from the nominal training data. When all of the features are available, the AIS mechanism can be trained. Once again nominal test data must be collected, only this time, all of the AIS features are incorporated. These data are grouped into self-clusters to form the self. After the self has been established, detectors are generated at random until adequate coverage of the non-self has been achieved. Detectors overlapping the self or with centers falling within another detector are rejected during the non-self generation process. This process is summarized in the flowchart provided below in Figure 5-15.

![Figure 5-15. IMRAC preparation flowchart](image)

During operation, a given data point is tested against all of the detectors to determine if it falls within one. If the data point falls within a detector, then the detector is marked as active and the operating condition is considered abnormal. If any
A parametric estimate piece of data is associated with the activated detector, then the integrator in the MRAC adaptation law is re-initialized using the parameters. If the activated detector has not been matured, but a close neighboring one within a threshold is, then that value will be used to initialize the adaptation. If no matured detectors are available, then the adaptation parameters will simply start from zero. Once the MRAC has converged, the parameters are saved as metadata alongside the activated detector; this forms a matured detector. A flowchart of this process can be seen in Figure 5-16 below.

Figure 5-16. IMRAC operation flowchart
5.2.4.2 Generation of Detectors

Several steps are required to generate a suitable set of detectors. In this research effort, the West Virginia University Immunity-Based Failure Detector Optimization and Testing application was used to generate the detectors used by the IMRAC algorithm. This utility is described in detail in Davis[17] and Davis et al[109]. For completeness, the detector generation process is summarized in this section.

First, sample data at normal operating conditions must be collected. These data are used to form the definition of the self. They should cover as many expected normal operating conditions as feasible.

Once the self-data have been collected, they must be normalized. This ensures that all of the identifiers are of the same scale. To allow for data outside the nominal range, an additional padding is added to the normalization so that data points outside the typical operating range can be detected as well. A given normalized identifier is given as

\[ \chi_{ki} \in [0 + \mu_n, 1 - \mu_n] \]

where \( \chi_{ki} \) is a given state variable, and \( \mu_n \in [0,0.5) \) is the normalization padding constant. The maximum value, \( \max(\chi_{ki}) \), of a given identifier will have a value of \( 1 - \mu_n \), and the minimum value, \( \min(\chi_{ki}) \), will have a normalized value of \( 0 + \mu_n \). The normalized set of self data is completely contained within a unit-hypercube.

After the data have been normalized, duplicate, or redundant, points are removed. If multiple points are identical, only one point will be retained. Equally, if the multiple points are very close together, within a threshold, the points will be removed and replaced by a single point at the center of the points. This step helps with the computational overhead of clustering.

Once the data are normalized and any duplicates are removed, points are grouped together using the \( k \)-means clustering algorithm\[110,111]. The clustered data are ultimately
expressed as hyper-spheres or hyper-rectangles. This algorithm greatly reduces the computational requirements for evaluating if a test point falls within the self. At this stage of the process, the self-data have been processed, and a clustered representation of the self has been created.

Next, the detectors can be generated. During this process, detectors, in this case hyper-spheres, are generated at random. The detectors’ centers are placed randomly within the unit hyper-cube, and the radius is given a random value within a pre-determined range. Each time a detector is generated, a test is performed to determine if the detector falls within or intersects the self or is completely inside another detector; if it does not, then it is added to the detector set, otherwise it is discarded. Note that overlap with the self is not permitted, but it is with other detectors. This process continues until a satisfactory portion of the non-self feature space is covered by detectors. The coverage is calculated probabilistically using the Monte Carlo volume estimation method\[^{112, 113}\]. The process of generating detectors is summarized in the flowchart provided below in Figure 5-17.
Any state variable can be used as a feature. However, some features are much more capable than others of obtaining high detection rates. Previous research has shown that neural network-based state predictors are exceptionally well-suited to serve as AIS features\cite{11,108}. Fundamentally, nominal data are collected from the system and used as training data to teach the neural network, by adjusting the neuron weights, how to predict the state.

As discussed in Section 2.2.6, neural networks consist of many neurons logically grouped into nested layer structures. As seen in Figure 5-18, an individual neuron consists of a simple activation function that produces a single output based on multiple
weighted inputs. Individually, the neurons are primitive; however, when arranged into hierarchical networks they are capable of sophisticated prediction and pattern matching capabilities.

In this research, a back propagation ANN with one hidden layer is used. As shown in Figure 5-19, the back propagation network consists of an input layer, a hidden layer, and an output layer.
In order for a back propagation ANN to be useful, it must first be trained. Training is an iterative process where training data points are presented to the network. A given training point contains both inputs and expected outputs. The training process consists of two generalized steps: a feedforward step and a back propagation step. During the feedforward step, the hidden and output layer outputs are calculated. Then the back propagation step starts at the outputs and works toward the inputs and adjusts the weights along the way.

The first step in training the network is to initialize the hidden and output layer weights to random, small bipolar numbers within a specified range, for example, $[-0.3, 0.3]$. The choice of the range is not very critical so long as all of the values are between -1 and 1.

Next a training pattern is presented to each of the hidden and output layer neurons and their outputs are calculated. The training pattern consists of an input vector, $x_1, ..., x_{n_m}$ where $n_m$ is the number of inputs to a given layer and the desired outputs, $d_1, ..., d_m$ where $m$ is the number of outputs. The neurons are referred to by $N_{km}$ where
$k$ is the index of the neuron in layer $m$ where 1 is the hidden layer, and 2 is the output layer. The weight of a given neuron input is denoted by $w_{km}$ where $k$ is the neuron, $m$ is the layer, and $i$ is the input number.

The outputs of the individual neurons are determined using a weighted sum of the inputs and an activation function. The weighted sum of inputs for a given neuron $N_{km}$ at epoch $t$ is defined as:

$$\sigma_{km}(t) = \sum_{i=1}^{n} w_{km}(t)x_i(t)$$ \hfill (121)

There are many different types of activation functions, and the choice of activation function may play an important role in the performance characteristics of the network; however, the intricacies of the various activation functions are out of the scope of this research. In this application, the bipolar sigmoid function, as given in Equation 121 is used. As a general requirement for activation functions, the bipolar sigmoid function is both nonlinear and differentiable.

$$f(x) = \frac{2}{1 + e^{-x}} - 1$$ \hfill (122)

Using the bipolar sigmoid activation function, the neuron outputs at epoch $t$ for the back propagation ANN structure are given as:

$$y_{km}(t) = \frac{2}{1 + e^{-\sigma_{km}}} - 1$$ \hfill (123)

The neuron output is calculated the same in the feedforward pass as it is when the network is fully trained and it is being used for prediction.

After the feedforward pass, a recursive, or back propagation, algorithm is used to adjust the neuron weights, starting at the output layer. For a given layer, the change in weights is defined as follows:

$$\Delta w_{km}(t + 1) = \eta \delta_{km} x'_{km}(t) + \alpha \Delta w_{km}(t)$$ \hfill (124)
for $0 < i < n_m$, where $\eta \in (0, 1)$ is the learning rate, $x'_{km}(t)$ is the $i$ input to neuron $k$ for layer $m$, $\alpha \in [0, 1]$ is the momentum term, and $\delta_{km}$ is given by:

$$\delta_{km} = 0.5(1 - y_{km}(t))(1 + y_{km}(t))e_k(t)$$

for the output layer. Let $e_k$ be the error in the output as defined by

$$e_k(t) = d_k - y_{km}(t)$$

and $\delta_{km}$ is given by:

$$\delta_{km} = 0.5(1 - y_{km}(t))(1 + y_{km}(t))w_{km}(t)\delta_{k(m+1)}$$

for the hidden layer.

Finally, the weights can be updated as follows:

$$w_{kml}(t + 1) = w_{kml}(t) + \Delta w_{kml}(t + 1)$$

This feedforward then back propagation process is iterated until the error term for the output layer converges below a specified threshold, or a maximum number of iterations have been performed.

### 5.2.4.4 Application to UAV Trajectory Tracking

For use in UAV trajectory tracking, the resulting IMRAC algorithm is used in the same place as the robust MRAC controllers. Any controller can be used for the outer-loop; for this research the simple, linear controller from the position PID method is used. In the inner-loop, two separate IMRAC controllers are used: one for the lateral-directional channel and the other for the longitudinal channel.

For the AIS component of the controller, the feature set is defined as:

$$F = \{p, q, r, NN_p, NN_q, NN_r\}$$

where $NN_p, NN_q$ and $NN_r$ are the neural network-based attitude state estimates. The inputs to the ANN are:

$$X = \{V, z_e, \alpha, \beta, e_p, e_q, e_r, \delta_e, \delta_a, \delta_r, \delta_T\}$$
where \( V \) is the velocity, \( z_e \) is altitude, \( e_p, e_q, \) and \( e_r \) are the errors between the actual and commanded attitude rates, and \( \delta_e, \delta_\alpha, \delta_r, \) and \( \delta_T \) are the commanded surface deflections and throttle produced by the controllers.

Before the ANN can be used for state predictions, it must be trained. This consists of collecting training data and adjusting the ANN weights until satisfactory performance is achieved. For obtaining training data, the robust MRAC algorithm was used to fly several different trajectories at various points in the flight envelope. The ANN inputs were saved out of this data. This resulted in a very large amount of data points. To reduce the computational overhead of training the ANN and to reduce the risk of over training the network, only 1% of the available training data was used. This data was selected randomly from the full set of training data. Sample results from the trained ANN state predictor can be seen below in Figure 5-20. As seen in Figure 5-21, the error for the estimates generally stays close to zero with the exception of a few spikes to approximately 0.1 rad/s around turning maneuvers.
Figure 5-20. Neural network angular rate estimation for figure-8 trajectory
Note that the ANN-based state predictors are not necessary for the AIS detection scheme; any feature set capable of capturing the signature of a failure can be used. As described earlier, they are used in this research effort since they have been shown to possess exceptional detection capabilities.

Once all of the features are available, the AIS detector set must be generated before the IMRAC scheme can be used. In order to define the self, test data must be collected. In this application, the robust MRAC was used to fly the sample trajectories at nominal conditions. This is similar to the process used when collecting data to train the ANN.
state predictor, except now the ANN is running in an open-loop configuration so that the state estimates can be recorded as part of the feature set. This data is used to generate the self following the methodology described above in Section 5.2.4.2.

Immediately after the detectors have been generated, the controller is considered naïve; its performance is the same as the robust MRAC algorithm. However, the controller learns to better deal with failure conditions as it gains experience. Given sufficient experience, the performance of the IMRAC controller under failure conditions should approach that of the LQR plus integral controller under nominal conditions.

The IMRAC controller was implemented in Simulink. Like the other controllers, it was created with modularity and re-use in mind. Specialized Simulink subsystems had to be created to perform the ANN state estimations and AIS-based abnormality detection in real-time. The simulation makes heavy use of triggering and latch mechanisms for re-initializing MRAC parametric uncertainty estimates with those stored in a given activated detector. For completeness, the Simulink implementation for the IMRAC controller is provided in Appendix A.
When comparing the performance of the various trajectory tracking controllers, it is important that all of the controllers be optimized such that an unbiased comparison can be made. As such, a technique is needed for optimizing the various controller gains. In this section, a genetic algorithm is developed for optimizing the controllers.

6.1.1 Genetic Algorithm Background

Genetic algorithms are a category of artificial intelligence techniques that are inspired by biological evolution. In biological evolution, organisms within a population that are more suited to a given environment are more likely to survive long enough to produce offspring, while unfit individuals are more likely to die off before they produce offspring. When an individual produces offspring, many of the traits that facilitated its survival are passed down to its offspring. Thus, over many generations, through the mechanism of natural selection, the fitness of the individuals within a population is expected to increase.

In biological organisms, deoxyribonucleic acid (DNA) forms the building blocks of life. DNA acts as a type of blueprint that defines the traits, characteristics, and inner workings of the organism. Within DNA, genes contain instructions for producing proteins and chemicals the body needs. A chromosome is a threadlike linear strand of genetic material, or genes. When two organisms produce an offspring, each parent passes on a portion of their DNA to the child; the DNA of the offspring is a combination of parent’s DNA. When a section of genes from one chromosome is swapped with a section of the genes from another chromosome, this is referred to as crossover. In addition to crossover, random mutations may also occur and alter individual genes. Through the processes of crossover and mutation, the offspring’s genetic material may differ from that of either parent, and thus will express different character traits which
may make the individual more or less suited to a given environment. Over many
generations, the random variations in offspring and natural selection mechanisms lead
to individuals better suited for survival in a given environment than were originally
present in earlier populations.

6.1.2 Genetic Algorithm Applied to Controller Optimization

When applying the genetic or evolutionary paradigm for optimization of gains for
aircraft trajectory tracking control laws, an individual is defined as a potential solution
that is a set of gains for the trajectory-tracking controller. In this research, the structure
of the control laws is assumed fixed, although modifications can be implemented by
setting some of the gains to zero or non-zero values. Within an individual, a gene is used
to refer to a particular controller gain. A population is therefore defined as a collection of
individuals, or sets of potential gain configurations. A genetic operator is an action that
results in a modification to an existing individual’s gene configuration. For this
research, two genetic operators are defined, crossover and mutation.

A modified genetic algorithm was designed and implemented to optimize the
controller gains for UAV trajectory tracking algorithms[114]. In order to accomplish this, a
streamlined version of the WVU UAV Simulation Environment was developed, as was
discussed in Chapter 4. The purpose of this compact framework is to provide a user-
friendly interface between the simulation environment and the GA. This improves the
generality and flexibility of the environment as well as the GA by easily allowing for
tuning of gains of controllers with various structures in response to different flight
scenarios. This reduces user interaction and decreases computational overhead.

The first stage of the optimization is to initialize the Simulink model. During this
step, the aircraft aerodynamic model is loaded with its initial conditions and
aerodynamic coefficients. The optimization algorithm can be run at nominal conditions
or with failures injected. If failures are desired, they are initialized here. Finally, the desired test trajectory is loaded.

The gains for a given trajectory tracking controller are represented to the GA as a vector of size $l$; each element of the vector is used to represent a unique controller gain. *Real-value* representation is used for the chromosome elements. The population is defined as an $n$ by $l$ matrix of $n$ individuals containing $l$ gains.

A summary of the optimization process is as follows. As in the biological analogy, an initial population must exist. The initial population is generated randomly within pre-determined bounds. Once this process is complete, the evolutionary process may begin. The individuals in the population are subjected to random mutations and crossover operations. This new population is then rated using a performance index or fitness function, which serves the purpose of the environment. A new population is generated based on the fitness of the individuals in a process that mimics natural selection. These operations make up one generation for the population. The distinct stages of this algorithm will be explained in more detail in the following section. The evolutionary operations repeat until a convergence criterion is met, generally a predefined number of generations. An overview of the GA optimization algorithm is presented in Figure 6-1.

![Figure 6-1. Overview of the evolution process](image-url)
6.1.2.1 Generation of Initial Population

The initial population can be generated in two different ways. The first method is to define a large lower and upper bound for each of the genes/gains and then generate random values that fall within the specified bounds. This approach has the benefit of providing a lot of variety to the population, which leads to a more thoroughly explored solution space. The downside to this method is that the high dimensionality of the chromosomes and the complexity of the trajectory tracking controllers statistically lead to very poor starting controllers; thus the GA will take much longer to converge. To mitigate this problem, the individuals in the starting population are rated and ones falling below a specified threshold are discarded and re-generated until the convergence criterion has been met. This improves convergence time, but greatly increases the time required to generate the initial population.

The other method for generating the initial population uses an initial seed. It consists of a heuristically-tuned set of gains verified to exhibit basic trajectory tracking proficiency and stability during nominal operating conditions. This initial seed is inserted into the population as the first individual. Subsequent individuals are generated as before, but the range is narrowed around the initial guess. In practice, genes were given a range that was approximately ±50% of the corresponding gene from the initial guess. The resulting population, \( P \), is represented as an \( n \) by \( l \) matrix; \( P_{ij} \) is used to refer to generation \( i \), individual \( j \). This overall initialization process (using initial seeding) is presented in Figure 6-2.
Note that the initial bounds are primarily used in conjunction with the initial seed for the purpose of shortening the convergence time of the algorithm. This assumes that the optimal solution is near the heuristically-tuned solution. However, while bounds are used to generate the initial population, such limits are not enforced throughout the optimization process. Consequently, if the true optimum lies outside of the initial bounds, the solution may still be discovered by the GA.

6.1.2.2 Mutation Operator

The first operation performed on the population is mutation. First, a random decision matrix, $Λ$, is defined in Equation 131. In this equation, the $\text{rand}$ operator produces an $n$ by $l$ matrix of uniformly-distributed random numbers on the closed interval from 0 to 1.

$$Λ = \text{rand}(n, l)$$  

Next, an $n$ by $l$ matrix of genes chosen for mutation, $Γ$, is generated by Equation 132.

$$Γ_{ij} = \begin{cases} 
1 & \text{if } Λ_{ij} \leq μ \\
0 & \text{if } Λ_{ij} > μ
\end{cases}$$  

In this equation, the mutation rate, $μ$, is defined as the percentage of the genes in the population that should statistically be subjected to a mutation operation, subscript $i$ represents the index of individual in the population, and subscript $j$ represents the index of the gene. From matrix $Γ$, a conjugate matrix, $\bar{Γ}$, is defined as:
\[
\Gamma_{ij} = \begin{cases} 
1 & \text{if } \Gamma_{ij} = 0 \\
0 & \text{if } \Gamma_{ij} = 1 
\end{cases}
\]

where the subscripts \(i\) and \(j\) retain their meaning from Equation 132.

An \(n\) by \(l\) chromosomal modification limit matrix, \(\Sigma\), is defined in Equation 134. In this equation, the \(ones\) operator is defined as a \(n\) by 1 matrix where each element is equal to 1. The chromosomal modification limit, \(\bar{\sigma}\), is defined as a 1 by \(l\) matrix. The indices of the modification limit vector define the percentage by which a gene selected for mutation can mutate.

\[
\Sigma = \text{ones}(n,1) \ast \bar{\sigma}
\]

An \(n\) by \(l\) potential modification matrix, \(K\), is defined as:

\[
K = P + P \odot (-\Sigma + 2 \ast \text{rand}(n,l) \odot \Sigma)
\]

where \(\odot\) is used to define piecewise multiplication. Once the potential modification matrix is calculated, it can be combined with \(\Gamma\) and \(\overline{\Gamma}\) to produce the new mutated population.

\[
P = \Gamma \odot K + \overline{\Gamma} \odot P
\]

An illustration of a mutated population is presented in Figure 6-3. A flowchart of the effective mutation process can be seen in Figure 6-4.
**Figure 6-3.** Mutation operation

![Mutation Matrix](image1)

<table>
<thead>
<tr>
<th>K1,1</th>
<th>K1,2</th>
<th>K1,3</th>
<th>K1,4</th>
<th>K1,5</th>
<th>K1,6</th>
<th>K1,7</th>
<th>K1,8</th>
<th>K1,9</th>
<th>K1,10</th>
<th>K1,11</th>
<th>K1,12</th>
<th>K1,13</th>
<th>K1,14</th>
</tr>
</thead>
<tbody>
<tr>
<td>K2,1</td>
<td>K2,2</td>
<td>K2,3</td>
<td>K2,4</td>
<td>K2,5</td>
<td>K2,6</td>
<td>K2,7</td>
<td>K2,8</td>
<td>K2,9</td>
<td>K2,10</td>
<td>K2,11</td>
<td>K2,12</td>
<td>K2,13</td>
<td>K2,14</td>
</tr>
<tr>
<td>K3,1</td>
<td>K3,2</td>
<td>K3,3</td>
<td>K3,4</td>
<td>K3,5</td>
<td>K3,6</td>
<td>K3,7</td>
<td>K3,8</td>
<td>K3,9</td>
<td>K3,10</td>
<td>K3,11</td>
<td>K3,12</td>
<td>K3,13</td>
<td>K3,14</td>
</tr>
<tr>
<td>K4,1</td>
<td>K4,2</td>
<td>K4,3</td>
<td>K4,4</td>
<td>K4,5</td>
<td>K4,6</td>
<td>K4,7</td>
<td>K4,8</td>
<td>K4,9</td>
<td>K4,10</td>
<td>K4,11</td>
<td>K4,12</td>
<td>K4,13</td>
<td>K4,14</td>
</tr>
<tr>
<td>K5,1</td>
<td>K5,2</td>
<td>K5,3</td>
<td>K5,4</td>
<td>K5,5</td>
<td>K5,6</td>
<td>K5,7</td>
<td>K5,8</td>
<td>K5,9</td>
<td>K5,10</td>
<td>K5,11</td>
<td>K5,12</td>
<td>K5,13</td>
<td>K5,14</td>
</tr>
<tr>
<td>K6,1</td>
<td>K6,2</td>
<td>K6,3</td>
<td>K6,4</td>
<td>K6,5</td>
<td>K6,6</td>
<td>K6,7</td>
<td>K6,8</td>
<td>K6,9</td>
<td>K6,10</td>
<td>K6,11</td>
<td>K6,12</td>
<td>K6,13</td>
<td>K6,14</td>
</tr>
<tr>
<td>K7,1</td>
<td>K7,2</td>
<td>K7,3</td>
<td>K7,4</td>
<td>K7,5</td>
<td>K7,6</td>
<td>K7,7</td>
<td>K7,8</td>
<td>K7,9</td>
<td>K7,10</td>
<td>K7,11</td>
<td>K7,12</td>
<td>K7,13</td>
<td>K7,14</td>
</tr>
<tr>
<td>K8,1</td>
<td>K8,2</td>
<td>K8,3</td>
<td>K8,4</td>
<td>K8,5</td>
<td>K8,6</td>
<td>K8,7</td>
<td>K8,8</td>
<td>K8,9</td>
<td>K8,10</td>
<td>K8,11</td>
<td>K8,12</td>
<td>K8,13</td>
<td>K8,14</td>
</tr>
</tbody>
</table>

- **Pop. size n = 8**
- **Chromosome length l = 14**
- **not mutated**
- **mutated**

**Figure 6-4.** Mutation flowchart

```
Start

Select next individual within the population

Select next gene within the current individual

Is rand < mutationRate?  
  Yes  
  Randomly add or subtract rand * kMutationRate to the selected gene
  Yes  
  Is another gene present?  
    Yes  
    Is another individual present?  
      Yes
End

No

Is another individual present?  
  Yes  
  Is another gene present?  
    Yes

No

Is another gene present?  
  Yes

No

Is rand < mutationRate?  
  No

Select next individual within the population
```
After mutation, the crossover operation is performed. A crossover rate, \( \nu \), is defined as half the probabilistic percentage of the population that should be subjected to a crossover operation in one generation. Note that this is defined as half because each crossover operation affects both a source and target individual; thus, this is the probability of choosing an individual as the source individual. For each individual in the population, a random number in the range \([0,1]\) is chosen. If \( \nu \) is less than this random number, then a crossover operation is performed on the current individual, \( P_{ic} \).

To perform a crossover operation, a random integer index, \( a \), is selected in the range \((1,l)\). This serves as the lower crossover bound. The upper crossover bound, \( b \), is a random integer value selected in the range \((a,l)\). A random individual from the population that is not the current individual, \( P_{ix} \), is chosen to crossover the current individual with. The crossover operation is described by Equation 137; note that in this equation, the notation \( P_{ij}(g:h) \) is defined as a vector consisting of elements \( g \) through \( h \) of individual \( j \) in the population at generation \( i \). An illustration of the crossover operation is presented in Figure 6-5. An overview of this process can be seen in Figure 6-6.

\[
P_{ic} = [P_{ic}(1:a-1) P_{ix}(a:b) P_{ic}(b+1:l)]
\]

\[
P_{ix} = [P_{ix}(1:a-1) P_{ic}(a:b) P_{ix}(b+1:l)]
\]

![Crossover Operation Diagram](image)
6.1.2.4 Fitness Function

A performance index is used by the genetic algorithm to evaluate the fitness of a given proposed solution to the optimization problem. Any grading metric can be used so long as it rewards desired performance and penalizes inferior performance. For the genetic algorithm in question, the performance index is presented in Section 4.2.

When performing an optimization, it is helpful to expose the aircraft to a wide range of operating conditions. This helps prevent the controller from becoming overly optimized for a singular scenario and not robust to others. For this reason, in many
instances, both nominal and failure trials are incorporated into a singular fitness evaluation. In addition to failure condition, several different trajectories may be incorporated into the fitness evaluation so as to cover a larger portion of the flight envelope. For most of the trials performed, a trajectory consisting of a series of climbing then descending S-turns was used, as it alone covers a large portion of the flight envelope. When computing a performance index for multiple conditions, a simple average of the individual trials is used in the selection process.

### 6.1.2.5 Selection

The PI is a normalized measure of the aircraft’s trajectory following quality for a given gain/trajectory combination. It ranges from 0 to 1, where 0 is the worst possible score, and 1 is the theoretical best possible score. Let $\vec{PI}$ be defined as an $n$-dimensional vector containing the performance indices for all of the individuals in a given generation of the population.

The performance data is then used to select a new population. The roulette wheel selection method is used along with elitist selection strategy. In roulette wheel selection, each individual in the population is given a probability of survival proportional to its performance index. With elitist selection, the individual with the highest performance index is reinserted into the population independent of the outcome of the roulette wheel selection to ensure its presence in the next generation.

For the roulette wheel selection process, first the probability of selection of each individual $\tilde{p}$ is defined as:

$$\tilde{p} = \frac{\vec{PI}}{\sum \vec{PI}}$$

Next, the cumulative probability of selection $\vec{q}$, is defined as a $n$ dimensional vector given as:
\[
\tilde{q} = \left[ \sum_{i=1}^{1} p_i \sum_{i=1}^{2} p_i \ldots \sum_{i=1}^{n-1} p_i \sum_{i=1}^{n} p_i \right]
\]

or

\[
q_i = q_{i-1} + p_i
\]

where \(p_i\) is defined as element \(i\) of the vector \(\tilde{p}\).

For \(n\) individuals in the new population, a uniformly distributed random number in the range \([0,1]\) is generated. Then the value of the random number is compared to each element of \(\tilde{q}\). The first index in which the random number is greater will correspond to the individual chosen for placement in the new population. This continues until a new population has been generated. As stated above, once the roulette wheel selection is complete, elitist selection is used to ensure that the elite individual remains in the population and that the best solution is never lost unless a better one is discovered.

A novel variation to this selection process is to add an additional scaling, or normalization, step prior to the roulette wheel selection in order to help the algorithm converge more quickly. An issue with the fitness function used is that once the population is highly evolved, there is not significant distinction in the performance indices. This leads to the individuals in a population having similar probabilities of selection in the roulette wheel method. The additional normalization step leads to a dilated PI set that statistically focuses efforts on refining the best solutions. This provides quicker convergence to some optimized solution, but limits the exploration quality; thus a good solution is quickly found, but it may not be the global optimum.

The first step in the normalization process is to sort the PIs of the current population in ascending order. The normalization will result in the best individual with the highest PI obtaining a new scaled PI of 1.0, while the individual with the lowest PI will receive a scaled PI of \(m_{min} \in [0,1)\). The sorted PIs are grouped into two groups: those in the
lower $p$ percentile, and those above this percentile. The bottom tier is scaled linearly as defined in Equation 140.

$$ P_{I_{\text{normalized}}} = m_{\text{min}} + \frac{m_{\text{cutoff}} - m_{\text{min}}}{P_{I_{\text{cutoff}}}} * P_{I_{\text{original}}} $$

where $P_{I_{\text{cutoff}}}$ is the un-scaled PI of the best individual in the lower performing tier, $P_{I_{\text{original}}}$ is the PI currently being scaled, and $m_{\text{cutoff}}$ is the desired performance of the top individual in the lower-tier. This falls in the range of $m_{\text{min}} \leq m_{\text{cutoff}} < 1$. The upper tier of PIs is linearly scaled using the following relation:

$$ P_{I_{\text{normalized}}} = 1 + \frac{1 - m_{\text{cutoff}}}{P_{I_{\text{best}}}} * \left( P_{I_{\text{original}}} - P_{I_{\text{best}}} \right) $$

where $P_{I_{\text{best}}}$ is the un-scaled PI of the best individual in the current population. This described mapping function can be seen below in Figure 6-7.

As stated, the extra normalization is an optional step for obtaining quicker convergence. The overall selection process, including the normalization step, can be seen below in Figure 6-8.
An analysis was performed on the capabilities of the genetic algorithm for the purpose of optimizing the gains of the trajectory tracking controllers. For this analysis, a limited set of controllers was optimized due to the large time required for the algorithm to converge. For illustrative purposes, the position PID, outer-loop NLDI, extended NLDI, and the AIS adaptive position PID controller were optimized. This analysis was conducted to verify the functionality and performance of the GA. Additionally, the performance results of the optimized controllers helped to shape the development of the more intricate adaptive control techniques.

As seen in Figure 6-9 through Figure 6-12, the trajectory tracking controllers appeared to converge after approximately 200 generations. As can be seen in the figures, the average performance of the controllers stayed close to that of the elite individuals. This is the selection of the population size, mutation and crossover rates, and the normalization parameters. The initial and final performance indices obtained from the GA can be seen below in Table 6-1.
Table 6-1. Optimization results for coupled outer/inner-loop tuning

<table>
<thead>
<tr>
<th>Controller Type</th>
<th>Starting Nominal PI</th>
<th>Optimized Nominal PI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position PID</td>
<td>0.86</td>
<td>0.96</td>
</tr>
<tr>
<td>Outer-Loop NLDI</td>
<td>0.78</td>
<td>0.89</td>
</tr>
<tr>
<td>Extended NLDI</td>
<td>0.87</td>
<td>0.95</td>
</tr>
<tr>
<td>Adaptive Position PID</td>
<td>0.78</td>
<td>0.97</td>
</tr>
</tbody>
</table>
One of the novel approaches presented in this algorithm is the addition of an extra normalization technique before the roulette wheel selection. For the following results, the GA from the previous results was modified to include the novel normalization technique with a cutoff percentile, $p$, of 0.55, a minimum normalized PI, $m_{\text{min}}$, of 0.05, and a normalization PI at the cutoff, $m_{\text{cutoff}}$, of 0.15. As seen in Figure 6-13 through Figure 6-16, the addition of the normalization step produced substantially quicker convergence. As with any GA, this undoubtedly results in slightly lower exploration of the solution space, but for most of the applications presented here, this is an acceptable trade-off due to the computation time required otherwise.
Increased performance indices, indicating improved performance, were reliably achieved using the GA. As an example of such improvements, the tracking error obtained with a set of heuristically tuned gains (PI=0.86) is presented in Figure 6-17 and the tracking error obtained with the GA-optimized gains (PI=0.96) is presented in Figure 6-18 showing that the position PID tracking errors are greatly reduced with the GA optimized gains.
One side effect of the controller optimization is that a degree of robustness was generally lost. This is one of the traditional drawbacks often experienced with adaptive or robust control: as performance increases, robustness decreases. As mentioned in Section 4.2, the current performance metric does not include any evaluation of robustness. The only ways to quantify this is to incorporate robustness metrics from classical control theory, or to incorporate multiple test flights of varying aircraft health into the performance evaluation. In this research, the latter approach was used; a single GA performance index was computed by taking the average of nominal and failure flight conditions. For the baseline controller for the IMRAC algorithm, each individual in the population was evaluated using a test trajectory at nominal conditions and three different failure conditions of high severity on the three primary control surfaces. The results of this optimization can be seen below in Figure 6-19.

Figure 6-17. Tracking error for a non-optimized position PID controller for 3D s-turns trajectory

Figure 6-18. Tracking error for a GA-optimized position PID controller for 3D s-turns trajectory
Figure 6-19. Robust optimization results for the LQR with integral action controller
7 RESULTS AND ANALYSIS

In order to compare the performance of the various trajectory tracking algorithms presented in this dissertation, a series of simulation tests have been performed using the WVU UAV simulation environment. The IMRAC architecture has been developed by incorporating the best features observed from preliminary analyses of the position PID, outer-loop NLDI, extended NLDI, LQR, AIS suppression mechanism, $L_1$ adaptive control, and MRAC. The preliminary results of these analyses are included in support of the decisions made in the formulation of the IMRAC architecture. The performance of both the AIS suppression mechanism and $L_1$ adaptive control were evaluated; however, it was ultimately determined that both of these controllers were lacking in overall robustness to failure conditions. Drawing from the strengths of the evaluated methods, IMRAC was developed. The preliminary results of these analyses are included in support of the decisions made in the formulation of the IMRAC.

7.1 EVALUATION ARCHITECTURE

A consistent methodology for evaluating the performance of the various trajectory tracking controllers has been developed. All of the performance comparisons presented in this dissertation have been evaluated using the WVU YF-22 research aircraft since the model has been validated with extensive flight data. The experiment design details described herein are with reference to the YF-22 aircraft. A set of trajectories were developed to evaluate the given aircraft over its expected range of operating conditions. All simulation tests have been performed at an initial nominal speed of 77.7 knots and an altitude of 1000 feet. Three simple geometric flight paths, as well as one produced by an obstacle avoidance path generator have been used to compare the performance of the control laws. The geometric paths are a constant-altitude figure-8 (see Figure 7-1), a constant-altitude oval (see Figure 7-2), and a sequence of climbing and descending S-
turns (see Figure 7-3 and Figure 7-4). The path generated from an obstacle avoidance algorithm is presented in Figure 7-5.

Figure 7-1. Figure-8

Figure 7-2. Oval

Figure 7-3. S-turns – horizontal projection

Figure 7-4. S-turns – vertical projection
In addition to the varied flight path geometry, several different factors are also evaluated. Since the main focus of this research is on fault tolerance, it is logical that a series of different faults and failures should be considered. In general, it is desirable to evaluate each controller under the different failures built into the environment. The upset conditions considered include locked aileron, stabilator, and rudder control surfaces, and atmospheric turbulence. In the later IMRAC-specific performance evaluations, modeling uncertainties, which manifest themselves in the linearized lateral and longitudinal aircraft state matrices, will be evaluated.

For the classes of failures considered, it is important to evaluate the effects of the abnormal condition at varying magnitudes. For all failures, a range of mild to high severity is considered. For the main classes of failures considered, the corresponding failures and severities evaluated are given below in Table 7-1.
Table 7-1. Description of upset condition severities

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mild</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>10</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Moderate</td>
<td>9</td>
<td>15</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Severe</td>
<td>12</td>
<td>20</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>25</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

In comparing the performance of the various trajectory tracking controllers, a performance index is used in order to quantify the relative performance of a given test flight. The performance index used for comparison is the same as the one described in Section 4.2. The weights utilized by the performance index are given below in Table 7-2. A maximum error of 100m is considered to be the threshold at which the aircraft is no longer considered to be on-course.

Table 7-2. Performance index weights and normalization cut-offs

<table>
<thead>
<tr>
<th>Trajectory Tracking Performance</th>
<th>Global PI Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>Mean</td>
</tr>
<tr>
<td>XYZ</td>
<td>Z</td>
</tr>
<tr>
<td>XY</td>
<td>50</td>
</tr>
<tr>
<td>W_TT</td>
<td>0.06</td>
</tr>
<tr>
<td>W_RTT</td>
<td>0.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Control Activity Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface Activation Index</td>
</tr>
<tr>
<td>Elevator</td>
</tr>
<tr>
<td>Normalization Cut-off</td>
</tr>
<tr>
<td>W_CA</td>
</tr>
<tr>
<td>W_CA</td>
</tr>
</tbody>
</table>

7.2 **Preliminary Comparison of Existing Controllers**

A thorough performance comparison of the position PID, outer-loop NLDI, extended NLDI, and their AIS-based counterparts was performed in Wilburn et al.\[91\]. This analysis is herein revisited with the addition of the LQR with integral action controller. Additionally, in this analysis, all of the controllers were rigorously optimized using the GA discussed in the previous chapter. For this optimization, a measure of robustness was incorporated into the evaluation of each individual in the population by evaluating the performance of the controller in response to several different flight
conditions consisting of both nominal and failure conditions. In the performance comparison, the four trajectory geometries and various failure operating conditions discussed above were used. For each flight controller, nineteen different conditions were tested for each path; this consisted of one nominal operating condition and five severities for the locked stabilator, aileron, and rudder as well as three severities of atmospheric turbulence. Figure 7-6 below shows the average performance of each controller at these nineteen operating conditions. The important conclusion that can be drawn from this analysis is that the LQR with integral action controller has an overall better performance than the other control architectures. As seen in Figure 7-7, the performance of all of the controllers is very similar for the nominal conditions; however, under most failure conditions, the LQR controller exhibits better robustness. For this reason, the LQR algorithm is used as the baseline controller for the IMRAC architecture. Equally, the need for an adaptive control architecture is evident; significant performance degradations are observed for many of the moderate to severe failures. It is the goal of adaptive control to bring the performance of the aircraft back to that of the nominal case.
The AIS suppression mechanism was used to augment the position PID, outer-loop NLDI, and extended NLDI controllers. Several preliminary analyses, including the early
stages of this research effort, have shown promising results for the AIS-augmented control architecture [82-84, 91]. However, when the GA was used to optimize the AIS-based suppression mechanism controllers, they failed to display the anticipated performance improvements. This statement is illustrated in the comparison of the position PID controller to the AIS-augmented position PID controller. This example can be seen in Figure 7-8 below. Like before, each of the data points presented in this figure is the average of the nineteen failure and nominal conditions at a given trajectory.

Figure 7-8. Average performance index by path for position PID versus AIS-augmented position PID

As can be seen in this figure, the average PI of the optimized position PID is universally higher than the AIS-augmented version. This was also the case when the outer-loop NLDI and the extended NLDI algorithms were augmented with the AIS suppression mechanism. As with any system optimized using a GA technique, it is
possible that local optima were found, and the global optimum may provide different results. However, the results presented herein are indicative of several iterations of the GA optimization. Additionally, the complex interaction between the outer-loop controllers and the general non-linear nature of the aircraft model may be adversely affecting this performance. From this analysis, it was decided that while the AIS suppression mechanism may be useful in many application, using it for augmentation of UAV trajectory tracking controllers is suboptimal.

To investigate the potential benefits of the $L_1$ controller, a comparison was performed between the position PID controller and a position PID controller augmented with $L_1$ adaptive control. Like the previous comparison, the performance of the controllers was evaluated over four different reference trajectories and a multitude of failure conditions. As seen below in Figure 7-9, this augmentation provided nearly universal performance and robustness improvements versus the PPID alone. In this figure, the performance ratings of the controllers are averaged for the four trajectories evaluated. In all but one case, the $L_1$ augmented controller maintained higher robustness as the intensity of the failures were increased. In addition to the results shown here, several different trials were performed where failures were compounded, and the $L_1$ performed respectably, while the non-adaptive controller crashed the aircraft nearly immediately once the failures were initiated.
The results shown here are for a heuristically tuned $\mathcal{L}_1$ controller as the $\mathcal{L}_1$ controller formulation is not very conducive to optimization with the GA. Due to the filter, which is the notable change from MRAC to $\mathcal{L}_1$, the controller runs $1/20^{th}$ the speed of the other algorithms, making GA tuning time-prohibitive. Furthermore, the filter modification requires that $\mathcal{L}_1$ be formulated in terms of transfer functions rather than tunable gains, which would require non-trivial modification of the GA structure or significant derivation of a more conducive formulation of $\mathcal{L}_1$, which could introduce inconsistencies in the method. However, the promise shown by these results indicate that a modified MRAC algorithm could be utilized to provide the adaptation necessary to accommodate failure conditions. This modified MRAC has been described in Section

Figure 7-9. Average total performance for L1 adaptive control
5.2.3 above. Further results will be shown to support the choice of MRAC for this purpose.

7.3 Robust MRAC

The LQR with integral action controller was augmented with MRAC in order to improve its response under failure. Originally, traditional Lyapunov-based MRAC was implemented. As seen in Figure 7-10, the standard MRAC architecture was not capable of dealing with the highly nonlinear nature of the UAV and the associated uncertainties. Like the previous comparison charts, each trajectory in the figure represents the aggregated average of the nineteen failure conditions. For this reason, as described in Section 5.2.3, several robustness and transient performance modifications were made to the basic MRAC algorithm. As seen in the figure, this modified MRAC architecture, referred to as RMRAC, is much more capable of handling the uncertainties and disturbances encountered with UAV trajectory tracking. This improved performance is attributed to the fact that the RMRAC has better accommodation for the inherent unmatched uncertainties that are present in the system. Additionally, the reference model modification presented in Section 5.2.3.3 allows for better transient performance, i.e., quicker adaptation. This is important because, even though both MRAC algorithms are by definition stable under the appropriate conditions, if the transient response is not quick enough, the outer-loop trajectory-tracking controllers will drive the total aircraft response unstable.
Figure 7-10. Average total performance for MRAC derived controllers

Additionally, as shown in Table 7-3, it is important to note that the performance of all three of the controllers is essentially equal, within rounding, under nominal conditions. The subtle differences observed are due to the MRAC controller’s adapting to the inherent uncertainty encountered with the nonlinearity of the UAV equations of motion. Equally, it is important to comment on the definition of good performance for the remainder of this documentation – the goal of MRAC is to restore the performance of the system to that of the baseline controller, LQR, in the event of uncertainty or failure conditions. As such, ideal performance of an MRAC derived methodology, including IMRAC, is never expected, or desired, to surpass the performance of the optimized baseline controller at nominal conditions. Perfect MRAC performance is when the PI is equal to the LQR nominal performance.
The MRAC architecture functions by estimating the parametric uncertainty present in the given system. A simple example of this estimation ability was given in Section 5.2.3 in which a known parametric uncertainty was introduced to the system and the MRAC converged to the values of the uncertainty. In the case of UAV trajectory tracking, the actual uncertainty is not known *a priori*. Nevertheless, the MRAC and RMRAC architectures are still capable of estimating this parametric uncertainty. As seen in Figure 7-11 below, for a static failure or uncertainty, the parameter estimates eventually level off to a near constant value. This example is of the aircraft subjected to an elevator failure of $12^\circ$ at 8 seconds into the simulation. Additionally, as can be seen below in Figure 7-12, if the adaptation gain is not high enough, then the parameter estimates may not converge in a timely manner, and optimal performance may not be restored.
Figure 7-11. Parametric uncertainty estimate for RMRAC with high adaptation rate for an elevator failure

Figure 7-12. Parametric uncertainty estimate for RMRAC with low adaptation rate for an elevator failure
At first inspection it may seem that the adaptation gain should just be increased arbitrarily high such that the parametric uncertainty is estimated quicker. However, in practice there are physical limitation to this. If the algorithm tries to adapt too quickly, it will introduce high frequency oscillations into the system. This coupled with the system dynamics and other outer-loop controllers onboard will eventually force the system to go unstable. Therefore, it is desired to obtain rapid adaptation without the need for excessively high adaptation rates. As discussed earlier, this is the premise of IMRAC. In general, if the particular abnormal condition can be identified, and an MRAC algorithm has already discovered the parametric uncertainty for the scenario, then when the system encounters the same condition again, those estimates can be recalled immediately and thus transient performance gains can be realized.

7.4 IMRAC

The main contribution of this research is the development of the immunity-based MRAC algorithm. The success of this algorithm depends on several different factors. First, the AIS-based detection scheme must be capable of recognizing an abnormal condition. However, if detection is not triggered, the performance of the controller will be the same as the enhanced MRAC algorithm. Fortunately, high detection rates have been demonstrated by prior researchers\[11, 17, 108\].

The performance of the AIS detection scheme developed for the IMRAC algorithm described herein is comparable to that of similar detection schemes\[11, 17, 108\]. To illustrate this, the detection rates for several classes of possible failure conditions are given below in Table 7-4. In this analysis, the detection rate is defined as the percentage of data points that exhibit correct recognition of nominal or abnormal status, false alarm rate is defined as the percent of data points that activate a detector when no failure is present, and activation time is the time it takes the scheme to initially detect that a failure has occurred. In practice, the activation time is the most relevant attribute for the IMRAC
algorithm; this value determines how long it takes the augmentation to kick in after the failure has occurred. Additionally, while 100% detection is ideal, it only takes a very small percentage to trigger the IMRAC algorithm. As seen in the table below, all but three failure conditions, low magnitude aileron failure and moderate to high severity turbulence, are activated within 0.10 seconds or sooner; if turbulence is left out, the average activation time is 0.062 seconds. With the exception of turbulence, which is not adequately detected using this scheme, the detection results presented here are sufficient to not interfere with the timely injection of matured parametric uncertainty estimates.

Table 7-4. IMRAC detection rate

<table>
<thead>
<tr>
<th></th>
<th>Activation Rate</th>
<th>Activation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stuck Aileron 3°</td>
<td>9%</td>
<td>0.155</td>
</tr>
<tr>
<td>Stuck Aileron 6°</td>
<td>37%</td>
<td>0.090</td>
</tr>
<tr>
<td>Stuck Aileron 9°</td>
<td>64%</td>
<td>0.075</td>
</tr>
<tr>
<td>Stuck Aileron 12°</td>
<td>98%</td>
<td>0.065</td>
</tr>
<tr>
<td>Stuck Aileron 15°</td>
<td>99%</td>
<td>0.060</td>
</tr>
<tr>
<td>Stuck Rudder 5°</td>
<td>95%</td>
<td>0.065</td>
</tr>
<tr>
<td>Stuck Rudder 10°</td>
<td>97%</td>
<td>0.045</td>
</tr>
<tr>
<td>Stuck Rudder 15°</td>
<td>99%</td>
<td>0.045</td>
</tr>
<tr>
<td>Stuck Rudder 20°</td>
<td>100%</td>
<td>0.040</td>
</tr>
<tr>
<td>Stuck Rudder 25°</td>
<td>99%</td>
<td>0.040</td>
</tr>
<tr>
<td>Stuck Elevator 3°</td>
<td>98%</td>
<td>0.070</td>
</tr>
<tr>
<td>Stuck Elevator 6°</td>
<td>99%</td>
<td>0.050</td>
</tr>
<tr>
<td>Stuck Elevator 9°</td>
<td>99%</td>
<td>0.050</td>
</tr>
<tr>
<td>Stuck Elevator 12°</td>
<td>98%</td>
<td>0.040</td>
</tr>
<tr>
<td>Stuck Elevator 15°</td>
<td>97%</td>
<td>0.040</td>
</tr>
<tr>
<td>Turbulence Light</td>
<td>99%</td>
<td>0.010</td>
</tr>
<tr>
<td>Turbulence Moderate</td>
<td>100%</td>
<td>0.000</td>
</tr>
<tr>
<td>Turbulence Severe</td>
<td>100%</td>
<td>0.000</td>
</tr>
</tbody>
</table>

As mentioned in the previous section, the IMRAC algorithm serves to speed up the convergence of the parametric uncertainty estimate. For instance, when the example given in Figure 7-11 is repeated with the IMRAC algorithm, the parameter estimates continue where the initial exposure left off. As seen in Figure 7-13 below, the
convergence is essentially instantaneous. The algorithm is free to continue to adapt; however in this case, the parametric uncertainty was already converged. In this example, the naïve IMRAC (for the given detector) algorithm had a PI of 0.9291, and the matured IMRAC had a PI of 0.9404.

![IMRAC Longitudinal parametric uncertainty estimate](image)

**Figure 7-13.** Parametric uncertainty estimate for IMRAC with low adaptation rate for an elevator failure

In order to illustrate the performance of the IMRAC algorithm, a series of examples are provided below. The first example is that of a B matrix uncertainty. In this example, the baseline LQR controller is expecting one system model, but the plant model for the simulation has been altered. This change is implanted manually during the initialization script. This change is constant for the duration of the simulation; i.e, there is no time step like in the control surface or atmospheric conditions. For this example, the lateral control matrix was scaled by a factor of 0.5 to represent modelling error. The performance is analyzed using the Figure 8 trajectory. As seen in Table 7-5, the fixed-parameter LQR crashes early into the trial and thus receives a poor PI. Next it is
important to note that the performance of the RMRAC is identical to that of the naïve (0 exposures) IMRAC. As can be seen in the table, the IMRAC greatly improves the performance of the controller on its first exposure. Additionally, it can be seen that subsequent exposure to the same condition serves to further improve the performance of the controller. In this example, the IMRAC increases the performance of the controller by 2.2%. Note that this improvement is on top of controllers that have already been rigorously optimized using the GA.

Table 7-5. Convergence of the IMRAC algorithm for a control matrix uncertainty

<table>
<thead>
<tr>
<th></th>
<th>PI_TT</th>
<th>PI_CA</th>
<th>PI_Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>LQR</td>
<td>0.0000</td>
<td>0.3229</td>
<td>0.0969</td>
</tr>
<tr>
<td>RMRAC</td>
<td>0.8865</td>
<td>0.8798</td>
<td>0.8845</td>
</tr>
<tr>
<td>IMRAC (0 Exposures)</td>
<td>0.8865</td>
<td>0.8798</td>
<td>0.8845</td>
</tr>
<tr>
<td>IMRAC (1 Exposures)</td>
<td>0.8902</td>
<td>0.9315</td>
<td>0.9026</td>
</tr>
<tr>
<td>IMRAC (2 Exposures)</td>
<td>0.8902</td>
<td>0.9353</td>
<td>0.9037</td>
</tr>
<tr>
<td>IMRAC (3 Exposures)</td>
<td>0.8901</td>
<td>0.9371</td>
<td>0.9042</td>
</tr>
<tr>
<td>IMRAC (4 Exposures)</td>
<td>0.8901</td>
<td>0.9380</td>
<td>0.9045</td>
</tr>
</tbody>
</table>

The second example provided is that of the left rudder stuck at an angle of 20° at 8 seconds. This trial is also executed using the Figure 8 trajectory. As seen in Table 7-6, the results of this trial are not as pronounced as the previous example, however the same trends are present. During the LQR-only trial, the performance is degraded, but the aircraft does not crash. The following trials show the performance of the controller improving with each exposure.

Table 7-6. Convergence of the IMRAC algorithm for a stuck rudder failure

<table>
<thead>
<tr>
<th></th>
<th>PI_TT</th>
<th>PI_CA</th>
<th>PI_Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>LQR</td>
<td>0.8639</td>
<td>0.9222</td>
<td>0.8814</td>
</tr>
<tr>
<td>MRAC</td>
<td>0.8830</td>
<td>0.9343</td>
<td>0.8984</td>
</tr>
<tr>
<td>IMRAC (0 Exposures)</td>
<td>0.8830</td>
<td>0.9343</td>
<td>0.8984</td>
</tr>
<tr>
<td>IMRAC (1 Exposures)</td>
<td>0.8864</td>
<td>0.9364</td>
<td>0.9014</td>
</tr>
<tr>
<td>IMRAC (2 Exposures)</td>
<td>0.8904</td>
<td>0.9366</td>
<td>0.9043</td>
</tr>
<tr>
<td>IMRAC (3 Exposures)</td>
<td>0.8910</td>
<td>0.9382</td>
<td>0.9052</td>
</tr>
<tr>
<td>IMRAC (4 Exposures)</td>
<td>0.8913</td>
<td>0.9383</td>
<td>0.9054</td>
</tr>
</tbody>
</table>
The final detail performance analysis is another uncertainty in the system model. This time the moments of inertia for the system were altered. In this example, $I_{xx}$ was scaled by 0.75, and $I_{yy}$ was scaled by 0.5. This trial was conducted using the obstacle avoidance trajectory. As with the other two trials, the trend is increasing in performance from the LQR baseline controller up to the last provided exposure of IMRAC; the best performance is provided for the final IMRAC exposure.

Table 7-7. Convergence of IMRAC algorithm for uncertain moments of inertia

<table>
<thead>
<tr>
<th></th>
<th>PI_TT</th>
<th>PI_CA</th>
<th>PI_Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>LQR</td>
<td>0.8920</td>
<td>0.8153</td>
<td>0.8690</td>
</tr>
<tr>
<td>MRAC</td>
<td>0.8910</td>
<td>0.8845</td>
<td>0.8891</td>
</tr>
<tr>
<td>IMRAC (0 Exposures)</td>
<td>0.8910</td>
<td>0.8845</td>
<td>0.8891</td>
</tr>
<tr>
<td>IMRAC (1 Exposures)</td>
<td>0.8911</td>
<td>0.9272</td>
<td>0.9020</td>
</tr>
<tr>
<td>IMRAC (2 Exposures)</td>
<td>0.8910</td>
<td>0.9411</td>
<td>0.9060</td>
</tr>
<tr>
<td>IMRAC (3 Exposures)</td>
<td>0.8909</td>
<td>0.9518</td>
<td>0.9092</td>
</tr>
<tr>
<td>IMRAC (4 Exposures)</td>
<td>0.8909</td>
<td>0.9544</td>
<td>0.9099</td>
</tr>
</tbody>
</table>

As demonstrated by these results, the IMRAC controller is not only superior to other controllers in its naïve format (RMRAC) but continues to improve its performance in response to repeated exposures to the same failure.

One of the most promising features of IMRAC is the concept of vaccination. The concept works on two assumptions. First is that similar magnitude failures are often located close together in the solution space. Evidence of this has been shown for the concept of failure identification. This is discussed in greater detail in Moncayo\[11\]. The second assumption is that failures of similar magnitude will have parametric uncertainties that are comparable in value. Based on these two assumptions, it reasonable to expect that even if the system has not encountered particular magnitude failure before (i.e., a naïve detector is activated), a neighboring detector could be activated instead to provide a better chance of survival. This is demonstrated with the following example. For the oval test trajectory, when the aircraft is exposed to a 16° stuck right elevator failure, both the LQR and RMRAC algorithms are unable to adapt.
quickly enough to keep the aircraft from crashing. Likewise, the IMRAC algorithm with no prior experience will also crash. However, if the IMRAC algorithm is first exposed to a smaller magnitude failure of the same nature, in this example a $15^\circ$ stuck stabilator failure, it can survive the otherwise imminent crash with the $16^\circ$ failure. This example can be seen in Table 7-8. A comparison of the baseline LQR and the IMRAC trajectory tracking performance can be found below in Figure 7-14 and Figure 7-15, respectively.

Table 7-8. Example of IMRAC vaccination

<table>
<thead>
<tr>
<th></th>
<th>PI_TT</th>
<th>PI_CA</th>
<th>PI_Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>LQR</td>
<td>0.0000</td>
<td>0.3411</td>
<td>0.1023</td>
</tr>
<tr>
<td>MRAC</td>
<td>0.0000</td>
<td>0.3610</td>
<td>0.1083</td>
</tr>
<tr>
<td>IMRAC (Vaccinated)</td>
<td>0.8658</td>
<td>0.8664</td>
<td>0.8660</td>
</tr>
</tbody>
</table>

This behavior is analogous to vaccination in which an organism is first exposed to weaker threats to train the immune system how to deal with larger threats in the future. While not fully explored in this research, it is anticipated that IMRAC algorithm could be greatly expanded beyond the range of capabilities presented herein given enough exposure to various uncertainties and failures.

Figure 7-14. LQR with integrator subjected to $16^\circ$ right stabilator failure

Figure 7-15. “Vaccinated” IMRAC algorithm subjected to $16^\circ$ right stabilator failure
Finally, a thorough comparison of LQR with integrator, RMRAC, and IMARC is provided. The results from this analysis can be seen below in Figure 7-16. The results from this chart are consistent with the results provided above. In nearly every case, the IMRAC algorithm outperforms the LQR and RMRAC algorithms. This increase in performance is most evident in higher severity failures, such as an aileron stuck at 12° or an elevator stuck at 15°, where the baseline controller begins to falter.

![Figure 7-16. Total performance by path for IMRAC](image-url)
8 CONCLUSIONS AND FUTURE WORK

8.1 CONCLUSIONS

UAV trajectory tracking is a very complex problem due to the wide range of subsystems on an aircraft which can experience failures, uncertainties during the mathematical modeling process, and abnormalities and external disturbance occurring in the environment. Many different types of adaptive control have been attempted in the past, but none to date have achieved a comprehensive solution capable of handling the many different problems that may occur on a UAV.

Of the many adaptive fault-tolerant control techniques which have been developed, the AIS paradigm offers some of the greatest potential in providing a comprehensive solution to this problem. As such, a novel AIS-inspired adaptive control methodology, referred to as IMRAC, has been developed and tested in simulation. In addition to the AIS paradigm, the proposed methodology draws inspiration from other traditional adaptive control techniques. The resulting methodology has been shown to outperform both LQR with integral control and the enhanced MRAC architectures under failure conditions. Like the adaptive immune system, the algorithm learns from prior experiences in order to produce a better reaction to similar problems in the future. From a traditional adaptive control standpoint, this solves the problem of slow transient response often associated with MRAC.

In addition to the primary contribution of this research, which was the IMAC algorithm, several tools and analysis methodologies were developed. From a simulation standpoint, the WVU UAV environment was updated to better deal with the computational demands of the simulations and to add increased modularity. Additionally, the UAVDashboard tool was developed for mission configuration and
visualization purposes. For analysis purposes, a composite PI was developed in order to help better compare the quality of a given controller. This PI was used in quantitatively ranking the various intermediate controllers developed over the course of this research. Finally, for comparing the trajectory tracking performance of different controllers, it is important to ensure that all of the architectures are operating optimally. For this reason, a GA was developed, and then utilized in the subsequent controller comparisons.

To summarize, the contributions of this research include:

- A novel, comprehensive adaptive control framework for UAV trajectory tracking was developed by combining model reference adaptive control techniques with principles from the artificial immune system paradigm.
- The performance and overall effectiveness of this proposed controller was evaluated using the WVU UAV simulation environment.
- In developing this algorithm, a back-propagation neural network was created and trained for the purpose of augmenting the AIS detection scheme.
- A genetic algorithm was developed for the purpose of ensuring optimal trajectory tracking performance, and thus unbiased comparisons between competing controllers.
- A composite performance index for comparing the performance of a given controller was developed.
- The WVU UAV Simulation Environment was updated to allow for greater modularity, more failure conditions, the execution of batch processes, and real-time mission configuration and monitoring.

These contributions serve to provide the framework, supporting architecture, and tools required for the comprehensive, robust, and adaptive controller capable of accommodating surface failures and model uncertainties based on the artificial immune system paradigm. A list of these papers published over the course of this research can be found in Appendix B.
8.2 Future Work

An adaptive control framework was developed using concepts from the biological immune system. Positive results were obtained using this methodology. Nevertheless, further performance and robustness gains may potentially be achieved with modification. Some recommendations for future work are:

- High detection rates are crucial to the operation of the IMRAC algorithm. While the detection rates were in general high for faulty control surfaces, it was not very effective at detecting turbulence failures or low magnitude control surface failures. Thus, any improvements that can be made to the detection process are expected to directly translate to better IMRAC performance. It is anticipated that such improvements to the detection process may be achieved by adjusting the features used for detection, or by altering the collection of self-data. As such, it is recommended that additional features and self data be evaluated for the creation of detector sets.

- The results presented in this dissertation have all been collected using simulation. A high fidelity model has been used for these simulations; the aircraft model and associated failure modes used have been hardware validated. A key strength of MRAC and IMRAC is the ability to adapt for modeling uncertainties. Such uncertainties are simulated in the environment; however, a more accurate assessment of the real-world algorithm performance would be achieved by performing flight tests with actual hardware, where actual model uncertainties would be present. Such trials would help to further illustrate the benefits of IMRAC versus LQR.

- The IMRAC algorithm has been evaluated using several different types of hardware failures. While care has been taken to ensure that many different conditions are evaluated, the list of failure and uncertainty conditions is far from
extensive. As such, it is recommended that a more comprehensive study be performed with more failure conditions.

- Several modifications have been made to the basic MRAC structure to improve the robustness of the algorithm. However, there are still instances, namely turbulence or noise-based disturbances, where the algorithm exhibits poor robustness. Therefore, it is recommended that further robustness modifications be made to the algorithm presented herein. It is likely that such robustness improvements may be achieved using classical noise rejection techniques from robust control theory.

- The MRAC and IMRAC approaches can be used to add an adaptive component to any number of baseline controllers. In the research presented here, the LQR with integral action controller was used. However, further improvements to the overall performance of the system may be obtained by using a more intricate baseline controller. One should note that the LQR baseline controller was in part chosen because of its convenient integration into the MRAC control law derivation. Incorporating certain more elaborate controllers such as the extended NLDI into the MRAC architecture would be nontrivial.

- The choice of regressor vector plays a central role in the performance of the MRAC and IMRAC algorithms. The algorithm presented herein uses a quasi-standard formulation of the regressor vectors. However, as referenced in Section 5.2.3, the choice of this regressor vector has been a much studied, and often improved upon, topic. As such, overall performance gains may be obtained by choosing another regressor vector. For example, RBF-based regressor vectors have been shown to be more capable of representing a broader range of uncertainty in a system in many situations.

- Key concepts of the biological immune system have been used as inspiration for the IMRAC architecture. However, direct parallels were not established with all
of the various behaviors of the immune system. While not guaranteed, there is a moderate possibility that the IMRAC algorithm could be further improved by strengthening the biological analogy and perhaps extending the core functionality.

- As mentioned in Chapter 6, the GA has a tendency to optimize for performance at the expense of robustness. To mitigate this problem, multiple failure trials were incorporated into a singular performance index during the GA optimization process. This does in fact help to improve the robustness of the controllers obtained using the GA. However, it is anticipated that more rigorous metrics derived from classical robust control could be incorporated into the PI to obtain a higher robustness more efficiently.
9 BIBLIOGRAPHY

doi: 10.1155/2010/736201


Togayev, A., "Immunity-Based Accomodation of Aircraft Subsystem Failures," West Virginia University, 2014.

Benjamin, E.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Title</th>
</tr>
</thead>
</table>


A – SIMULINK MODELS
9.1 Robust MRAC

Figure A-1. Outer-loop, inner-loop control architecture

Figure A-2. RMRAC Inner-loop controller

Figure A-3. Inside the RMRAC module
Figure A-4. RMRAC control law

Figure A-5. RMRAC reference model

Figure A-6. RMRAC adaptive law
Figure A-7. Projection operator

Figure A-8. $\mu$ operator
9.2 IMRAC

Figure A-9. IMRAC inner-loop controller

Figure A-10. Inside the IMRAC control module
Figure A-11. IMRAC control law

Figure A-12. IMRAC reference model

Figure A-13. IMRAC adaptive law
Figure A-14. IMRAC projection operator

Figure A-15. ANN normalization module

Figure A-16. ANN activation module

Figure A-17. Bipolar sigmoid function
Figure A-18. ANN inverse normalization module
Figure A-19. IMRAC detection and parametric uncertainty initialization
B – RELEVANT PUBLICATIONS


