Learning and Evolving Flight Controller for Fixed-Wing Unmanned Aerial Systems

©2020

Daksh Shukla

Submitted to the graduate degree program in Department of Aerospace Engineering and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Committee members

Dr. Shawn Keshmiri, Chairperson

Dr. Nicole M. Beckage, Co-Chair

Dr. Mark Ewing

Dr. Emily Arnold

Dr. Craig McLaughlin

Dr. Sara Wilson, External Reviewer

Date defended: May 8, 2020

The Dissertation Committee for Daksh Shukla certifies
that this is the approved version of the following dissertation :

Learning and Evolving Flight Controller for Fixed-Wing Unmanned Aerial Systems

Dr. Shawn Keshmiri, Chairperson

Date approved: May 8, 2020

Abstract

Artificial intelligence has been called the fourth wave of industrialization following steam power, electricity, and computation. The field of aerospace engineering has been significantly impacted by this revolution, presenting the potential to build neural network-based high-performance autonomous flight systems. This work presents a novel application of machine learning technology to develop evolving neural network controllers for fixed-wing unmanned aerial systems.

The hypothesis for an artificial neural network being capable of replacing a physics-based autopilot system consisting of guidance, navigation, and control, or a combination of these, is evaluated and proven through empirical experiments. Building upon widely use supervised learning methods and its variants, labeled data is generated leveraging non-zero set point linear quadratic regulator based autopilot systems to train neural network models, thereby developing a novel imitation learning algorithm.

The ultimate goal of this research is to build a robust learning flight controller using low-cost and engineering level aircraft dynamic model and have the ability to evolve in time. Discovering the limitations of supervised learning methods, reinforcement learning techniques are employed to learn directly from data, breaking feedback correlations and dynamic model dependence for a control system. This manifests into a policy-based neural network controller that is robust towards un-modeled dynamics and uncertainty in aircraft dynamic model. To fundamentally change flight controller tuning practices, a unique evolution methodology is developed that directly uses flight data from a real aircraft: factual dynamic states and the rewards associated with them, in order to re-train a neural network controller.

This work has the following unique contributions:

1. Novel imitation learning algorithms that mimic “expert” policy decisions using data aggre-

gation are developed, which allow for unification of guidance and control algorithms into a single loop using artificial neural networks.

2. A time-based and dynamic model dependent moving window data aggregation algorithm is uniquely developed to accurately capture aircraft transient behavior and to mitigate neural network over-fitting, which caused low amplitude and low frequency oscillations in control predictions.

3. Due to substantial dependence of imitation learning algorithms on “expert” policies and physics-based flight controllers, reinforcement learning is used, which can train neural network controllers directly from data. Although, the developed neural network controller was trained using engineering level dynamic model of the aircraft with low-fidelity in low Reynold’s numbers, it demonstrates unique capabilities to generalize a control policy in a series of flight tests and exhibits robustness to achieve the desired performance in presence of external disturbances (cross wind, gust, etc.).

4. In addition to extensive hardware in the loop simulations, this work was uniquely validated by actual flight tests on a foam-based, pusher, twin-boom Skyhunter aircraft.

5. Reliability and consistency of the longitudinal neural network controller is validated in 15 distinct flight tests, spread over a period of 5 months (November 2019 to March 2020), consisting of 21 different flight scenarios. Automatic flight missions are deployed to conduct a fair comparison of linear quadratic regulator and neural network controllers.

6. An evolution technique is developed to re-train artificial neural network flight controllers directly from flight data and mitigate dependence on aircraft dynamic models, using a modified Deep Deterministic Policy Gradients algorithm and is implemented via TensorFlow software to attain the goals of evolution.

Acknowledgements

I would like to thank National Aeronautics and Space Administration (NASA), DARcorporation (Design Analysis and Research) at Lawrence, Lockheed Martin Corporation, Heising-Simons Foundation and Paul G. Allen Foundation for funding various projects during the tenure of my PhD. A special thanks to NASA for providing their support towards my PhD research project. I appreciate the support of these organizations and would like to thank them for providing the opportunities to pursue advanced and cutting-edge research in the field of aerospace engineering that directly supported my PhD.

Dr. Shawn Keshmiri, my PhD advisor and I started on this exciting research journey in the fall of 2016. At that time, Dr. Keshmiri talked about his vision with me in a casual meeting and said "*Daksh, I envision your work, a controller that learns on its own. A silent layer running in background hiding behind a standard autopilot and improving itself with each flight test*". The research idea at that time seemed to be impossible to manifest into a tangible technology and very complex to achieve (almost science fiction). But today I can safely say that his vision not only succeeded to the extent that it can be quantified to something that I am proud of, but also helped me learn and grow tremendously and realize my potential to do complex research projects. I cannot thank my advisor enough for his consistency throughout my PhD years in pushing me towards the right direction, being extremely patient with me and helping and supporting me at every step of my research. The skills that I developed under his supervision and the experience that I gained in his laboratory are invaluable, and I will always be indebted to him for providing me with such great opportunities, putting his faith in me when I started this research with very limited knowledge.

Dr. Nicole Beckage, my co-advisor and Research Scientist at Intel Labs, has guided me throughout my doctoral program and immensely helped me achieve difficult and unapproachable goals by innovating amazing ideas. Our regular meetings and long discussions helped me create

an intuition for fundamental machine learning research. Her excellent advice on writing and organization helped me develop the technical skills that I needed to complete my work. I am extremely grateful to Dr. Beckage for supporting me throughout my thesis work and taking huge amounts of time from her schedule to mentor me.

I would like to thank Dr. Richard Hale, Professor and Chair, Department of Aerospace Engineering for his continued support. Through his excellent leadership, he ensures that students have the opportunity to meaningfully participate in the ambitious research activities of the department. Dr. Mark Ewing's support towards various projects conducted at KU's flight research laboratory is really appreciated. Under his excellent guidance and outstanding management our laboratory has achieved many research goals and his direction has lead us to secure multiple projects. I really appreciate the support and feedback from my committee members, Dr. Emily Arnold, Dr. Craig McLaughlin and Dr. Sara Wilson. I have enjoyed their courses and have learned a great deal from each of them. Ms. Amy Borton, Administrative Associate in the Department of Aerospace Engineering has been of immense support, helping students keep on track from behind the scenes.

I would also like to thank Dr. Heechul Yun, Associate Professor, KU-EECS for his advice and support during the development of the onboard autopilot software. The research collaboration with him has always played a crucial role in developing advanced flight test software and simulation testing platforms. Many other professors whose courses shaped my research deserve mention, including Dr. Suzanne M. Shontz, Dr. Huazhen Fang, Dr. Zsolt Talata, and Dr. Martin Kuehnhausen.

I would also like to express my gratitude towards my team members at KU's flight research laboratory without whom these projects would have been impossible. My amazing, extremely knowledgeable and intellectual team members have helped me achieve my research and flight test goals throughout the tenure of my PhD. Aaron Blevins and Dustin Hauptman deserve my special thanks for helping develop Skyhunter hardware, avionics, and autopilot ground control station software. I am highly grateful to Aaron Mckinnis for his consistent support towards flight tests. Alex Zugazagoitia, our team's UAS pilot has done an amazing job flying through several of my

"stable" autopilot designs, and somehow has always managed to outperform my neural network flight controllers, I thank him for his support. I am thankful to Thomas Le Pichon, Hady Benyamen, Grant Godfrey, Kenton Dreiling, Mira Wilson and Dr. Alexander David Earl for their flight test support.

My wife, Alexis J. Fekete-Shukla, perhaps deserves the most significant acknowledgment due to her patience over the last few years. She not only helped me emotionally to cope with the stressful times during my PhD but most surprisingly also proved to be a great partner to discuss technical aspects of my research work. Her thorough feedback on my dissertation is greatly appreciated.

I would like to thank my family for supporting me throughout my career and always encouraging me to pursue my academic ambitions. My mother, Renu Shukla is an amazing person; she has been my source of inspiration and a personal counselor in the times of difficulty, where she has always managed to convince me of all the positivity and good in the world. She has motivated and encouraged me for my whole academic and professional career. Writing this with a heavy heart, I am extremely thankful to my late father, Harsha Shukla for his constant support, practical guidance and encouragement, I wish he knew of my academic and career achievements. I would like to thank my sister, Ridhima Shukla for listening to my nerdy science stories and trying very hard to show that she understands them. I am grateful to my family and friends who encouraged me to pursue graduate studies in the United States, especially my uncles, Rakesh Sharma and Sanjay Sharma, my mentors Dr. M.P. Poonia and Ms. Richa Rajput, and my dear friends Suvarnalata Xanthate Duggirala and Dr. Sumiti Saharan. As my father once said, we live in a society characterized by complex relationships and human interactions, which contribute to what we become, our failure, success or just our everyday life; I express my deepest gratitude to all my mentors, family and friends that have affected my career in some way and have contributed towards my successful endeavor of completing a doctorate in aerospace engineering.

Contents

1 Introduction **1**

1.1 Motivation 5

1.2 Why Artificial Neural Networks 7

1.3 Literature Review 8

1.4 Unique Contributions 12

2 Theory and Background **14**

2.1 Guidance, Navigation and Control 14

2.2 Aircraft Equations of Motion 15

2.3 Base Autopilot 19

2.3.1 Guidance 20

2.3.2 Control 23

2.4 GNC to Artificial Neural Networks 28

2.5 Artificial Neural Networks: Training 30

2.5.1 Back-Propagation and Gradient Descent 31

2.5.2 Scaled Conjugate Gradient Descent Algorithm 33

2.5.3 The Adam Algorithm 35

2.6 Imitation Learning: Data Aggregation Set 36

2.6.1 Imitation Neural Network Autopilot 38

2.6.2 DAgger for Fixed-Wing Aircraft 42

2.6.3 Simulation Setup and Data Aggregation Models 44

2.7 Moving Window DAgger: Addressing Oscillations 53

2.8 Grid Search for Optimal Neural Network Architectures 54

2.9 Reinforcement Learning 56
2.9.1 Preliminaries 57
2.9.2 Deep Q-Network 59
2.9.3 Policy Gradients 60
2.9.4 Deep Deterministic Policy Gradients 62

3 Aircraft Platform, Hardware and Software Development 66

3.1 Skyhunter Aircraft 66
3.2 Skyhunter 6-DOF Equations 68
3.3 Skyhunter LTI Model 70
3.4 Monte-Carlo Simulations: Off-Nominal Initial Condition Flights 71
3.5 Aircraft Avionics 73
3.6 Onboard ROS-Autopilot Software Architecture 76
3.7 Evolution: Learning From Flight Data - TensorFlow Software 79

4 Validation, Verification, and Discussions 82

4.1 Base Autopilot - LQR Controller 82
4.2 Imitation Autopilots Trained via Moving Window DAgger 86
4.2.1 Imitation Model 1: Unification of Guidance, Navigation and Control 86
4.2.2 Imitation Model 2: Unification of Guidance and Control 90
4.2.3 Imitation Model 3: Decoupled Longitudinal and Lateral Controllers 94
4.3 DDPG Trained Neural Network 100
4.3.1 Longitudinal Environment: Aircraft Model and Reward Function 101
4.3.2 Actor and Critic Neural Networks - Longitudinal Controller 104
4.3.3 Training Results - Longitudinal Controller 106
4.3.4 LTI Simulations - Longitudinal Neural Network Controller 107
4.3.5 Lateral Environment: Aircraft Model and Reward Function 108
4.3.6 Actor and Critic Neural Networks - Lateral Controller 111

4.3.7 Training Results - Lateral Controller 112
4.3.8 LTI Simulations - Lateral Neural Network Controller 113
4.4 Flight Test Validation - Longitudinal Neural Network Controller 115
4.4.1 Flight Scenario 1.9: Extreme Wind Conditions 121
4.4.2 Flight Scenario 2.5: Airspeed Command 124
4.4.3 Flight Scenario 3.6: Altitude Command 126
4.4.4 Flight Scenario 2.1 and 3.1: Flight Test Comparison LQR vs ANN 129
4.5 Neural Network Evolution using Flight Test Data 133

5 Conclusions **137**

List of Figures

2.1 Navigation → Guidance → Control: General Architecture 14

2.2 Base Autopilot 19

2.3 Roll Angle Attitude Guidance Logic 20

2.4 Pitch Angle Attitude Guidance Logic 21

2.5 Non Zero Set Point Lateral Controller 26

2.6 Artificial Neural Network: General Architecture 28

2.7 GNC to Deep Neural Network 30

2.8 Guidance, Navigation, and Control Replaced by an Artificial Neural Network Autopilot 38

2.9 ANN architecture 39

2.10 Data Aggregation Concept: Novel Data and Error Propagation 42

2.11 All models are evaluated in a simulated flight test setting to follow the dashed desired waypoint path, and the corresponding aircraft trajectories are compared. . . 44

2.12 Aircraft Lateral states comparison for first 4 models. Left column shows roll angle (ϕ) and right column shows yaw angle (ψ). 46

2.13 Aircraft Longitudinal states comparison for all models. Left column shows total velocity (V_T) and right column shows pitch angle (θ). 47

2.14 Flight time slopes for DAgger training, comparison for all models. 48

2.15 Sequential DAgger: Intermediate trajectories flown by ANN during data aggregation. 49

2.16 GNC McDAgger: Intermediate trajectories flown by ANN during training: ANN Learning Curves 51

2.17 Reinforcement Learning Problem Framework 57

3.1 Skyhunter UAS Picture 66

3.2 AAA Model: Top, Front, and Side Views 67

3.3 Skyhunter Avionics Setup 75

3.4 ROS Onboard Software Architecture: HiTL and Flight Modes 77

3.5 ROS Graph: Communication and Data Exchange Among the Autopilot Processes (nodes) 79

4.1 Trajectory Comparison: Expert Policy and ANNs: DAgger, McDAgger, GNC McDAgger and MwDAgger 87

4.2 Moving Window DAgger ANN-Simulation Flight Test: Lateral States and Controls 88

4.3 Moving Window DAgger ANN-Simulation Flight Test: Longitudinal States and Controls 89

4.4 Moving Window DAgger - 3-Dimensional Trajectory Comparison with GNC . . . 90

4.5 Imitation Model 2: Unification of Guidance and Control via Neural Network . . . 92

4.6 Moving Window DAgger - Trajectory Comparison Imitation Model 2 and GNC . . 93

4.7 Imitation Model 3 Training Concept 94

4.8 Imitation Model 3 96

4.9 Imitation Model 3: 2D Trajectory Tracking Figure-8 97

4.10 Imitation Model 3: Lateral Neural Network - Trajectory Tracking 98

4.11 Imitation Model 3: Lateral Neural Network - Altitude Tracking 99

4.12 Longitudinal Neural Network Controller Architecture 104

4.13 Critic Neural Network Architecture: For Longitudinal Controller 105

4.14 Cumulative Rewards during training: Longitudinal Neural Network 106

4.15 Longitudinal Neural Network Controller, Closed-Loop LTI Simulation Results: States 107

4.16 Longitudinal Neural Network Controller, Closed-Loop LTI Simulation Results: Control Variables and Rates 108

4.17 Lateral Neural Network Controller Architecture 111

4.18 Critic Neural Network Architecture: For Lateral Controller 112
4.19 Cumulative Rewards During Training: Lateral Neural Network 113
4.20 Lateral Neural Network Controller, Closed-Loop LTI Simulation Results: States . . 114
4.21 Lateral Neural Network Controller, Closed-Loop LTI Simulation Results: Control Variables and Rates 115
4.22 Aircraft Software-System Architecture: Flight Test 116
4.23 Airspeed and Altitude Data Distributions for all Flights with DDPG Trained Neural Network Longitudinal Controller 117
4.24 Throttle and Elevator Controls Data Distributions for all Flights with DDPG Trained Neural Network Longitudinal Controller 118
4.25 Flight Scenario 1.9: Longitudinal States for Flight Test Under Extreme Wind Condition 122
4.26 Flight Scenario 1.9: Lateral States for Flight Test Under Extreme Wind Condition . 123
4.27 Flight Scenario 1.9: 2D Trajectory for Flight Test Under Extreme Wind Condition . 123
4.28 Flight Scenario 2.5: Longitudinal States for Flight Test with Different Airspeed Commands 124
4.29 Flight Scenario 2.5: Three dimensional trajectory - three different airspeed commands 125
4.30 Flight Scenario 2.5: Lateral States for Flight Test with Different Airspeed Commands 125
4.31 Flight Scenario 2.5: 2D Trajectory for Flight Test with Different Airspeed Commands 126
4.32 Flight Scenario 3.6: Longitudinal States for Flight Test with Different Altitude Commands 127
4.33 Flight Scenario 3.6: Three dimensional trajectory - two different altitude commands 127
4.34 Flight Scenario 3.6: Lateral States for Flight Test with Different Altitude Commands 128
4.35 Airspeed Command, Flight Scenario 2.1: 2D Trajectory, Airspeed Commanded and Measured Values, and Altitude Commanded and Measured Values 130

4.36 Altitude Command, Flight Scenario 3.1: 2D Trajectory, Altitude Commanded and Measured Values, and Airspeed Commanded and Measured Values 131
4.37 Airspeed and Altitude Errors for Varying Airspeed and Altitude Commands, respectively 132
4.38 Throttle and Elevator Controls Comparison 133
4.39 Cumulative Rewards Averaged Over 100 Test Episodes with Different Initial Conditions, During Re-Training of Longitudinal Neural Network from Flight Data . . . 135
4.40 Airspeed and Altitude States Comparison for Original and Evolved Longitudinal Neural Network: 6-DOF Simulations 136

List of Tables

2.1 Comparison of ANN autopilots trained using different imitation learning paradigms. Model evaluation focuses on training feasibility and flight time during validation and testing. 53

3.1 Moment of Inertia Estimates from Bifilar Pendulum Tests 67
3.2 Aircraft Main Wing Geometric Parameters 68
3.3 Ranges and Intervals for Aircraft States Used to Generate Initial Conditions for Monte-Carlo Flight Simulations 72
3.4 Ranges for Aircraft States, Controls and Control Rates Used to Test a Success or Failure Case for a Monte-Carlo Flight Simulation 73

4.1 Lateral Controller Closed-Loop System Properties 83
4.2 Longitudinal Controller Closed-Loop System Properties 83
4.3 Success and Failure Rates for Monte-Carlo Flight Test Simulations Conducted on the Base Autopilot 84
4.4 Base Autopilot Lateral Controller Gain Matrix 84
4.5 Base Autopilot Lateral Controller Gain Tuning: Flight Test 85
4.6 Base Autopilot Longitudinal Controller Gain Matrix 85
4.7 Base Autopilot Longitudinal Controller Gain Tuning: Flight Test 85
4.8 Imitation Model 1 - Training Statistics 87
4.9 Success and Failure Rates for Monte-Carlo Flight Test Simulations Conducted on Imitation Model 3 97
4.10 General Flight Test Statistics: DDPG Longitudinal Neural Network 117

4.11 Flight Test Scenarios with Varying Altitude, Airspeed and Wind Conditions: DDPG Longitudinal Neural Network 119
4.12 Flight Test Scenarios with Airspeed Command Maneuvers: DDPG Longitudinal Neural Network 119
4.13 Flight Test Scenarios with Altitude Command Maneuvers: DDPG Longitudinal Neural Network 120
4.14 Flight Test Statistics: DDPG Longitudinal Neural Network vs LQR Base Autopilot 129
4.15 DDPG Longitudinal Neural Network Trained using Flight Data 134
4.16 DDPG Longitudinal Neural Network Trained using Flight Data: Monte-Carlo Simulations 135

Chapter 1

Introduction

Artificial intelligence (AI) is a fast growing field that attempts to develop autonomous cognitive systems without the use of formal, explicit, standard programming practices that distinctly define decision making processes through a set of rules, protocols or mathematical functions. AI leverages the power of machine learning (ML) and the recently discovered potential of deep learning techniques using highly nonlinear and deep artificial neural networks (ANN) with a huge number of parameters, to realize this goal. AI has been named as the "fourth wave" of the industrial revolution that has directly impacted our daily lives, finding applications in agriculture [127, 46, 98, 45, 87], healthcare [81, 29, 7, 48, 92, 147, 71], natural language processing [144, 106, 35, 105], finance [61, 139, 41, 37], etc. The aviation industry has also been significantly impacted by this revolution. Machine learning has found numerous applications in aircraft design [99, 113, 9, 8, 97], defense [47], autopilot systems [18, 17, 26, 14, 73, 27, 13, 33], etc.

Advancements in autopilot systems are driven by increased use of autonomous unmanned aerial systems (UAS) in the field of aerospace engineering. UAS or drones are becoming an active and rapidly growing area of research among the scientific and engineering community due to their emergent demand for many civilian applications such as precision agriculture, remote sensing, aerial 3D mapping or geological surveying, hurricane hunting, search and rescue, etc. [123, 152, 90, 49]. Both fixed-wing [90] and rotor-craft UAS [49] are finding applications in these areas. One of the greatest challenges that lies in front of engineers is to tackle the problem of designing robust guidance and control algorithms for drones, especially for fixed-wing aircraft. Fixed-wing aircraft must maintain a minimum speed and have many structural and aerodynamic constraints, characterizing highly nonlinear dynamics that are further amplified by effects from

external weather conditions and noise on the onboard devices and sensors.

The goal of this research is to develop evolving flight controllers that directly train neural networks from real flight data, specifically for fixed-wing aircraft. To begin with, this work tests the hypothesis of replacing a standard autopilot system with an artificial neural network. In this foundational step, neural network autopilot models are trained using supervised learning techniques by leveraging existing standard autopilot systems. An imitation learning framework known as the data aggregation set (DAgger [118, 119]) algorithm is utilized, which aims to mimic a unified GNC system to fly a fixed-wing UAS. Utilizing nonlinear six-degrees-of-freedom (6-DOF) aircraft simulations, it is shown that several modifications in the existing imitation learning techniques must be considered. DAgger algorithm when applied sequentially to augment data along the desired flight trajectory to train the neural network autopilot is unable to generalize across turning and straight-line maneuvers, and hence cannot learn to fly stably, eventually terminating the simulation. Monte-Carlo methods when incorporated into the DAgger algorithm allow for sampling of random data along the desired flight trajectory, proved to be effective to train the neural network autopilot to fly indefinitely with acceptable tracking errors, but with low frequency oscillations in aircraft states. The oscillatory behavior is dealt with by introducing a time-based moving window (Mw) along the trajectory for data addition in conjunction with the standard DAgger algorithm (MwDAgger). All these models are tested and compared using closed-loop 6-DOF flight simulations, in which 3-dimensional (3D) trajectory tracking and stability in states are evaluated with evidence supporting the idea that a neural network autopilot can behave as a unified GNC system and fly a fixed-wing UAS.

After obtaining empirical evidence that a neural network can mimic guidance, and control, in simulated flight tests, this research explores the goal of learning directly from environment: reinforcement learning. A controller is the most complex, important, and integral part of autopilot systems. In this phase of the research, the hypothesis of replacing a flight controller with an artificial neural network is tested, with an ambitious goal of directly learning from data and evolving the flight controller with subsequent flight tests. Thus a controller is formulated and parameterized

using artificial neural networks that are trained utilizing reinforcement learning techniques. The greatest challenge in using a learning flight controller onboard an aircraft is that its nonlinear and unpredictable training nature raises safety concerns that make it difficult to certify the autopilot. Federal Aviation Administration (FAA) approval requires a high degree of assurance on the air-worthiness of the onboard guidance, navigation and control software, which can only be met by deterministic controllers. Verification of adaptive and learning control techniques is at the fore-front of the aircraft safety assessment and certification process. A straightforward solution to this problem is to fly the aircraft with two flight controllers running in parallel: a standard GNC system and an evolving neural network controller. The standard GNC software runs as a backup system in the mainstream autopilot software while the neural network is in control of the aircraft. This soft-ware is augmented with a safety monitoring algorithm that tracks and observes the neural network controls in real-time (and the corresponding predicted aircraft states based on a time horizon), and raises a safety-critical flag if the aircraft is on the verge of becoming unstable, thereby switching the control from the learning autopilot to the standard GNC. The safety switching algorithm was developed in collaboration with the "Robotics Laboratory" at the Kansas State University.

An artificial neural network longitudinal controller for a fixed-wing UAS is trained offline via reinforcement learning using the Deep Deterministic Policy Gradients (DDPG) learning algorithm [88]. The training is performed using a linear time invariant (LTI) decoupled longitudinal model of the aircraft. The neural network controller is tested and validated using LTI and full nonlinear 6-DOF models of the aircraft. It is shown that the neural network controller performs well under significant noise and disturbance inputs, and is able to generalize its outputs when tested with the 6-DOF model of the aircraft. Several validation and verification flight tests with the DDPG trained longitudinal neural network controller are conducted in varying wind conditions using the Skyhunter UAS platform. Flight test is performed using a backup LQR (linear quadratic regulator) controller together with the safety monitor switch and it is shown that the switching takes place safely. The backup LQR controller is part of a "base autopilot" system that runs the whole package of standard GNC algorithms and is capable of performing fully autonomous flights.

The base autopilot is tuned and tested thoroughly using the full 6-DOF aircraft model. A total of 100,000 Monte-Carlo simulation flight tests were performed on the base autopilot with a broad range of random initial conditions in terms of aircraft states and its inertial position. In all these simulated flights, the aircraft states, controls and control rates are evaluated against specific and safety-critical threshold criteria in which it is shown that 99,690 flights (99.69%) passed this evaluation of stability and boundedness. The decoupled LQR controllers for both lateral and longitudinal dynamics are evaluated using the LTI closed-loop aircraft models. It is found that the LQR controller has stable (negative) poles, and all modes are sufficiently damped, satisfying a level I handling quality requirement as specified in MIL-SPEC C 8785.

During initial flight tests, the LQR controller fails to fly the aircraft stably (real aircraft, not simulation), and imparts unstable lateral and longitudinal oscillations. The LQR controller gains are tuned during the flight several times to achieve a reliable and robust performance from the autopilot. In contrast, the DDPG trained neural network controller when activated during the real flight test exhibits excellent performance and does not require any tuning nor re-training to acclimate to the real environment. The DDPG neural network exceeds in performance as compared to the LQR controller in terms of tracking of guidance commands. Not only is the DDPG neural network able to generalize and adapt to a realistic environment, it is able to perform stable flights in the presence of high wind conditions in an unstructured environment.

The DDPG longitudinal neural network flight tests are subjected to climb and descend maneuvers for which the ANN was not trained in the DDPG-LTI simulation environment. The desired altitude is changed dynamically while the longitudinal neural network controller is flying the aircraft autonomously. Note that the LTI model used for training the neural network controller is only designed around steady-state straight line cruise condition and therefore any controller designed using this model is only valid for small state and control value perturbations around that trim point. However, the neural network controller is able to follow the altitude commands (via the longitudinal guidance) while generalizing control outputs and stably climbing and descending to the desired altitude. Also, the desired airspeed is commanded dynamically while the aircraft is

in autopilot mode and is flying using the longitudinal neural network controller. The flight tests show that the DDPG trained neural network is able to follow dynamically changing airspeed and pitch commands, which are no longer small perturbations around the trim point and still is able to generalize the control outputs which are stable, not aggressive and safe.

Using the DDPG reinforcement learning algorithm, a lateral neural network controller is also developed. A unique methodology involving the scheduling of reward function, based on roll angle and roll rate ranges, is developed to train the lateral neural network controller. This controller is evaluated using closed-loop LTI simulations and is also subjected to noise. The performance of the lateral neural network controller is found to be satisfactory with stable control outputs and bounded control rates.

1.1 Motivation

Autonomous UAS principally involve pre-programmed guidance, navigation and control (GNC) algorithms. An intelligent GNC system should entail adaptive characteristics that allow the UAS to take proper decisions in case of a system failure or off-the-norm flight conditions (wind, gust, turbulence, etc.). Most of the controllers for an aircraft are designed using linearized and decoupled dynamics of its six-degrees-of-freedom equations of motion (EOM) [116, 136, 83]. The system dynamics are often linearized around desired trim points that entail specific flight conditions like steady-state rectilinear flight, level turn and symmetrical pull-up. Therefore, the controllers work well as long as the aircraft characteristics do not change substantially with respect to its trim point. However, nonlinear aerodynamics due to external disturbances such as wind, gust, etc, together with the uncertainties in dynamic model and un-modeled dynamics can degrade the performance of controllers that rely heavily on aircraft physics-based dynamic models. In a manned aircraft, skilled pilots can control the aircraft even under abnormal flight conditions; they have the ability to compensate for nonlinearity and react appropriately. As stated previously, many controllers rely heavily on physics-based models and are pre-programmed, therefore a flight controller is only as good as the reliability and accuracy of an aircraft model. Developing high-fidelity aircraft

dynamic models is very expensive and the progress is usually slow [136]. Such efforts demand for large financial investments and special facilities. A simple question that confronts this, is: can we design a flight controller that is capable of generalizing (learning) over a large range of trim conditions from a low-fidelity, low-cost aircraft dynamic model and still perform on a real aircraft under extreme external disturbances? This thesis attempts to address this question and validate a flight controller through real flight tests over a broad flight envelope.

Standard control practices used for designing a flight controller involve classical single-input-single-output (SISO) techniques, gain scheduling, linear parameter varying, dynamic inversion, feedback linearization, model predictive and adaptive control. SISO techniques are not practical for aircraft involving coupled unstable dynamic modes, and especially they are not applicable to different trim points. Gain scheduling is a widely used technique for designing aircraft controllers at different trim points; however, there is no systematic procedure for selecting the scheduling variables [55] and stability is not guaranteed at other than the selected trim points. Linear parameter varying [16] attempts to express the nonlinear model of an aircraft at different equilibrium points based on first order linear approximations. However, these models are valid around the trim points selected and hence the controllers work well only around the acceptable trim region [93]. Techniques such as dynamic inversion and model predictive control directly and heavily rely on the accuracy of the aircraft dynamic model. The main driving motivation in this work is to mitigate the need for high-fidelity physics-based aircraft dynamic model, and develop a method that adapts a neural network control system across varying trim conditions while preserving previously learned generalizations, and thus evolving the flight controller.

Designing autonomous UAS, fully capable of performing intelligent guidance and control is one of the highly challenging tasks in the aerospace engineering field. In recent years, there has been a tremendous increase in autonomous UAS; however, pre-programed flight controllers currently used in UAS cannot operate over a broad range of trim conditions, fail to react to unforeseen failures, and more importantly, the whole design process must be repeated for different UAS platforms. This absence of adaptability is rooted in the lack of intelligence in current flight control

systems whereas in contrast, a skilled pilot can acclimatize to different flight conditions, safely perform wide range of maneuvers at the edge of aircraft flight envelope, compensate for aerodynamics and propulsive nonlinearity, and even improvise during system failures. While pilots learn and sharpen their skill-sets through several years of flight experience, current autopilot systems lack such learning ability. This research aims to develop evolving flight control systems that can learn from each flight experience and ultimately perform at a comparable or better level than skilled human pilots. The goal is to reduce the dependency of controllers on the quality of physics-based dynamic models of aircraft and to develop machine learning algorithms that can be used for control while adding evolving intelligent features towards designing a fully autonomous UAS capable of flying over a broad range of flight envelope and that gets better after each flight test experience.

Robustness and performance are the most important required characteristics for a fixed-wing flight controller, especially for flights in unpredictable, uncertain and unstructured environments. Nature often guarantees the presence of these uncertainties together with exogenous inputs and disturbances. In the absence of ability to predict such disturbances and their interactions with the aircraft aerodynamics, robustness of controller and achieving good permanence becomes a major challenge. Therefore, in order to ascertain a reliable flight control system it becomes intrinsic to this problem to introduce adaptive controllers that have the capability to interact with the environment and evolve by learning during real-time flight operations. This research aims to design a learning and evolving flight controller for a fixed-wing UAS using artificial neural networks that are thoroughly flight test validated showcasing repeatability, robustness and performance in presence of external disturbances.

1.2 Why Artificial Neural Networks

There are many independent and mathematical studies [39, 132, 66, 57, 146, 150] that showcase the excellent approximation and generalization capabilities of artificial neural networks. One of the most important work in this area of research is in Reference [39], which shows that artificial neural networks with only one hidden layer consisting of a sigmoidal activation function(s) can

universally approximate any arbitrary but continuous function to an arbitrary degree of accuracy, given that the number of nodes and size of weights are not constrained. The applications of neural networks in the field of control systems has proven to be effective for feedback control of nonlinear systems [53, 111, 15, 86]. It is shown in Reference [132] that neural networks with as few as two hidden layers are sufficient for feedback stabilization of a nonlinear control system.

A very important study on neural networks was conducted in 1987 by MIT Lincoln Laboratory, which was directed by Defense Advanced Research Projects Agency (DARPA) [142]. It was concluded that neural networks will provide the necessary ingredients for the next generation of intelligent machines. They offer knowledge formation and organization. The major conclusions from the study were that neural networks can basically replace application-specific software due to their capability of self-adaptation and learning. And, that the mathematical advances in the study of neural networks that have been made in the recent years has helped the research to mature greatly. The study also concluded that significant demonstrations in the use of neural networks have been made for speech recognition, signal processing robotics, etc. Another major study conducted at the National Aeronautics and Space Administration (NASA) Ames Research Center on using artificial neural networks for flight control, concluded that neural network algorithms can solve certain problems that standard control methods were not able to address effectively [73]. The study indicated that the generalization capabilities may increase robustness characteristics for control systems designed using artificial neural networks.

1.3 Literature Review

There has been considerable research in utilizing neural networks and applying machine learning techniques for assisting in the development of autopilots and control systems. However, the work presented in this thesis is unique as it entails development of neural network flight controllers that directly learn from data, breaking feedback correlations and generalizing a policy-based control, with a potential to continuously evolve with subsequent flight tests. The work on using ANNs for an aircraft autopilot goes back to the year 1991 [140]. Most of the research conducted in

this area has been concentrated around developing or training neural networks to act as low-level control systems [60, 104, 122, 17, 36] that compute control surface outputs when provided with guidance commands. In Reference [26] neural networks are employed for predicting if the stability augmentation system's gains are too high. The major difference in the approach presented in this dissertation is to directly use the neural network as the controller which has all the nonlinear gains and system embeddings as its internal parameters that generalizes over a large range of system and environmental conditions. In Reference [140] an artificial neural network is used to output longitudinal controls using pilot commands as inputs. In Reference [20], neural networks are used as controllers but they estimate the gains in the varying trim conditions regime using gain scheduling techniques.

Another major application of ML techniques that use neural networks has been in developing dynamic models, also known as aircraft system identification [91, 110, 120, 124]. Overall, the specific areas in which ML based ANN approximations are applied can be broadly categorized into the following areas:

- Control systems: gain scheduling [20], adaptive [14], nonlinear [27], wing rock [131], decoupled controllers [140], optimal control for spacecraft landing [125, 30].
- Aircraft system identification or modeling of flight dynamics [109], modeling of nonlinear aerodynamics [134], inverse plant model identification for feedback linearization [74, 78].
- Navigation path planning, trajectory optimization [151, 65, 64].
- On-board system failure detection[103], sensor validation [28].

None of the above applications utilize the universal approximation and generalization capabilities of neural networks that can truly build a fully autonomous and intelligent neural network flight controller. Such neural network controllers have the potential to evolve and learn to generalize across a wide range of flight envelope and environmental conditions.

Recently, reinforcement learning techniques, where an agent makes decisions in a simulated environment, have emerged as a new direction for ML control. The revival of reinforcement learning for control and the new excitement for this work comes from the use of deep neural networks as the policy learning agent. In this dissertation the aim is to extend successful reinforcement learning algorithms from the application to rotary wing aircraft [54, 31, 10, 67, 79, 110, 40, 34, 43, 24, 6, 5, 124, 33, 84] to a fixed-wing aircraft. The fundamental reason behind the application of reinforcement learning techniques limited to this domain is that these types of aircraft (quadcopters) can often take a "stop-think-act" approach as part of control decisions resulting in very quick alignment of neural networks to a steady "hover" state. Quadcopters behave as point mass objects and do not have the same complexity of a fixed-wing aircraft that has to maintain a minimum velocity (above stall) to keep flying. This presents a stronger challenge for initial learning as well as more complicated flight dynamics in the presence of even minimal adverse environmental conditions.

In Reference [119], an imitation learning algorithm is implemented in conjunction with DAgger algorithm for collision avoidance of a quadcopter through difficult forest environments. This is done using a monocular camera and using input images and correct control surface outputs from an expert pilot that demonstrates avoiding obstacles. In standard imitation learning paradigms, researchers use expert policy decisions to train a controller. The control decisions of the expert are used to provide feedback to the learner such that the learner converges to the expert policy over the course of training. This type of approach has resulted in state-of-the-art controllers in a variety of different applications including in robotics and vehicle control e.g. [11, 107, 12]. In the research herein, this approach is adapted to fixed-wing UAS where policy decisions jointly capture guidance, navigation and control decisions. While previous research has considered using imitation learning [119] or reinforcement learning e.g. [75, 31, 67, 151] for parts of UAS flight control, the application of these techniques has been elusive in fixed-wing aircraft, especially towards validation and verification flight tests.

Reference [151] describes a geometric reinforcement learning algorithm that is applied to the problem of path planning for single and multiple UAS. The effectiveness is shown via simulations

and there is no indication about the adaptability and generalization of the learned policies. Reference [50] presents a reinforcement learning algorithm for motion planning applied to a specific quadrotor platform, with a suspended weight with the objective of producing minimal residual oscillations.

Microsoft Research is using machine learning for aerial photogrammetry processing that is applied to agricultural applications [143]. Their aircraft platform mainly consists of quadrotors [32]. Unlike Microsoft Research, the approach used here is to design neural network controllers for high-speed and high-inertia fixed-wing aircraft. Google's Project Wing, a part of X – The Moonshot Factory, is focused on developing delivery drones with hybrid rotary and fixed-wing aircraft platforms [1]. The research involves development of collision avoidance systems using object detection techniques for making decisions in real-time [2].

Reference [153] presents a technique that utilizes a model predictive controller to train a deep neural network policy that is applied to a quadrotor platform. In references [5, 33, 6] learning techniques are implemented on a helicopter using apprenticeship learning algorithms. Based on the literature surveyed, learning techniques have not been explored extensively to the application of fixed-wing UAS for developing a fully autonomous system involving guidance and control. There is one example Reference [148], where reinforcement learning techniques are employed to a fixed-wing UAS, for navigating through thermal updraft with an objective of autonomous soaring. However, this application is representative of only path planning and guidance algorithms.

More recent research utilizing reinforcement learning algorithms for hybrid unmanned aerial systems capable of vertical take-off and landing [149] shows the application of neural networks as controllers. However, the neural network controller applies only to the quadcopter thrust control and does not extend to fixed-wing control surfaces which is a crucial difference in this research. Reference [145] shows application of reinforcement learning techniques to control a flock of unmanned aerial vehicles. However, only roll angle maneuvers are controlled as an outer loop through reinforcement learning techniques and the low level control is still performed by standard PID (Proportional-Integral-Derivative) autopilot systems. Reference [23] presents an application rein-

forcement learning technique into designing an attitude controller for a fixed-wing aircraft using proximal policy optimization [126] algorithm; however, only simulated flight tests are conducted to showcase neural network generalization capabilities.

The application of machine leaning methods being limited to rotary-wing platforms is undoubtedly due to higher complexities innate to a fixed-wing aircraft owing to fast changing dynamics due to high speed and nonlinear, unsteady aerodynamics. In contrast, a rotary wing aircraft flies relatively slowly, with hovering capabilities, potentially allowing the aircraft to have a minimum velocity of '0'. This leads to slow changing dynamics and almost a linear behavior in the system, with a capacity to stop, think and act.

1.4 Unique Contributions

1. Empirical evidence demonstrates drawbacks in simple supervised learning algorithms, such as data aggregation set, that collect fixed labeled data sequentially, and are not applicable to complex flight missions for a nonlinear fixed-wing aircraft. Monte-Carlo DAgger algorithm that can aggregate variable data needed to train an artificial neural network along complex flight paths is developed to successfully mimic an "expert" guidance, navigation, and control policy.

2. A novel imitation learning algorithm called Moving Window DAgger that can generalize over complex flight paths without causing oscillations in aircraft states is developed for training neural networks. A time-based window that successfully captures aircraft transient behavior and hence mitigates oscillatory and over-fitted predictions is employed in the standard DAgger algorithm for data collection along the flight trajectory.

3. The imitation neural network models inherently depend on the "expert" policy that in turn depends on flight controllers, which are designed using physics-based aircraft models. Therefore, in order to design neural network controllers that are aircraft model agnostic, reinforcement learning techniques are used. Neural network controllers are directly trained from

data generated using engineering level dynamic model of the aircraft with low-fidelity in low Reynold's numbers. The neural network controller demonstrated unique capabilities to generalize a control policy in a series of flight tests and exhibits robustness to achieve the desired performance in presence of external disturbances (cross wind, gust, etc.).

4. Extensive Monte-Carlo software in the loop simulation flight tests are conducted to verify the developed neural network flight controller. Hardware in the loop tests are also conducted after implementing the neural network controller onboard the aircraft computer, and finally it is flight tested using a Skyhunter UAS that exhibits nonlinear characteristics and is a foam-based, twin-boom, pusher aircraft.

5. The longitudinal neural network controller trained using reinforcement learning and low-fidelity aircraft dynamic model is validated for performance in 15 distinct flight tests, spread over a period of 5 months (November 2019 to March 2020), consisting of 21 different flight conditions. Flight missions consisting of automatic deployments based on aircraft's inertial location are programmed on the onboard flight computer. The missions are activated through the ground control station to conduct a fair assessment and comparison of LQR and neural network flight controllers.

6. The Deep Deterministic Policy Gradients (DDPG), a reinforcement learning algorithm, is implemented to re-train and evolve the longitudinal neural network flight controller directly from realistic flight data. Portions of the DDPG algorithm are extracted and modified to implement the evolution learning paradigm using TensorFlow software [4].

Chapter 2

Theory and Background

2.1 Guidance, Navigation and Control

A complete autonomous software for an aerospace vehicle consists of individual guidance, navigation and control (GNC) algorithms. Each of these GN&C algorithms as laid out in blue boxes of Figure 2.1 entail an inherent level of complexity in terms of computation, processing of sensor-state-measurement information, and intra-signal exchange among algorithms [76]. Navigation algorithms (*where am I?*) generally consist of sensor - instrumentation data acquisition, pre-processing, signal conditioning, filtering, etc. Kalman filtering algorithms that perform real time state estimation are a part of the navigation block. The navigation system accepts inputs (raw data) from various sensors such as Global Positioning System (GPS), Inertial Measurement Unit (IMU), Differential Pressure Sensor connected to a Pitot Tube, etc.

After the state estimation algorithms and data pre-processing, these state outputs are fed into the guidance block (*where should I go?*). Guidance algorithms can be broadly divided into two major types: (1) path planning, and (2) state commands. In addition to the state estimation inputs, user

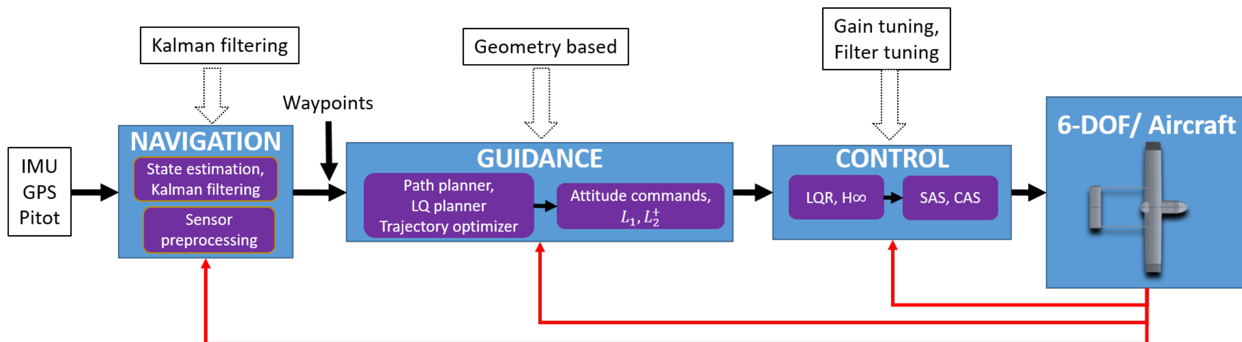


Figure 2.1: Navigation → Guidance → Control: General Architecture

defined waypoints are also fed into the guidance block. Using the state information and waypoints, the path planner finds the trajectory, generally in inertial coordinates that the aircraft needs to fly at. The path planner can involve optimal path planning algorithms, such as the LQR (Linear Quadratic Regulator) path planner [129], in conjunction with any automatic collision avoidance algorithms. The next major part of the guidance block is the state command generator. Generally, the state command generator can output the desired velocity (airspeed: $V_{T_{cmd}}$), altitude (h_{cmd}), pitch (θ_{cmd}) and roll (ϕ_{cmd}) angles [76]. These guidance algorithms are further divided into longitudinal and lateral logic. Longitudinal guidance generally provides the pitch angle command that the aircraft nose (body X-axis) should achieve relative to the local geographic horizontal plane, and velocity command. Lateral guidance outputs the roll angle command, or the approximate bank angle required to guide the aircraft to the desired lateral path.

Control algorithms (*how should I go there?*) take the commanded states as inputs and their goal to reduce the errors between these commands and the actual measured or estimated states. These algorithms include optimal controllers such as LQR, robust controllers such as H-∞ [56], etc. This block can also include systems, such as Stability Augmentation System (SAS) and Control Augmentation System (CAS) to provide further stability to the aircraft. The longitudinal and lateral guidance outputs serve as inputs to their respective counterpart control algorithms. Longitudinal control returns the throttle (δ_t) and elevator (δ_e) control variables, and Lateral control returns the aileron (δ_a) and rudder (δ_r) control variables, see Figure 2.1.

To provide a complete picture of the software architecture used for GNC simulation based design, tuning, and tests; the last block on the far right of Figure 2.1 depicts the full six-degrees-of-freedom (6-DOF), equations of motion (EOM) of the aircraft. In a real flight test, this is simply replaced by the aircraft itself or the sensors that measure its states.

2.2 Aircraft Equations of Motion

For this research a “Skyhunter” UAS platform is used for various simulations and flight test experiments, see Section 3.1. A 6-DOF rigid body model for Skyhunter is developed using the approach

in Reference [136]. Skyhunter is a twin-boom aircraft, equipped with a pusher electric motor, and three aerodynamic control surfaces: one elevator, one set of differential ailerons and two rudders. The nonlinear equations of motion are represented in 2.1. Here, $\{u,v,w\}$ and $\{p,q,r\}$ are the translational and angular velocity components respectively defined in the aircraft body coordinate frame. The 3-dimensional (3D) position coordinates are defined in an inertial right-handed frame of reference called as NED (North-East-Down), and the corresponding inertial coordinates are denoted as: $\{N,E,D\}$. The Euler or the attitude angles are denoted by: $\{\phi,\theta,\psi\}$. The body-axis aerodynamic forces and moments are denoted as: $\{X_A,Y_A,Z_A\}$ and $\{L_A,M_A,N_A\}$, respectively, and $\{X_T\}$ is the thrust force in the body X-axis direction.

$$\dot{u} = \frac{(X_T + X_A)}{m} + g_x + rv - qw \tag{2.1a}$$

$$\dot{v} = \frac{Y_A}{m} + g_y - ru + pw \tag{2.1b}$$

$$\dot{w} = \frac{Z_A}{m} + g_z + qu - pv \tag{2.1c}$$

$$\dot{p} = (I_{zz}L + I_{xz}N - (I_{xz}(I_{yy} - I_{xx} - I_{zz})p + ((I_{xz}^2) + I_{zz}(I_{zz} - I_{yy}))r)q)/(I_{xx}I_{zz} - (I_{xz}^2)) \tag{2.1d}$$

$$\dot{q} = (M - (I_{xx} - I_{zz})pr - I_{xz}((p^2) - (r^2)))/I_{yy} \tag{2.1e}$$

$$\dot{r} = (I_{xz}L + I_{xx}N + (I_{xz}(I_{yy} - I_{xx} - I_{zz})r + ((I_{xz}^2) + I_{xx}(I_{xx} - I_{yy}))p)q)/(I_{xx}I_{zz} - (I_{xz}^2)) \tag{2.1f}$$

$$\dot{X}_I = (\cos\theta\cos\psi)u - (\cos\phi\sin\psi - \sin\phi\sin\theta\cos\psi)v + (\sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi)w \tag{2.1g}$$

$$\dot{Y}_I = (\cos\theta\sin\psi)u + (\cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi)v - (\sin\phi\cos\psi - \cos\phi\sin\theta\sin\psi)w \tag{2.1h}$$

$$\dot{Z}_I = -(\sin\theta)u + (\sin\phi\cos\theta)v + (\cos\phi\cos\theta)w \tag{2.1i}$$

$$\dot{\phi} = P + \tan\theta(Q\sin\phi + R\cos\phi) \tag{2.1j}$$

$$\dot{\theta} = Q\cos\phi - R\sin\phi \tag{2.1k}$$

$$\dot{\psi} = (Q\sin\phi + R\cos\phi)/\cos\theta \tag{2.1l}$$

The terms $\{I_{xx},I_{yy},I_{zz}\}$ are the moments of inertia of the aircraft and $\{I_{xy},I_{yz},I_{xz}\}$ are the products of inertia [135]. As an instance, $I_{xx} = \int_{body}(y^2 + z^2)dm$, where y and z are the position vector

components of a small point mass on the aircraft (dm). They are generally represented in the inertia matrix (J_I), which is symmetric and is defined as in Equation 2.2.

$$\begin{bmatrix} I_{xx} & -I_{xy} & I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \tag{2.2}$$

The dynamics of the pusher electric motor, elevator, aileron and rudder servos are modeled as first order differential equations with time constants $\{\tau_t, \tau_e, \tau_a, \tau_r\}$ respectively and the servo states are denoted as: $\{\delta_t, \delta_e, \delta_a, \delta_r\}$, respectively. Mass of the aircraft is denoted as m.

In the above Equations 2.1, $\{g_x, g_y, g_z\}$ are the gravitational acceleration components in the aircraft body-axis. These are represented by Equations 2.3.

$$g_x = g \sin\theta \tag{2.3a}$$

$$g_y = g \sin\phi \cos\theta \tag{2.3b}$$

$$g_z = g \cos\phi \cos\theta \tag{2.3c}$$

In Equations 2.1, the first three sub-equations are replaced with the following states: total velocity (V_T), angle of attack (α) and the side slip angle (β) for implementation in the simulator. The relationship between translational aircraft velocities and these states are represented in Equations 2.4 and 2.5.

$$u = V_T \cos\alpha \cos\beta \tag{2.4a}$$

$$v = V_T \sin\beta \tag{2.4b}$$

$$w = V_T \sin\alpha \cos\beta \tag{2.4c}$$

$$V_T = \sqrt{u^2 + v^2 + w^2} \tag{2.5a}$$

$$\alpha = \sin^{-1}\left(\frac{v}{V_T}\right) \tag{2.5b}$$

$$\beta = \tan^{-1}\left(\frac{w}{u}\right) \tag{2.5c}$$

The differential equations for $\{V_T, \alpha, \beta\}$ are shown in Equation 2.6.

$$\dot{V_T} = \frac{u\dot{u} + v\dot{v} + w\dot{w}}{V_T} \tag{2.6a}$$

$$\dot{\alpha} = \frac{u\dot{w} - w\dot{u}}{u^2 + w^2} \tag{2.6b}$$

$$\dot{\beta} = \frac{\dot{v}(u^2 + w^2) - v(u\dot{u} + w\dot{w})}{{V_T}^2\sqrt{u^2 + w^2}} \tag{2.6c}$$

The servo dynamics, Equations 2.7 are modeled as first order ordinary differential equations. The variables $\delta_{t_x}, \delta_{e_x}, \delta_{a_x}, \delta_{r_x}$ are the previous step integrated values or the actual servo deflections for throttle, elevator, aileron, and rudder, respectively, which the aircraft actually experiences. The variables $\delta_t, \delta_e, \delta_a, \delta_r$ are the commanded values of these control surfaces.

$$\dot{\delta_t} = -\frac{1}{\tau_t}\delta_{t_x} + \frac{1}{\tau_t}\delta_t \tag{2.7a}$$

$$\dot{\delta_e} = -\frac{1}{\tau_e}\delta_{e_x} + \frac{1}{\tau_e}\delta_e \tag{2.7b}$$

$$\dot{\delta_a} = -\frac{1}{\tau_a}\delta_{a_x} + \frac{1}{\tau_a}\delta_a \tag{2.7c}$$

$$\dot{\delta_r} = -\frac{1}{\tau_r}\delta_{r_x} + \frac{1}{\tau_r}\delta_r \tag{2.7d}$$

The states from equations of motion, Equations 2.1 are augmented with servo dynamics (Equations 2.7) to represent the complete EOM of the aircraft that take into account the actuator dynamics. Therefore, the complete state vector consists of 16 variables:

$$\mathbf{X} = \left[V_T, \alpha, \beta, \phi, \theta, \psi, p, q, r, N, E, D, \delta_{t_x}, \delta_{e_x}, \delta_{a_x}, \delta_{r_x}\right]^T \tag{2.8}$$

The control vector consists of four variables:

$$\mathbf{U} = \left[\delta_t, \delta_e, \delta_a, \delta_r\right]^T \tag{2.9}$$

Therefore, the complete set of coupled nonlinear differential EOMs are represented in 2.10.

$$\dot{\mathbf{X}} = \mathbf{f}(\mathbf{X}, \mathbf{U}) \tag{2.10}$$

2.3 Base Autopilot

The base autopilot software consists of standard decoupled guidance and control algorithms, with a top-level path follower (waypoint switching logic). The base autopilot is designed around and tested to perform race-track trajectory tracking, altitude and velocity holds, and navigating via four waypoints defined by a user.

The path follower module of the software enables the guidance to follow straight waypoint lines comprised of two waypoints at an instant (wpt_1 and wpt_2). The path follower switches to the next two waypoints from a set of waypoints defined by a user, when the aircraft comes in close proximity (switching distance) to a waypoint (wpt_2). This switching distance is a function of heading change, minimum turn radius of the aircraft and the GPS velocity (V_{gps}).

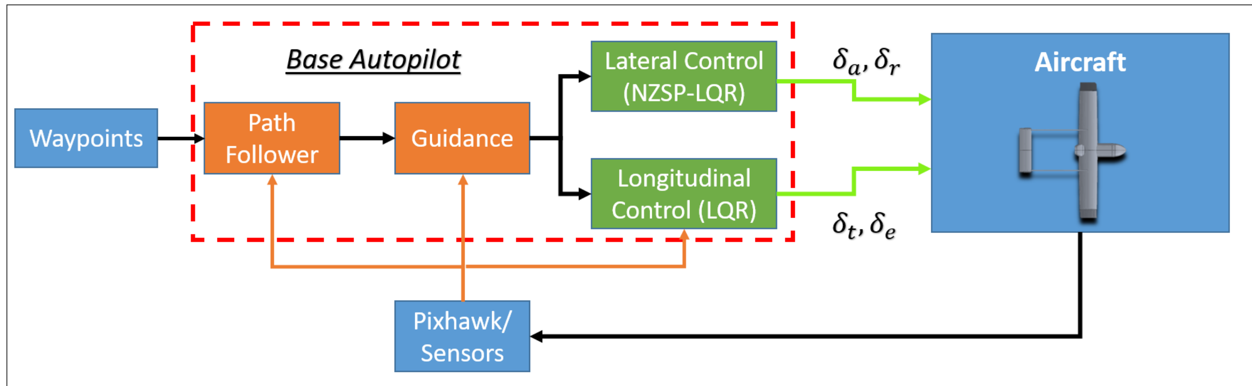


Figure 2.2: Base Autopilot

Decoupled lateral and longitudinal guidance produce roll and pitch commands respectively. The Lateral controller is designed using a NZSP-LQR algorithm [59]. The longitudinal controller is a command tracking LQR algorithm that minimizes the integral error for airspeed and pitch angle commands. For designing the decoupled controllers, separate linear time invariant (LTI) models for lateral and longitudinal dynamics are used. A LTI model equation is shown in 2.11.

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} \tag{2.11}$$

In Equation 2.11, $\mathbf{x}$ is the state vector and $\mathbf{u}$ is the control vector. The system matrices are

defined as A and B. The controller is designed assuming full state feedback.

2.3.1 Guidance

Lateral Guidance: The guidance algorithm used for roll command is inspired by L_2^+ algorithm as described in Reference [38]. For completeness, the main equations are detailed here. The algorithm mainly computes a lateral centripetal acceleration, see Equation 2.12, that is required to follow a circular arc that connects from the aircraft's current location to a reference point (RP) on the desired path, see Figure 2.3.

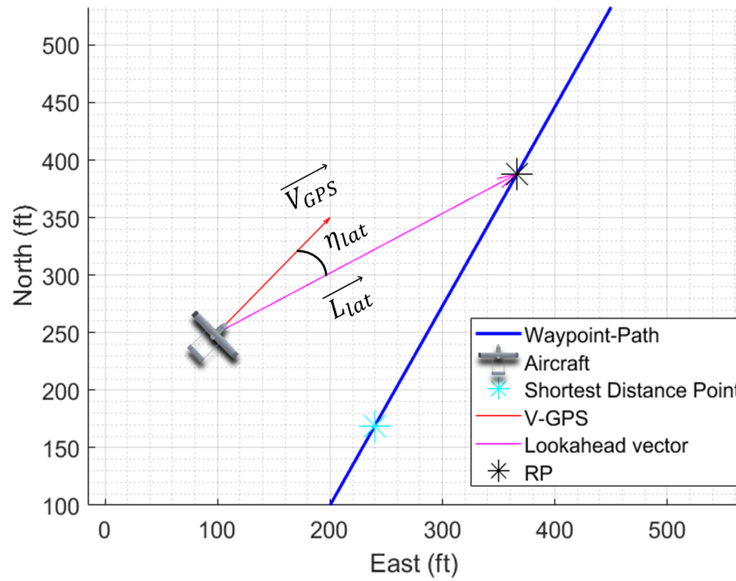


Figure 2.3: Roll Angle Attitude Guidance Logic

$$a_{lat} = 2\frac{V_{XY}}{T^*}sin(\eta_{lat}) \tag{2.12}$$

As shown in Figure 2.3, the lateral acceleration is generated to minimize the η_{lat} angle so as to follow a line of sight path towards the desired trajectory. η_{lat} is the angle between the aircraft's ground speed V_{XY} (L_2 norm of XY components of $\overrightarrow{V_{gps}}$) vector and an imaginary $\overrightarrow{L_{lat}}$ vector. The $\overrightarrow{L_{lat}}$ vector is defined by its magnitude called the lookahead distance and is a guidance tuning parameter. Based on the lookahead distance and the aircraft location, a line is projected from the

aircraft to the desired waypoint path. The intersection point of this $\overrightarrow{L_{lat}}$ vector with the desired waypoint path, is called the reference point (RP). This guidance law was designed by applying a suitable modification to its original counterpart presented in [108]. The L_2^+ logic compensates for roll response lag, and mitigates the dependency of pole location of roll dynamics on V_{XY}. Here, T^* is a constant, the value of which should be selected around three to four times the roll response lag of the aircraft, as shown in Reference [38]. Using the lateral acceleration (a_{lat}) from Equation 2.12, the roll command can be computed as shown in Equation 2.13. Here, g is the gravitational acceleration constant.

$$\phi_{cmd} = \tan^{-1}\left(\frac{a_{lat}}{g}\right) \tag{2.13}$$

The complete procedure for generating the roll guidance commands is shown in Algorithm 1. Note that all the vector inputs to this algorithm have finite values for their X and Y components but their third component is set to zero. The third component is needed to make the cross products work.

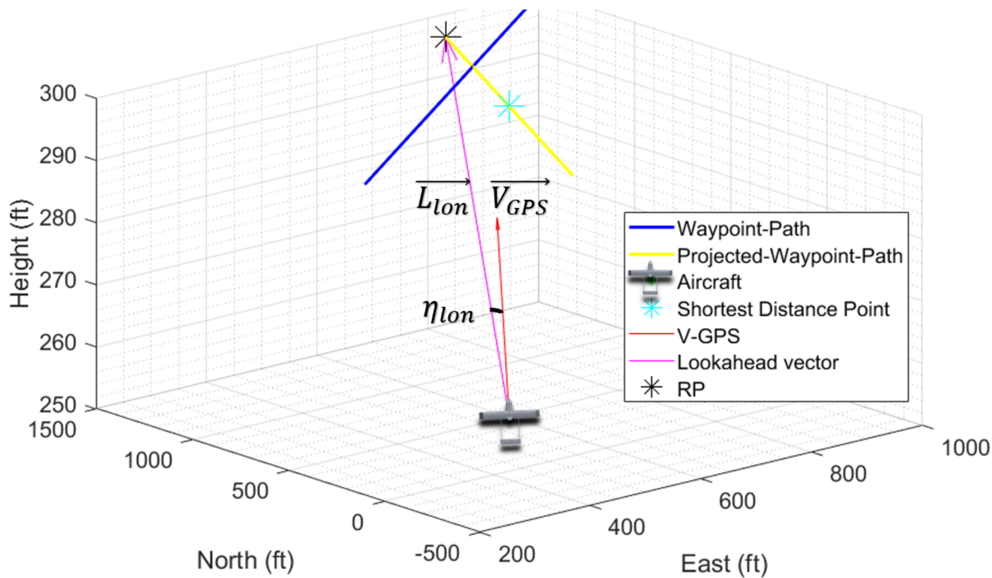


Figure 2.4: Pitch Angle Attitude Guidance Logic

Longitudinal Guidance: This guidance algorithm generates a pitch attitude command and is

Algorithm 1 Roll Command Guidance

Inputs: $L_1, T^*, \overrightarrow{w_1}, \overrightarrow{w_2}, \vec{P}, \overrightarrow{V_{XY}}, \phi_{limit}$

Output: ϕ_{cmd}

1: Vector from $\overrightarrow{w_1}$ to UAV position $\vec{P}$

$$\overrightarrow{w_{1P}} = \vec{P} - \overrightarrow{w_1}$$

2: Vector from $\overrightarrow{w_1}$ to $\overrightarrow{w_2}$

$$\overrightarrow{w_{12}} = \overrightarrow{w_2} - \overrightarrow{w_1}$$

3: Perpendicular distance or the shortest distance from the UAV to the waypoint line

$$p_d = \frac{||\overrightarrow{w_{1P}} \times \overrightarrow{w_{12}}||_2}{||\overrightarrow{w_{12}}||_2}$$

4: Distance from $\overrightarrow{w_1}$ to the intersection point (shortest distance point) of the perpendicular line from UAV

$$w_{1d} = \frac{\overrightarrow{w_{1P}} \cdot \overrightarrow{w_{12}}}{||\overrightarrow{w_{12}}||_2}$$

5: Angle between waypoint line and X-North axis: desired heading

$$\psi_d = \tan^{-1} \frac{\overrightarrow{w_{12}}(y)}{\overrightarrow{w_{12}}(x)}$$

6: Distance between shortest distance point and RP (RP = intersection point of $\overrightarrow{L_{lat}}$ vector with the waypoint line)

$$d_{RP} = \sqrt{L_1^2 - p_d^2}$$

7: Coordinates of the intersection point RP

$$\overrightarrow{RP} = \begin{bmatrix} \overrightarrow{w_1}(x) + (w_{1d} + d_{RP})\cos(\psi_d) \\ \overrightarrow{w_1}(y) + (w_{1d} + d_{RP})\sin(\psi_d) \\ \vec{P}(z) \end{bmatrix}$$

8: The lookahead vector

$$\overrightarrow{L_{lat}} = \overrightarrow{RP} - \vec{P}$$

9: Cross track error angle: η_{lat}

$$\eta_{lat} = \sin^{-1} \frac{||\overrightarrow{V_{XY}} \times \overrightarrow{L_1}||_2}{||\overrightarrow{V_{XY}}||_2 ||\overrightarrow{L_1}||_2}$$

10: Lateral Acceleration

$$a_{lat} = 2\frac{||V_{XY}||_2}{T^*} sin(\eta_{lat})$$

11: Roll angle command

$$\phi_{cmd} = sgn((\overrightarrow{V_{XY}} \times \overrightarrow{L_1})(z)) \tan^{-1}\left(\frac{a_{lat}}{g}\right)$$

12: **if** $|\phi_{cmd}| > \phi_{limit}$ **then**

13: $\quad |\phi_{cmd}| = sgn(\phi_{cmd})\phi_{limit}$

14: **end if**

designed based on a lookahead distance in a decoupled vertical plane that places a waypoint line in the direction of ground speed (V_{XY}) at the desired altitude. The geometry of this guidance is represented in Figure 2.4. The desired path is represented by a blue line. The algorithm projects a straight line (shown in yellow) in the direction of ground speed V_{XY} but at an altitude of the desired path. Using this projected waypoint path, the η_{lon} angle is computed between the $\overrightarrow{L_{lon}}$ and $\overrightarrow{V_{gps}}(\equiv \overrightarrow{V_{XYZ}})$ vectors as shown in Figure 2.4. The goal is to minimize the η_{lon} angle by generating an equivalent pitch command. This η_{lon} angle is passed through a sigmoid type function to smooth out any abrupt variations and is multiplied by a scaling factor to generate the pitch command (θ_{cmd}). The complete algorithm for pitch attitude guidance is shown in Algorithm 2.

2.3.2 Control

Lateral Controller: The lateral controller is designed to regulate four states and also command two states to a non-zero value. The four lateral states for this controller are: side slip angle (β), roll angle (ϕ), roll rate (P) and yaw rate (R). All of these four states are regulated to zero values, but the side slip angle and the roll angle are also commanded to some non-zero values generated by the lateral guidance algorithm. The lateral state vector ($\mathbf{x}$) and the control vector ($\mathbf{u}$) are defined in Equations 2.14. Note that the controllers are designed using the linear time invariant model (LTI) of the aircraft which is obtained by perturbation around a trim point. Hence, the states and controls involved in this design are also perturbed and not total values.

$$\mathbf{x} = \begin{bmatrix} \beta & \phi & P & R \end{bmatrix}^T \tag{2.14a}$$

$$\mathbf{u} = \begin{bmatrix} \delta_a & \delta_r \end{bmatrix}^T \tag{2.14b}$$

A non-zero set-point (NZSP) lateral controller [59] is used to control the roll and side-slip angles of the aircraft. In order to control these variables to non-zero values, new state and control trim points are defined as $\mathbf{x}^*$ and $\mathbf{u}^*$, respectively. The new trim point $\mathbf{x}^*$ is assumed to have the same LTI dynamics as $\mathbf{x}$, see Equation 2.15a. Therefore, the error in states and controls for the

Algorithm 2 Pitch Command Guidance

Inputs: $L_{lon}, H_d, \vec{P}, \overrightarrow{V_{XYZ}}, \theta_{limit}$

Output: θ_{cmd}

1: Perpendicular distance or the shortest distance from the UAV to the desired altitude

$$p_d = |-H_d - \vec{P}(z)|$$

2: Coordinates of the intersection point (shortest distance point) of the perpendicular line from UAV to the imaginary waypoint line in the vertical plane

$$\overrightarrow{w_{1d}} = \begin{bmatrix}\vec{P}(x) & \vec{P}(y) & -H_d\end{bmatrix}$$

3: Ground course or aircraft heading from GPS ground speed - V_{XY}

$$\psi_{GPS} = \tan^{-1} \frac{\overrightarrow{V_{XYZ}}(y)}{\overrightarrow{V_{XYZ}}(x)}$$

4: Coordinates of a point L_{lon} distance away from $\overrightarrow{w_{1d}}$ in the opposite direction of $\overrightarrow{V_{XY}}$

$$\overrightarrow{w^-} = \begin{bmatrix}\overrightarrow{w_{1d}}(x) - L_{lon}\cos(\psi_{GPS}) & \overrightarrow{w_{1d}}(y) - L_{lon}\sin(\psi_{GPS}) & \overrightarrow{w_{1d}}(z)\end{bmatrix}$$

5: Distance between shortest distance point and RP (RP = intersection point of $\overrightarrow{L_{lon}}$ vector with the waypoint line)

$$d_{RP} = \sqrt{L_{lon}^2 - p_d^2}$$

6: Coordinates of the intersection point RP

$$\overrightarrow{RP} = \begin{bmatrix}\overrightarrow{w^-}(x) + (L_{lon} + d_{RP})\cos(\psi_{GPS}) \\ \overrightarrow{w^-}(y) + (L_{lon} + d_{RP})\sin(\psi_{GPS}) \\ \overrightarrow{w^-}(z)\end{bmatrix}$$

7: The lookahead vector

$$\overrightarrow{L_{lon}} = \overrightarrow{RP} - \vec{P}$$

8: Angle between $\overrightarrow{L_{lon}}$ vector and the horizontal plane

$$\Gamma = \tan^{-1} \frac{\overrightarrow{L_{lon}}(z)}{\sqrt{\overrightarrow{L_{lon}}(x)^2 + \overrightarrow{L_{lon}}(y)^2}}$$

9: Flight Path Angle

$$\gamma = \tan^{-1} \frac{\overrightarrow{V_{XYZ}}(z)}{\sqrt{\overrightarrow{V_{XYZ}}(x)^2 + \overrightarrow{V_{XYZ}}(y)^2}}$$

10: Vertical track error angle: η_{lon}

$$\eta_{lon} = \sin^{-1} \frac{||\overrightarrow{V_{XYZ}} \times \overrightarrow{L_{lon}}||_2}{||\overrightarrow{V_{XYY}}||_2 ||\overrightarrow{L_{lon}}||_2}$$

11: Pitch angle command

$$\theta_{cmd} = sgn(\gamma - \Gamma)\theta_{limit} \frac{e^{\eta_{lon}} - e^{-\eta_{lon}}}{e^{\eta_{lon}} + e^{-\eta_{lon}}}$$

original trim points $\mathbf{x}$ and $\mathbf{u}$ with respect to the new trim points are as shown in Equations 2.15b. From Equations 2.15a and 2.15b, the error dynamics can be derived as shown in Equation 2.15c.

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} \qquad \text{and} \qquad \dot{\mathbf{x}^*} = A\mathbf{x}^* + B\mathbf{u}^* \tag{2.15a}$$

$$\tilde{\mathbf{x}} = \mathbf{x} - \mathbf{x}^* \qquad \text{and} \qquad \tilde{\mathbf{u}} = \mathbf{u} - \mathbf{u}^* \tag{2.15b}$$

$$\dot{\tilde{\mathbf{x}}} = A\tilde{\mathbf{x}} + B\tilde{\mathbf{u}} \tag{2.15c}$$

The goal of the LQR control theory is to minimize a cost function J, using Q and R weighting matrices, in order to solve for control, see Equation 2.16.

$$J = \int \left(\tilde{\mathbf{x}}^T Q\tilde{\mathbf{x}} + \tilde{\mathbf{u}}^T R\tilde{\mathbf{u}}\right) dt \tag{2.16a}$$

$$\tilde{\mathbf{u}} = -K_{lat}\tilde{\mathbf{x}} \tag{2.16b}$$

The dynamic system equations are shown in 2.17. From these equations, it can be seen that the new trim state ($\mathbf{x}^*$) has the same system matrices A and B. At steady state, $\dot{\mathbf{x}^*} \to 0$ and the system output reaches the commanded states, that is, $\mathbf{y} \to \mathbf{y_m}$.

$$\dot{\mathbf{x}^*} = A\mathbf{x}^* + B\mathbf{u}^* \tag{2.17a}$$

$$\mathbf{y} = H\mathbf{x}^* + D\mathbf{u}^* \tag{2.17b}$$

Therefore, at steady-state, the new system is shown in Equations 2.18. The matrix shown on the left hand side of this equation, consisting of elements: A, B, H, D is called the Quad-Partition-Matrix (QPM). This equation can be solved if the QPM is full rank, by taking its inverse, to obtain the values for $\mathbf{x}^*$ and $\mathbf{u}^*$, as shown in Equations 2.18.

$$\begin{bmatrix} A & B \\ H & D \end{bmatrix} \begin{bmatrix} \mathbf{x}^* \\ \mathbf{u}^* \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{y_m} \end{bmatrix} \tag{2.18a}$$

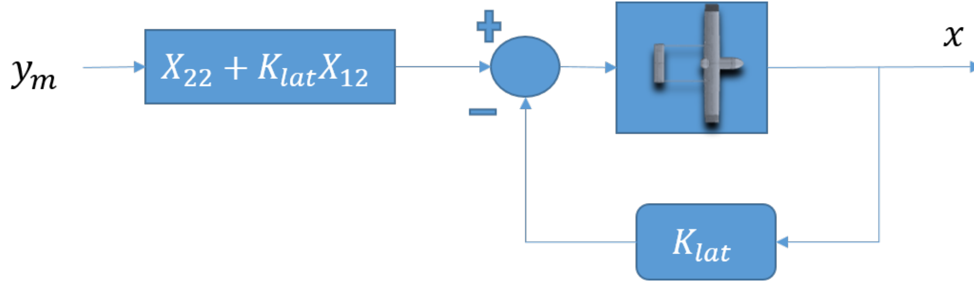


Figure 2.5: Non Zero Set Point Lateral Controller

$$\begin{bmatrix} \mathbf{x}^* \\ \mathbf{u}^* \end{bmatrix} = \begin{bmatrix} A & B \\ H & D \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ \mathbf{y_m} \end{bmatrix} \tag{2.18b}$$

$$\begin{bmatrix} \mathbf{x}^* \\ \mathbf{u}^* \end{bmatrix} = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} 0 \\ \mathbf{y_m} \end{bmatrix} \tag{2.18c}$$

From Equations 2.15b and 2.16b, it can be seen that $\mathbf{u} = \mathbf{u}^* + K_{lat}\mathbf{x}^* - K_{lat}\mathbf{x}$. Therefore, using the values for $\mathbf{x}^*$, $\mathbf{u}^*$ the final control is computed as shown in Equation 2.19.

$$\mathbf{u} = (X_{22} + K_{lat}X_{12})\mathbf{y_m} - K_{lat}\mathbf{x} \tag{2.19}$$

Here, X_{22} and X_{12} are elements of the inverse of QPM. K_{lat} is a 2×4 matrix, with first row representing gains that regulate the state vector ($\mathbf{x}$) using aileron controls (δ_a), in the respective order of elements of the state vector, and similarly the second row represents gains that regulate the state vector using rudder controls. $\mathbf{y_m}$ is a 2×1 vector consisting of state commands for the roll and the side-slip angles. X_{12} and X_{22} are 4×2 and 2×2 matrices respectively, which convert the state guidance commands to non-zero set point inputs to the regulator. The controller architecture is represented in Figure 2.5.

Longitudinal Controller: The longitudinal state and control vectors are shown in Equations 2.20. These four states are regulated to zero and two more integral states are added for command tracking of airspeed and pitch angle. Here, V_T is the total velocity which is assumed to be approximately equal to the forward airspeed (u), α is the angle of attack, θ is the pitch angle, and Q is

pitch rate.

$$\mathbf{x} = \begin{bmatrix} V_T & \alpha & \theta & Q \end{bmatrix}^T \tag{2.20a}$$

$$\mathbf{u} = \begin{bmatrix} \delta_t & \delta_e \end{bmatrix}^T \tag{2.20b}$$

The new state vector after augmenting the integral errors states is shown in Equation 2.21a. Assuming full state feedback, the new system equations are shown in 2.21b, see Reference [22].

$$\mathbf{x_{aug}} = \begin{bmatrix} V_T & \alpha & \theta & Q & \int V_{T_{cmd}} - V_T & \int \theta_{cmd} - \theta \end{bmatrix}^T \tag{2.21a}$$

$$\dot{\mathbf{x}}_{\mathbf{aug}} = A_{aug}\mathbf{x_{aug}} + B_{aug}\mathbf{u} \tag{2.21b}$$

Using LQR cost function, J and weighting matrices Q and R, the longitudinal control gain matrix (K_{lon}) can be computed as shown in Equation 2.22.

$$\mathbf{u} = -K_{lon}\mathbf{x} \tag{2.22}$$

The K_{lon} gain matrix is a 2×6 matrix, with the first four columns with both rows representing the gains for regulating the states, and the last two columns with both rows are the gains for the integral error states. The control input generated by this equation can end up accumulating integral errors and hence saturate to the maximum allowable control values. Therefore, if the control input saturates, then an anti-windup integral logic is used to reduce the control input. The control input $\mathbf{u}$ is assigned the maximum allowable value, if it increases the limit and it is denoted as $\mathbf{u_{sat}}$, otherwise it is simply $\mathbf{u_{sat}} = \mathbf{u}$. An anti-windup gain ($K_{aw}$) is multiplied with this error: $\mathbf{u_{sat}} - \mathbf{u}$ and added to the actual integral state errors ($\mathbf{x_{int}} = \begin{bmatrix} \int V_{T_{cmd}} - V_T & \int \theta_{cmd} - \theta \end{bmatrix}^T$). Then the controls for the integral error states is computed as shown in Equation 2.23, where K_{int} (2×2 matrix) is a part of the K_{lon} matrix.

$$\mathbf{u_{int}} = \int K_{int}(\mathbf{x_{int}} + K_{aw}(\mathbf{u_{sat}} - \mathbf{u}))dt \tag{2.23}$$

The final control input is computed using the difference between $\mathbf{u_{trim}}$ and $\mathbf{u_{int}}$ as shown in Equation 2.24, where $\mathbf{u_{trim}} = K_{trim}\mathbf{x}$ and K_{trim} (2×4 matrix) is a part of the K_{lon} gain matrix.

$$\mathbf{u} = \mathbf{u_{trim}} - \mathbf{u_{int}} \tag{2.24}$$

2.4 GNC to Artificial Neural Networks

Artificial neural networks or specifically feedforward neural networks are multi-layer perceptrons (MLPs) that in essence approximate complex relationships between inputs and outputs (mapping) and can operate as highly nonlinear functions or functions of functions (multiple layers) [58] that are parameterized by entities known as weights (w) and biases (b). ANNs can be thought of as nonlinear function approximation entities that generalize (statistically) over a large population of data, and are loosely inspired by the functioning of a human brain, containing hidden units (nodes or neurons) that perform computations in parallel, see Figure 2.6. Multi-layer ANNs are capable of performing nonlinear mapping and address the limitations inherent to single layered networks [89].

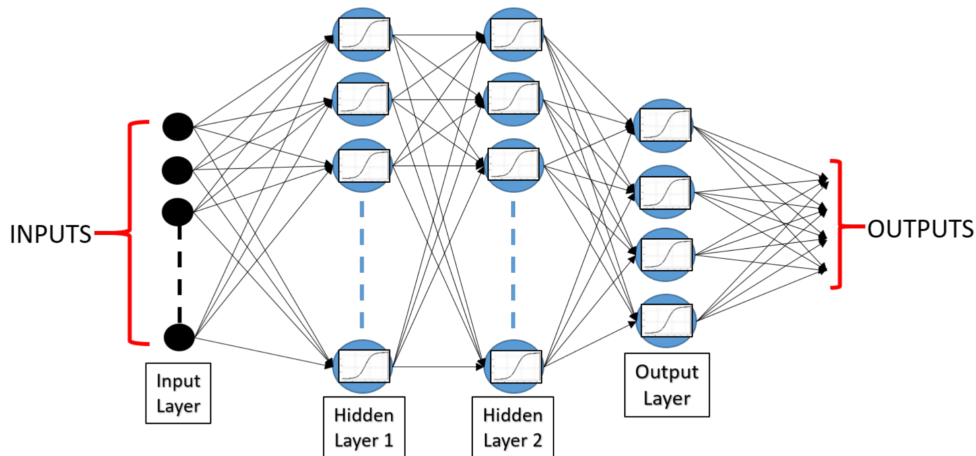


Figure 2.6: Artificial Neural Network: General Architecture

A single layer fully connected neural network parameterized by weights and biases, with a sigmoidal function can be represented as follows:

$$f(x|w,b) = \sigma(x^T w + b) \tag{2.25}$$

Here, x is the input to the neural network and σ is a sigmoidal type smoothing function, which can be a sigmoid, tanh (hyperbolic tangent), etc. A neural network's parameters are estimated using the so called "supervised learning" algorithms. In supervised learning the input to output data is given and it is labeled, meaning the inputs that correspond to specific outputs are known. Therefore, an objective function representing the mean squared error (MSE) between the actual outputs and the estimated outputs by the neural network, is minimized to estimate the weights and biases 2.26.

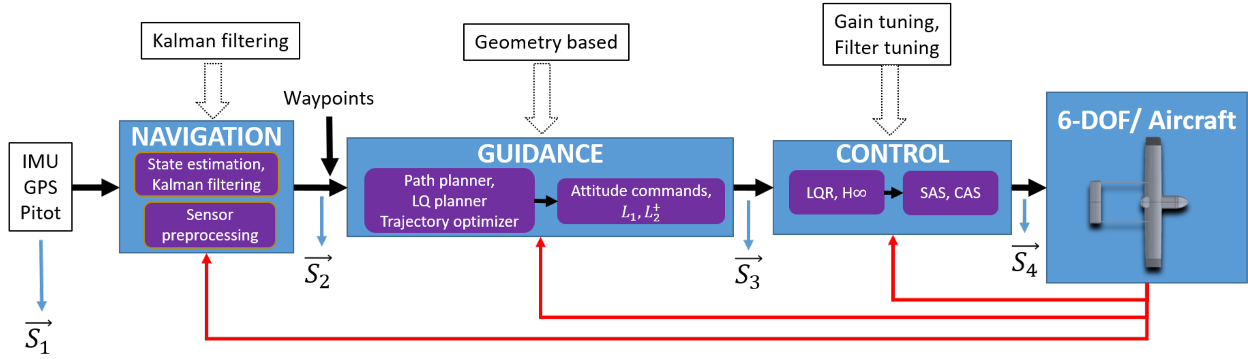
$$J(w,b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i|w,b))^2 \tag{2.26}$$

The general ANN architecture shown in Figure 2.6 is composed of three layers: two hidden and one output. The input stage is loosely called an "input layer" in most literature, however it is just a representational node. A generic L layer neural network is defined as consisting of an input stage, $(L-1)$ hidden layers, and one output layer [69]. Some layers are called hidden, because the training data does not explicitly represent the internal outputs of the "hidden layers". Mathematically, a multi-layer or a deep layer neural network, with two hidden and one output layers, can be represented as follows [58]:

$$f(\mathbf{x}) = f_3(f_2(f_1(\mathbf{x}))) \tag{2.27}$$

Comparing the properties of neural networks with a standard GNC system, it is empirically tested if the whole complex process of navigation $\rightarrow$ guidance $\rightarrow$ control can be learned with high enough fidelity for a stable flight by a neural network. A theoretical GNC block diagram to a neural network transformation is depicted in Figure 2.7.

Consider that the navigation block has an input vector consisting of state $\overrightarrow{S_1}$, thereby outputting a vector of state $\overrightarrow{S_2}$. The state $\overrightarrow{S_2}$ can be realized to include the feedback term from the sensor



$$\overrightarrow{S_2} = f_1(\overrightarrow{S_1}); \overrightarrow{S_3} = f_2(\overrightarrow{S_2}); \overrightarrow{S_4} = f_3(\overrightarrow{S_3})$$

$$\overrightarrow{S_4} = f_3(f_2(f_1(\overrightarrow{S_1})))$$

Figure 2.7: GNC to Deep Neural Network

measured states before becoming an input to a function f_2 and so forth. The main control output, represented here by $\overrightarrow{S_4}$, can be considered an output of a functional (function of functions) shown as $\overrightarrow{S_4} = f_3(f_2(f_1(\overrightarrow{S_1})))$. This is a general notion to show how a holistic GNC system, or its individual algorithms can be represented through a neural network that can learn its internal representations through data alone, without explicitly programming the logic and tuning the GNC parameters manually. Note, that the number of neural network layers required to perform this representation can vary significantly depending on the complexity of the problem at hand, the input states chosen, etc.

2.5 Artificial Neural Networks: Training

Artificial neural networks are generally trained using supervised learning techniques. Training an ANN means estimating its internal parameters: weights and biases, see Equation 2.25. The term supervised learning originates from the idea of a teacher teaching a student by simply associating correct answers to the questions in concern. For an ANN the questions correspond to inputs and the correct answers correspond to targets. In other words, the data given to a neural network is labeled, in a sense that the correct targets to the respective inputs are known in a supervised

learning paradigm. These supervised learning techniques utilize back-propagation [121] procedure and gradient descent algorithms to estimate the internal parameters of an ANN. Back-propagation algorithm provides a method of computing the gradients of the objective function with respect to the internal parameters of the network. Gradient descent is used to iteratively optimize the ANN parameters by driving the objective function to a low value.

2.5.1 Back-Propagation and Gradient Descent

Back-propagation, or simply referred to as backprop algorithm [58] is a method to compute gradients of a objective or cost function with respect to internal parameters of an ANN by back-propagating the errors or cost through multiple layers of the network. There is often a misconception that backprop is the complete algorithm that estimates these internal parameters. Backprop only refers to the method that provides gradients and propagates errors to the deep layers backwards (output $\rightarrow$ input). In contrast, the information flows in the forward direction from input to output in a typical neural network when it is used for prediction.

Backprop algorithm takes advantage of the chain rule of calculus to compute gradients for a neural network, and using these gradients the weights and biases are updated in the descending direction, that is applying gradient descent. Consider a single layer feedforward network with a sigmoid function as the activation:

$$\hat{y} = \sigma(wx + b) \tag{2.28}$$

Here, x is the input, w is the weight, b is the bias, $\hat{y}$ is the predicted output and σ is the sigmoid activation function, see Equation 2.29.

$$\sigma(z) = \frac{e^z}{e^z + 1} \tag{2.29}$$

Here, $z = wx + b$. Let the loss function be defined as follows:

$$L(y,\hat{y}) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.30}$$

Then, the derivative of a weight in the output layer can be derived as follows:

$$\frac{\delta L}{\delta w} = \frac{\delta L}{\delta \hat{y}}\frac{\delta \hat{y}}{\delta z}\frac{\delta z}{\delta w} \tag{2.31a}$$

$$\frac{\delta L}{\delta w} = 2(y-\hat{y})\sigma(z)(1-\sigma(z))x \tag{2.31b}$$

Here, $\sigma(z)(1-\sigma(z))$ is the derivative of the sigmoid function. The derivative computation for a bias parameter follows the similar method and can be derived in a straightforward way. This chain rule methodology can be extended to multiple layers by computing the gradients of the loss function to a weight which is situated further behind the output layer. After finding the derivative of a parameter (in this case w), the weight can be updated using gradient descent as follows:

$$w_{n+1} = w_n - \alpha_{lr}\left(\frac{\delta L}{\delta w}\right) \tag{2.32}$$

The learning parameter α_{lr} must be chosen to be a small number ($0 < \alpha_{lr} << 1$) to update the parameters slowly and prevent overshooting the minima of the loss function. Intuitively, the gradients provide useful information about the loss function regarding the direction in which the internal parameters should be changed in order to minimize the cost. There are certain practical and numerical issues that arise while following the straightforward gradient descent rule. When the gradients at a certain point are zero, then there is essentially no information about the direction to take for minimization. These points are known as critical points and can consist of undesired places such as *local minima* and *saddle points*. In a highly nonlinear neural network with deep layers and multi-dimensional inputs and outputs, numerous local minima and saddle points can exist. Other numerical minimization problems arise due to non-optimal and fixed learning rates. To mitigate these issues, second order gradient methods can be utilized.

The second derivative or gradient of the gradient, which is known as *Hessian* (multi-dimensional

inputs and outputs) gives information about how the gradient would change if the internal parameters are varied and can be used to detect local minima, local maxima and saddle points. In order to find the optimal learning rates, second order Taylor series approximation methods can be employed. Also a much faster class of second order approximation techniques known as Newton's methods can be used to directly jump to minimum using Hessian information. Let the Hessian be denoted by $\mathbf{H}$, and the objective or the loss function be $J(\theta)$, where θ is a vector of internal parameters (weights and biases). A typical Newton iteration to update the θ parameter is shown in Equation 2.33.

$$\theta_i = \theta_{i-1} - \mathbf{H}^{-1} \nabla_{\theta} J(\theta_{i-1}) \tag{2.33}$$

The Equation 2.33 is obtained by second order Taylor series approximation of the objective function $J(\theta_i)$, evaluated at iteration i. However, these methods only work well when the loss functions are of quadratic nature or when the Hessian is positive definite. In the latter case, Newton's method can be applied iteratively to approximate the internal parameters while finding the minimum. However, the use of Newton's method for estimating large number of internal parameters for neural networks presents a great computational burden due to inverse Hessian matrix calculations. This leads to the utilization of a class of methods known as *Conjugate Gradient Methods* [62, 52] that avoid the calculation of Hessian inverse.

2.5.2 Scaled Conjugate Gradient Descent Algorithm

Gradient descent methods or methods of steepest descent are applied iteratively to move the loss function in the decreasing gradient direction. Each iterative step which tends to decrease the loss, moves in a direction which is essentially orthogonal to the previous descent direction. This makes the method of steepest descent to move towards the minimum in a zig-zag fashion, making the overall optimization process relatively slower. In a sense, the iterative directions undo the progress made at the previous step in terms of finding the minimum. Therefore, conjugate directions [128]

are used which tend to add some weighted form of previous direction to the current descent direction. To compute conjugate directions, Eigen vectors of the Hessian matrix can be found in order to calculate the magnitude of previous direction that is to be added to current direction. However, Hessian calculations are avoided and the magnitude that incorporates previous search direction, can be computed by using several well known methods such as Fletcher-Reeves [51] or Polak-Ribière [115]. Let the current iteration direction be denoted by d_i, and let a coefficient β_i represent the magnitude of the previous iteration direction. Then, the current direction can be approximated as shown in Equation 2.34.

$$d_i = -\nabla_\theta J(\theta_{i-1}) + \beta_i d_{i-1} \tag{2.34}$$

Conjugate gradient methods are well known to handle large scale problems and are typically faster than standard back-propagation gradient descent methods [72, 101]. However, these methods still involve a computationally demanding step of line search, in which the best or optimal learning rate is found via minimizing the objective function evaluated with the updated parameters with a given conjugate direction. Let the optimal learning rate be denoted by α^*_{lr}, then this rate is estimated by using the following equations:

$$\alpha^*_{lr} = \min_{\alpha_{lr}} \frac{1}{m} \sum_{j=1}^{m} J(\theta_i) \tag{2.35a}$$

$$\theta_i = \theta_{i-1} + \alpha_{lr} d_i \tag{2.35b}$$

The *Scaled Conjugate Gradient* method (SCG) [102] instead uses a Levenberg-Marquardt [85] approach to compute the learning rate and hence avoids the line search step. First, the Hessian is approximated by using the first order derivative of the objective function and then a scalar factor is used to regularize the indefiniteness of the Hessian approximation. The complete details of the algorithm can be found in Reference [102].

2.5.3 The Adam Algorithm

Another class of first order algorithms that use adaptive learning rates holds significance for learning neural network parameters. The learning rate is the most important hyper-parameter that significantly affects a model's performance [58]. The notion of adapting learning rates can be tracked back to Reference [68], in which a parameter specific learning rate is increased if the derivative with respect to that parameter does not change sign, and vice-versa. In other methods such as *AdaGrad* [44], the learning rates are inversely scaled using square root of sum of all squared historical gradients. A slight modification to this algorithm known as *RMSProp* [63], takes into account exponentially weighted moving average of the square of gradients, and applies them to scale the learning rates in a similar way as *AdaGrad*. Other methods use a momentum term in the gradient descent step that reduces oscillations in the optimization process. The basic idea of using momentum is to incorporate an exponentially decaying moving average of the previous gradients in the current parameter update, see Equation 2.36. Let m_i denote the momentum value at learning iteration i and ε be the momentum or the gradient decay factor.

$$m_i = \varepsilon m_{i-1} - \alpha_{lr} \nabla_\theta J(\theta_{i-1}) \tag{2.36a}$$

$$\theta_i = \theta_{i-1} + m_i \tag{2.36b}$$

The *Adam* algorithm [80] incorporates both the momentum and *RMSprop* style scaling of learning rates, as its major learning step. The name *Adam* reflects the use of "Adaptive Momentum" in the gradient descent update equations. In a sense, momentum is directly applied to *RMSprop* rescaling with a major modification of applying bias corrections to momentum and squared gradient terms. The gradient is sometimes referred to as the first moment and the square of the gradient is referred as the second moment. Let ρ_1 and ρ_2 represent the first moment and second moment discount factors. Let s_i denote the second moment value at learning iteration i. The correction in the moment and second moment terms is represented by $\hat{m}$ and $\hat{s}$ respectively. Then the parameter updated equation in the *Adam* algorithm can be represented as in Equations 2.37.

$$m_i = \rho_1 m_{i-1} + (1-\rho_1)\nabla_\theta J(\theta_{i-1}) \tag{2.37a}$$

$$s_i = \rho_2 s_{i-1} + (1-\rho_2)[\nabla_\theta J(\theta_{i-1}) \odot \nabla_\theta J(\theta_{i-1})] \tag{2.37b}$$

$$\hat{m} = \frac{m_i}{1-\rho_1} \tag{2.37c}$$

$$\hat{s} = \frac{s_i}{1-\rho_2} \tag{2.37d}$$

$$\theta_i = \theta_{i-1} - \alpha_{lr}\frac{\hat{s}}{\sqrt{\hat{r}}+\delta} \tag{2.37e}$$

Here, δ is a small constant value that provides numerical stabilization by avoiding division by zero. The *Adam* algorithm is regarded as a robust learning algorithm in terms of hyper-parameter tuning [58].

2.6 Imitation Learning: Data Aggregation Set

As previously shown in Figure 2.1, a fully autonomous flight control system for an aerospace vehicle involves three major algorithms: guidance, navigation and control (GNC). The whole system of algorithms is jointly implemented in a feedback system that minimizes errors in control, guidance, and trajectory tracking. To design a fully functional GNC autopilot, a very thorough tuning of guidance parameters, controller gains, and path planning constraints needs to be carried out before implementing the system onboard an aircraft. The inter-algorithmic couplings, and nonlinear dependence of the G, N, and C blocks makes the task of tuning and designing a robust autopilot very tedious. The field of machine learning has a wide set of adaptable tools that are capable of learning these types of complex mappings of inputs to outputs; one of the most widely useful implementations of adaptive control has been neural networks but this type of adaptive control has been elusive in fixed-wing autopilots due to added complexity and stability issues. In this section we propose training a neural network autopilot with one hidden layer to unify and replace the current GNC system by directly mapping aircraft input variables (dynamic states and distances to waypoints),

to control surface values. Hence the neural network learns to preserve the meaningful correlations among the individual GNC blocks by simultaneously updating its parameters (weights and biases) to output optimal control values.

The neural network autopilot is trained on supervised data from an "expert" GNC policy implemented via closed-loop 6-DOF flight simulations. The data is generated in a collaborative manner, using the neural network in training and the GNC system, by applying a modified data aggregation set, DAgger [118] algorithm. When using the standard implementation of DAgger, we find many shortcomings that prevent robust learning leading to a neural network autopilot that cannot successfully fly the aircraft. Several modifications to the algorithm are proposed to address the specific problem of a stable fixed-wing UAS flight with multiple conditions, such as steady-state straight line, level turn, and altitude and airspeed holds. For each modification, the results are quantified by flight time and other metrics of flight stability.

A novel supervised learning algorithm to train artificial neural networks to behave as fully autonomous trajectory tracking autopilot for a small fixed-wing UAS is developed. The algorithm provides a training paradigm that allows the neural network to learn a direct mapping of sensor states to control values. Strong empirical evidence of drawbacks in standard supervised learning approaches and proposed solutions to deal with the features of complex 6-DOF dynamics with varying flight conditions is demonstrated. This leads to certain modifications to data collection and training, DAgger variants: Monte-Carlo-DAgger (McDAgger) and Moving-Window-DAgger (MwDAgger).

For the algorithmic implementation and data collection the nonlinear model of a Skyhunter aircraft is used as the training environment, which is developed using Advanced Aircraft Analysis (AAA) [42] software, and further tuned using flight test data. The wing span of the aircraft is 5.9 ft, the aircraft weighs about 8.4 pounds, and flies at a cruise speed of 50.63 ft/s. It has been suitably modified to reinforce its primary structural parts, such as wing, fuselage, etc. with the addition of winglets for increasing aerodynamic performance. The details of this platform with all the modifications and an in-depth analysis of flight dynamics can be found in [21].

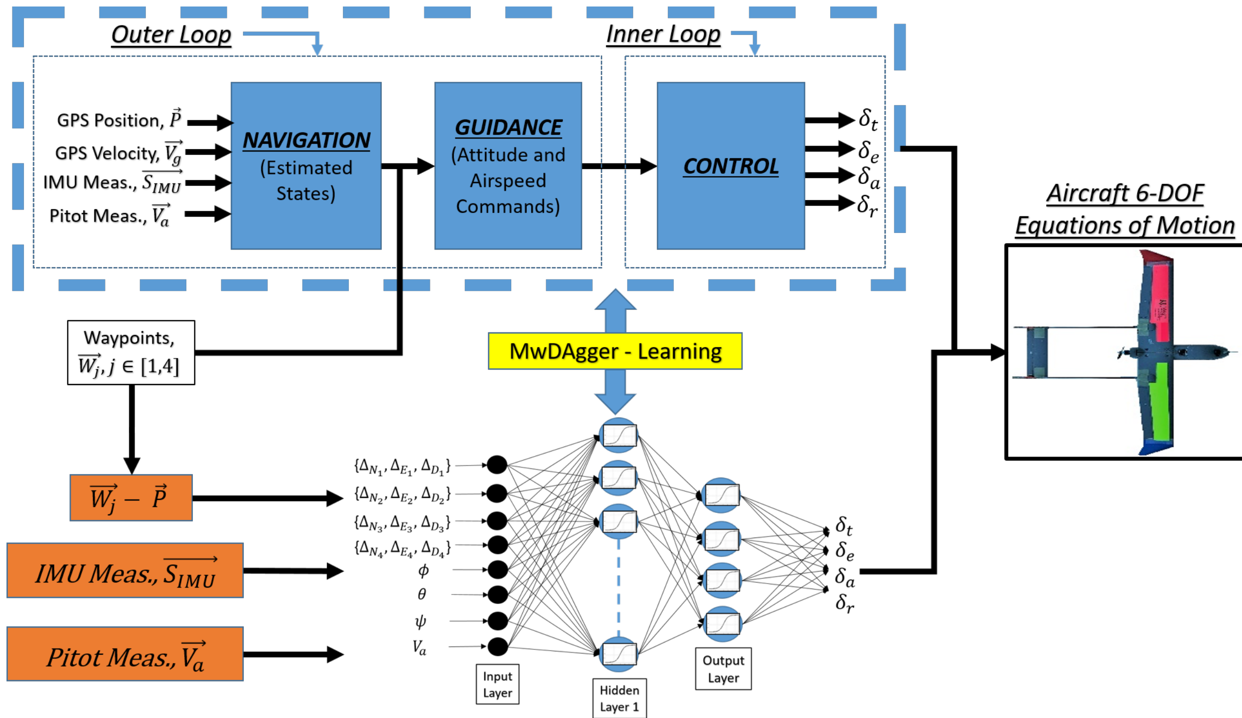


Figure 2.8: Guidance, Navigation, and Control Replaced by an Artificial Neural Network Autopilot

2.6.1 Imitation Neural Network Autopilot

The goal of the neural network autopilot is to fly user defined waypoints (in this case: four) while performing steady-state straight line flight, maintaining altitude and airspeed, including level turns. The ANN autopilot takes in high level input information and directly maps it to four control variables (throttle, elevator, aileron, and rudder).

The feedforward artificial neural network used for imitation learning consists of one hidden layer (35 neurons) and one output layer (4 neurons). All the neurons or nodes are assigned with tanh activation function. The ANN directly maps higher level features (namely *sensor states* and *distance to waypoints*) to end-effector control surface values, see Figure 2.8 where the box at the bottom represents the neural network. A feedforward neural network with one hidden layer is analytically proven to be sufficient to universally approximate any arbitrary measurable function [66, 39]. The primary goal here is to investigate the unification of GNC algorithms through ANN, therefore, the main focus is on higher level algorithmic development. Hence, the architecture

shown in Figure 2.9 would suffice to evaluate the methodology that fulfills this goal of unification. For training purposes, MATLAB's Neural Network toolbox [95] is used. From this Neural Network toolbox, scaled conjugate gradient descent [102] algorithm, see Section 2.5.2, is used for training the parameters of the neural network.

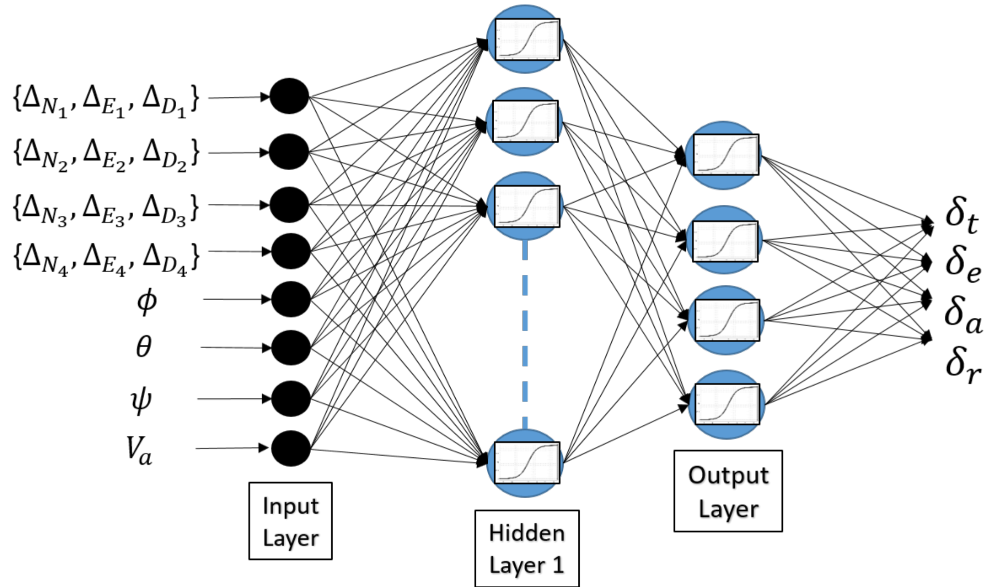


Figure 2.9: ANN architecture

Designing a neural network architecture also involves choosing appropriate input variables that are uncorrelated, so that it can learn a meaningful mapping to outputs. In the case of an aircraft, the inputs are the dynamic states and outputs are generally four control variables. An aircraft involves multiple dynamic states that are pertinent to its stable operation, that should be evaluated properly before choosing them as inputs to a neural network, for example the states can be chosen from a set of velocity (V_a), angle of attack (α), side-slip angle (β), attitude angles (ϕ, θ) and attitude rates (P, Q, R). It is stated here with great emphasis, that selection of input states for this research was extremely challenging. In general, neural networks can learn relatively more efficiently if the cross-correlations among the inputs are as low as possible. The inputs selected to design a neural network autopilot should represent maximum causal variability across the outputs, and hence possess all the characteristics that define the physical dynamic system, in order for the model to learn quickly and, ideally, learn a set of parameters that generalize well to real world flight.

The "expert" GNC policy, details in Reference [76], consists of an intelligent path planning algorithm that uses information from four waypoints (user defined) and the initial position of the aircraft (where autopilot is turned on) to decide which waypoint to initially follow based on minimum change in heading angle. Path planning accounts for waypoint completion and switching to the next waypoint as the aircraft progresses in its flight trajectory. For our ANN autopilot, we provide the North-East-Down (NED) distances to each of the four waypoints, as inputs, so that it will learn to path plan and automatically switch around the waypoints. This ΔNED input to the ANN autopilot, which consists of 12 variables can be defined as follows:

$$\Delta_{K_j} = \overrightarrow{W_j} - \overrightarrow{P}, \ \ K \in \{N, E, D\}, \ \ j \in [1, 4] \tag{2.38a}$$

$$\overrightarrow{W_j} = \{N_j, E_j, D_j\} \tag{2.38b}$$

$$\overrightarrow{P} = \{N_a, E_a, D_a\} \tag{2.38c}$$

Here, $\overrightarrow{W_j}$ is one of the waypoints: $\{N_j, E_j, D_j\}$ and $\overrightarrow{P}$ is the aircraft location: $\{N_a, E_a, D_a\}$. In addition to (ΔNED), it would be sufficient to provide the aircraft heading angle as part of the input representation to capture all the information needed by the path planning block in the "expert" GNC policy. The guidance algorithm of the "expert" GNC policy provides the roll angle, pitch angle and airspeed commands ($\phi_{cmd}, \theta_{cmd}, \ V_{a_{cmd}}$) as controller inputs. Therefore, it should be adequate to provide these aircraft states: roll, pitch and airspeed (ϕ, θ, V_a), as additional inputs to the ANN autopilot.

Using the above inputs, the neural network is trained to output control values for throttle (δ_t), elevator (δ_e), aileron (δ_a), and rudder (δ_r), with the goal to follow a race-track type trajectory composed of four waypoints shown as black dots in Figure 2.11.

A single training data point $x^{(i)}$ is defined as the following input vector:

$$\mathbf{x^{(i)}} = \left[\Delta_{N_j}^{(i)}, \Delta_{E_j}^{(i)}, \Delta_{D_j}^{(i)}, \phi^{(i)}, \theta^{(i)}, \psi^{(i)}, V_a^{(i)}\right], \ \ j \in [1, 4] \tag{2.39}$$

Here, $\Delta_{N_j}, \Delta_{E_j}, \Delta_{D_j}$ are the NED distances of the aircraft from the jth desired waypoint, ϕ, θ, ψ are the attitude angles, and V_a is the airspeed. $y^{(i)}$ is output or target data value at time i and consists of the control surface variables of the aircraft, namely: throttle, elevator, aileron and rudder.

$$\mathbf{y}^{(\mathbf{i})} = \left[\delta_t^{(i)}, \delta_e^{(i)}, \delta_a^{(i)}, \delta_r^{(i)}\right]. \tag{2.40}$$

In Equation 2.39, it is important to note that the value of j changes from 1 to 4. Therefore, the inputs comprise four distances to waypoints as 3D NED vectors. With the other state inputs, the inputs add up-to a total of 16. These inputs are chosen for a specific case of four waypoints only; however, it is noted here that the value of j can be arbitrary, that is, it can go up-to a positive integer n. Hence, by changing the input architecture a neural network autopilot can be trained to fly the aircraft to follow any number of waypoints.

The input-output labelled data is generated using the "expert" GNC policy. There are multiple ways to gather this data, the simplest way would be to fly the GNC on its normal race-track flight path, wait for the flight to finish and use the whole flight data to train the ANN autopilot. This approach detailed in Section 2.6.3 is very naive, and cannot train a working ANN autopilot, due to error accumulation. Therefore, it becomes essential to use algorithms such as DAgger, that allow for data aggregation for those flight points where error accumulation starts building up. However, it is shown in the following sections that this technique also fails for complex tasks of flight paths consisting of multiple conditions (steady-state straight line, level turn, and altitude and airspeed holds). Various and incremental modifications are experimentally applied and tested to the DAgger algorithm, which finally manifests into Moving Window DAgger (MwDAgger) approach that is able to train an ANN autopilot for stable ans sustained flights with steady-state straight line, level turns, and airspeed and altitude holds.

2.6.2 DAgger for Fixed-Wing Aircraft

The basic idea behind the DAgger algorithm [118], or data aggregation, is to add new and perturbed training data, which become labeled inputs and outputs, from an expert policy. The perturbed data is novel to the neural network as well as the expert policy, see Figure 2.10. In this research, the ANN is trained iteratively using the newly added data, until it is capable of flying in the flight simulation. The new data is generated in a coordinated way by the neural network and the "expert" GNC policy. First step is to perform supervised training on the ANN using fixed trajectory data generated by the GNC independently. However, this trained ANN does not perform identical to the GNC, and hence generates control outputs with small errors that accumulate over the course of a flight simulation. These slightly different outputs generated by the trained ANN, become the inputs to the aircraft 6-DOF model, which is propagated to generate a new perturbed state. These new states provide novel input to the GNC and when paired with the GNC control decisions become new labeled data from which the neural network can use for further training.

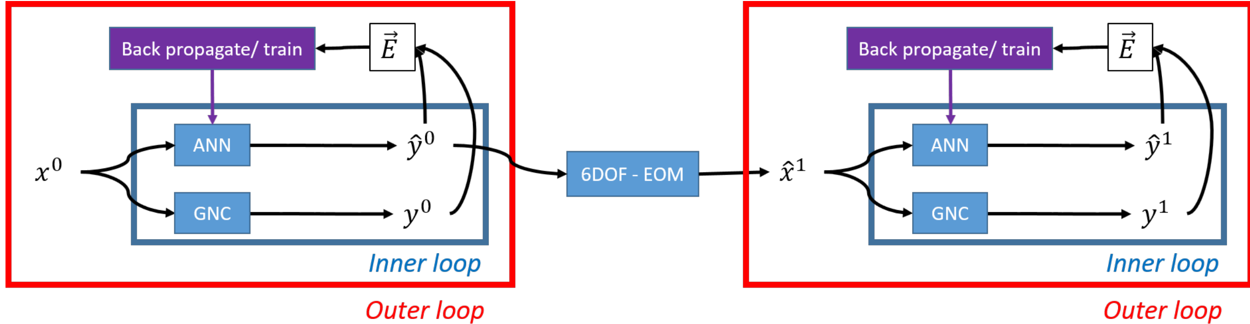


Figure 2.10: Data Aggregation Concept: Novel Data and Error Propagation

In other words, the ANN predicted controls $\hat{y}^{(t)}$ are fed to the 6-DOF aircraft model, which in turn puts the aircraft in a new state $\hat{x}^{(t+1)}$. This new data point is then fed into the GNC system such that a new training data pair, $\mathrm{D}^* := \left\{\hat{x}^{(t+1)}, \mathbf{GNC}\left(\hat{x}^{(t+1)}\right)\right\}$, is generated, which has been influenced by ANN control decisions. This new data point is then added to the original data set, and the neural network parameters are updated based on the complete data set. This process is continued iteratively until the neural network is considered trained via a stopping criterion which

Algorithm 3 DAgger for Aircraft

1: Initialize state $x^{(0)}$, D := [ ], T
2: **for** t = 0, T **do**
3: $y^{(t)}$ = **GNC**$\left(x^{(t)}\right)$
4: $x^{(t+1)}$ = **6-DOF** $\left(y^{(t)}\right)$
5: D := D $\cup$ $\left\{x^{(t)}, y^{(t)}\right\}$
6: **end for**
7: Initialize ANN
8: **while** stopping criteria not met **do**
9: ANN = **train_model**$(X, Y \subset D)$
10: Initialize state $x^{(0)}$, t = 0
11: **while** **safe**$\left(\hat{y}^{(t)}\right)$ & **safe**$\left(\hat{x}^{(t)}\right)$ & $t <$ T **do**
12: $\hat{y}^{(t)}$ = **ANN**$\left(x^{(t)}\right)$
13: $\hat{x}^{(t+1)}$ = **6-DOF**$\left(\hat{y}^{(t)}\right)$
14: D* := $\left\{\hat{x}^{(t+1)}, \textbf{GNC}\left(\hat{x}^{(t+1)}\right)\right\}$
15: $t = t + 1$
16: **end while**
17: D = D $\cup$ D*
18: **end while**

is defined as a small threshold difference between the GNC and ANN predicted control decisions. In addition to these more traditional stopping criteria for training, the ANN is also tested on its ability to fly the aircraft (see Figure 2.14 and Table 2.1).

The detailed steps of this methodology are given in Algorithm 3. The data is sampled and collected at 20Hz update rate. The first six lines of the algorithm show data collection process from the "expert" GNC policy for doing standard supervised learning. Step eight starts the data aggregation process and simultaneous training which is carried out until 1) the error between y and $\hat{y}$ is near 0, and 2) the ANN autopilot has flown the aircraft for full flight time that is "T" seconds. Steps 12, 13 and 14 show the generation of novel data which is unlikely to have been produced by the "expert" GNC policy alone. This process is repeated until any control predictions or aircraft states are deemed unsafe by the validation system. The weights of the neural network are updated in line 9 and will include control decisions from the "expert" GNC policy as new training data. The DAgger algorithm becomes the basis for the McDAgger and MwDagger algorithms as described in the subsequent sections.

2.6.3 Simulation Setup and Data Aggregation Models

A general simulation flight test setup runs the "expert" GNC policy to fly the aircraft, exciting and propagating 6-DOF aircraft equations of motion (EOM). Four waypoints are setup by the user and the GNC system guides the aircraft in a racetrack pattern around these four waypoints while maintaining altitude and airspeed, as shown in Figure 2.11. The coordinates of the four waypoints are selected and converted from geodetic coordinates from a flight test location in Lawrence, Kansas that is utilized for research flight tests by KU's flight research team, shown as black dots in Figure 2.11.

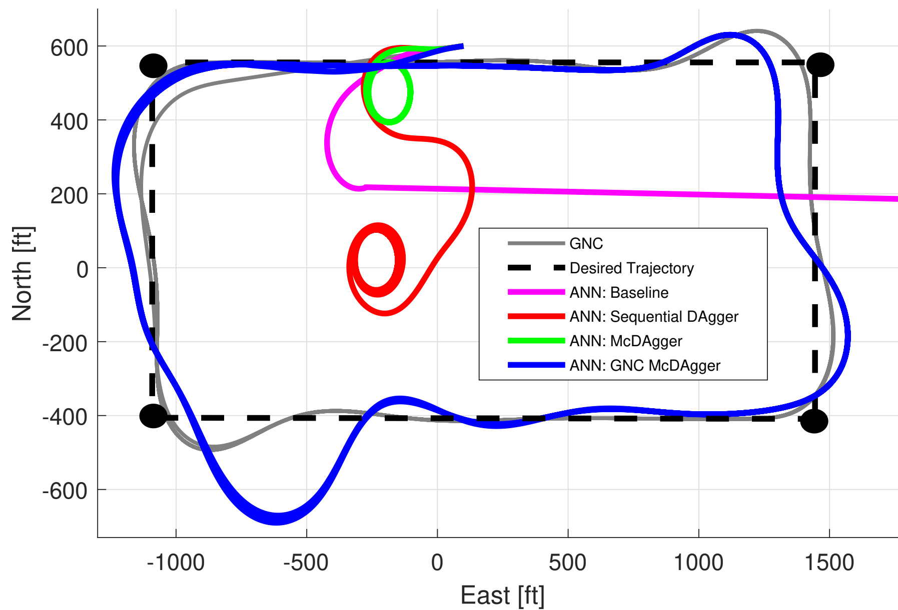


Figure 2.11: All models are evaluated in a simulated flight test setting to follow the dashed desired waypoint path, and the corresponding aircraft trajectories are compared.

Evaluating the distance that the UAS has to cover in about one racetrack loop around the waypoints ($\approx$ 7,000 ft), the time taken is approximately 140 to 150 seconds at a cruise airspeed of 50.63 ft/s. Depending on the initial aircraft position with respect to the waypoints, at which the flight starts in the autopilot mode, it takes about 50 to 100 seconds for the aircraft to converge on to the desired trajectory and altitude. For collecting an initial dataset and deciding on a reasonable

flight time to evaluate the ANN, about four and a half loops of racetrack trajectory was determined to be appropriate. Therefore, a flight time of 750 seconds was chosen as the target flight time for validation and data collection for the neural network autopilot.

Five different types of data aggregation algorithms are considered that train the ANN autopilot. All the ANN autopilot models produced by their respective data aggregation algorithms, are evaluated and addressed in the context of their results, as shortcomings of the preceding models motivate the changes made in later models. These models are defined as follows:

- **Baseline**: Supervised Learning with fixed data set. Only first nine lines from Algorithm 3 are utilized for this model, skipping the while loops and lines 10 to 17.

- **Sequential DAgger**: See details in Algorithm 3.

- **Monte Carlo DAgger (McDAgger)**: Select a random time t between 0 and T. Use the "expert" GNC policy until t instead of executing lines 12 to 14 in Algorithm 3, and then start normal DAgger after time t, such that the ANN is encouraged to fly at different time points throughout the course of flight.

- **GNC McDAgger**: Select a random time t between t_{delay} and T, that is $t_{delay} \leq t < T$. The GNC is allowed to fly the aircraft to a steady-state until delay time (t_{delay}), and then the ANN autopilot is invoked at some random time following Monte-Carlo data aggregation.

- **Moving Window DAgger (MwDAgger)**: DAgger for a fixed time window of flight, allowing to focus learning on a fixed flight path (time-based). After learning a fixed path the time window is incremented and the ANN autopilot's parameters are re-initialized, until the whole flight path is covered.

For all these five models, the "expert" GNC policy used to provide data for training the ANN autopilot is detailed in Reference [76]. This GNC policy is implemented in MATLAB SIMULINK [94] simulator, and is set up to accept 3D waypoints as inputs (user defined), and follow them by exciting 6-DOF equations of motion in a closed-loop time-series flight simulation. The GNC

policy tracks the waypoints based on their indices, while maintaining altitude and airspeed, and performs right or left steady-state turns based on the desired heading computed from waypoint path lines. The ANN autopilot is trained from this type of flight data, specifically a case of four waypoints laid out as vertices of an approximate rectangular box that exhibits a racetrack flight pattern and hence represents data for multiple flight control conditions: steady-state straight line, level turn, and altitude and airspeed holds.

Baseline Model: In this model, the ANN is trained via supervised learning using labeled data generated independently by the "expert" GNC policy. The training is carried out until the mean squared error (MSE) is (≈ 0) or the gradient (used to update the weights during training) is below ($\approx 1e-20$). Even when applying cross-validation by dividing data into training (70%), validation (20%) and testing (10%) sets, and means to reduce over-fitting, it is found that the MSE is not a good indicator for the neural network's ability to mimic the GNC system for a stable flight.

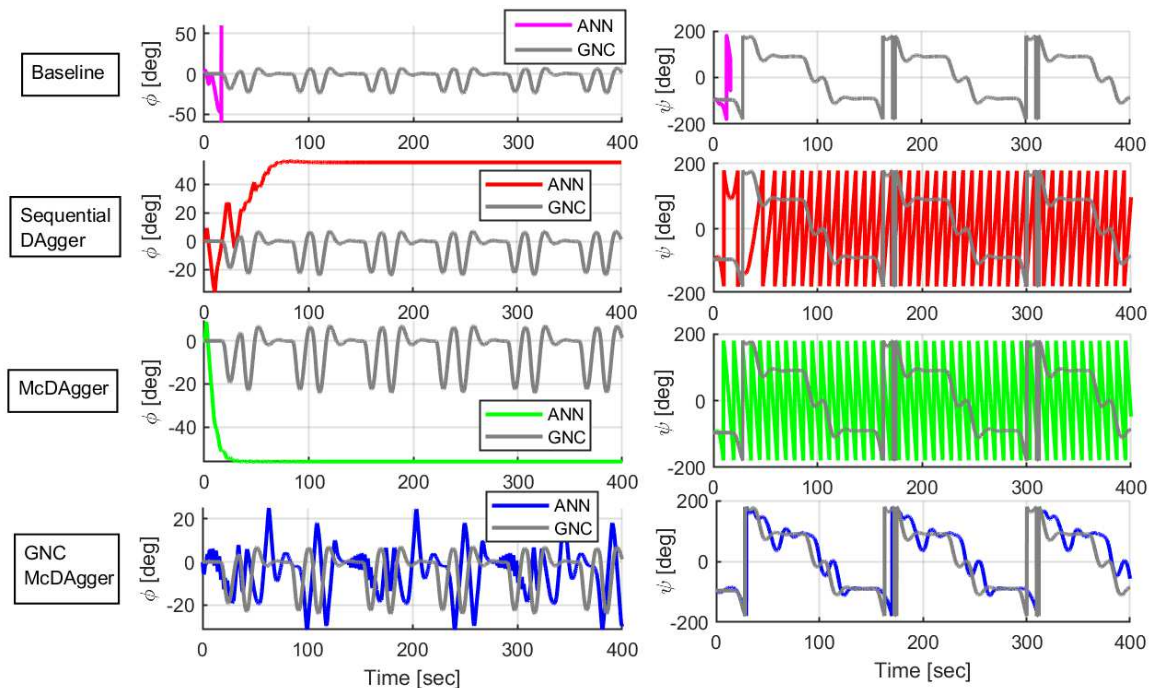


Figure 2.12: Aircraft Lateral states comparison for first 4 models. Left column shows roll angle (ϕ) and right column shows yaw angle (ψ).

The neural network baseline model, when tested on flying the aircraft via interaction with the 6-

DOF EOM in closed-loop simulations, chose a flight trajectory that was nowhere close to the GNC path (see the magenta plot in Figure 2.11). Since the ANN autopilot, trained using the baseline algorithm, cannot predict controls with absolute zero error with respect to the "expert" GNC policy, the aircraft enters a different state in a consecutive time-step, thereby error is accumulated over closed-loop time-series simulation leading to unstable control values predicted by the baseline ANN autopilot, and the aircraft becomes unstable within 16.375 seconds. This instability in aircraft flight, using the baseline model, is depicted in the top pane of Figures 2.12 and 2.13.

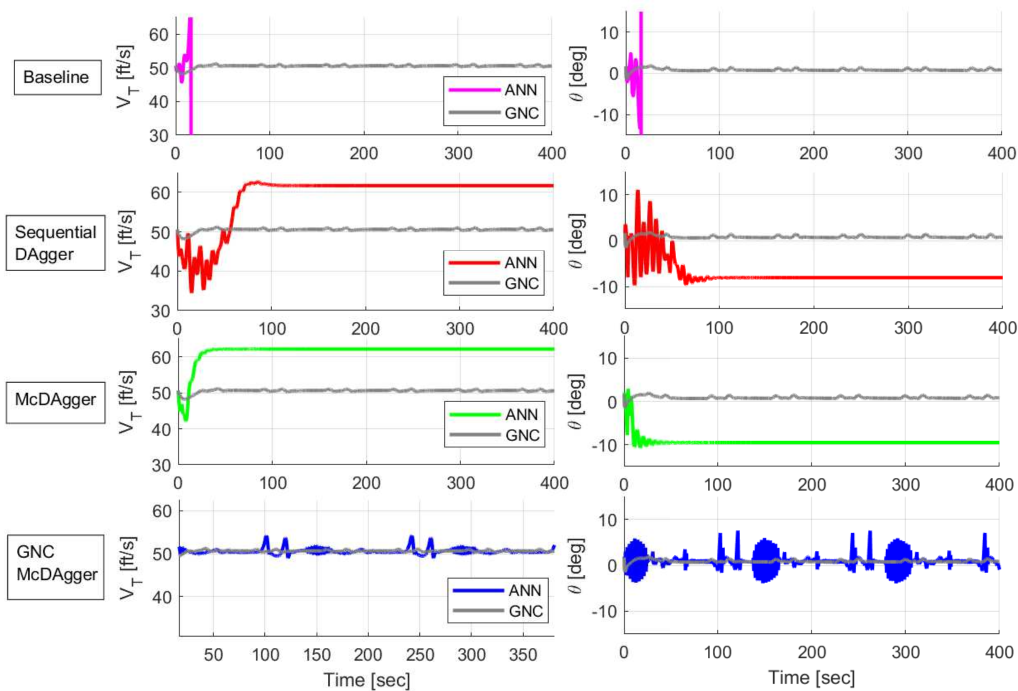


Figure 2.13: Aircraft Longitudinal states comparison for all models. Left column shows total velocity (V_T) and right column shows pitch angle (θ).

Sequential DAgger Model: The baseline model, a standard ANN autopilot trained in a supervised manner, is not capable of mimicking the GNC controls with high enough accuracy and precision to fly the aircraft. One way to address such an issue is to allow the ANN autopilot to make control decisions that result in novel states but to return to the "expert" GNC policy in order to find out what actions are optimal in such states. *Sequential DAgger*, as explained previously, allows the neural network to fly the aircraft until it is uncontrollable or if the control inputs to 6-DOF, as outputted by the ANN, become physically unrealistic. The stopping criterion for training is set

as root mean squared errors (RMSE) thresholds between the GNC controls and ANN controls: $e_t <= 1e-3$. This number is found by manually looking at the ANN predicted controls from the baseline model and observing when the data fit is reasonably accurate without any extreme values or oscillatory behavior.

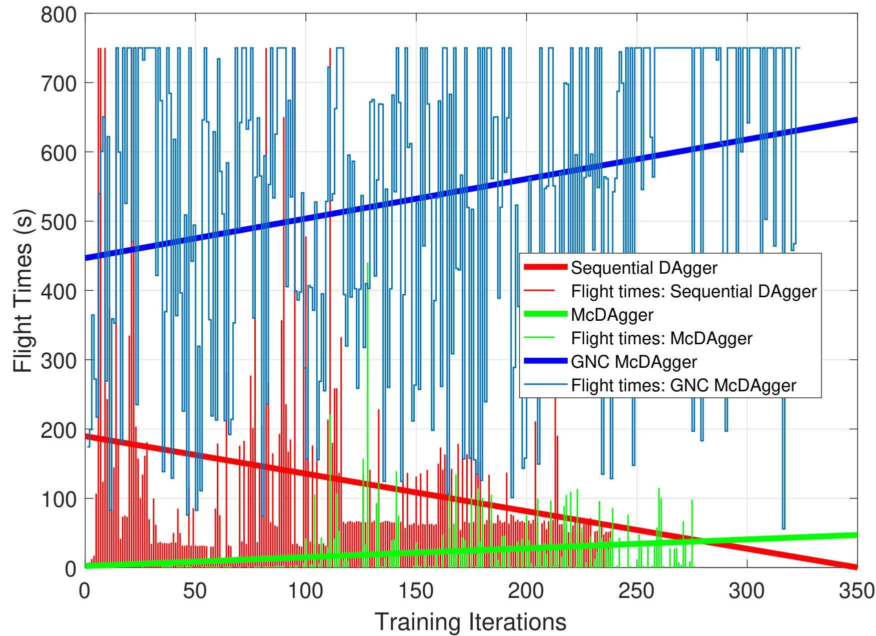


Figure 2.14: Flight time slopes for DAgger training, comparison for all models.

While adding data in each *sequential DAgger* iteration, it was found that the aircraft is able to fly and the ANN autopilot shows signs of learning. However after certain number of iterations, the ANN autopilot is not able to control the aircraft beyond a specific flight time (≈ 60-70 seconds). A general trend of decrease in flight path learning is observed. This trend is shown in Figure 2.14, where the sequential DAgger graph is in red color. The ANN autopilot is able to fly the aircraft for complete 750 seconds in a few select cases, but it flies the aircraft on a path which is completely off-track and loses altitude continuously. The RMSE values never decrease below 0.0051 and thus the training procedure is manually stopped at DAgger iteration 239.

Monte Carlo DAgger (McDAgger) Model: The intermediate training and flights for the Sequential DAgger ANN autopilot have large trajectory tracking errors (see Figure 2.11) and high

levels of instability in flight as depicted by second rows of Figures 2.12 and 2.13. Figure 2.15 visualizes the data aggregation steps (lines 12 to 14 in Algorithm 3) such that the light to darker color transitions indicate the increment in successive DAgger training iterations. As can be seen from the plot, some of the light colored trajectories show significant amount of flight time (500 - 750 seconds), however, the darker colored trajectories are limited to around (60 - 70 seconds). One of the major reasons that the training appears to converge around 60 seconds, and the slope of flight time is negative, is apparent by looking at Figure 2.15, that is, the additional data being aggregated to the original data set is non-uniformly distributed around the four waypoint trajectory. Or in other words, the data being added is more or less concentrated around the first 60 seconds of the flight path resulting in the model over-fitting to the earliest part of the flight. To overcome this issue, a straight-forward modification to the original DAgger algorithm was applied, in which the aircraft starts data addition at a different state at each iteration, selected by randomly sampling the starting time of the simulation, this would result in updating lines 10 to 16 in Algorithm 3.

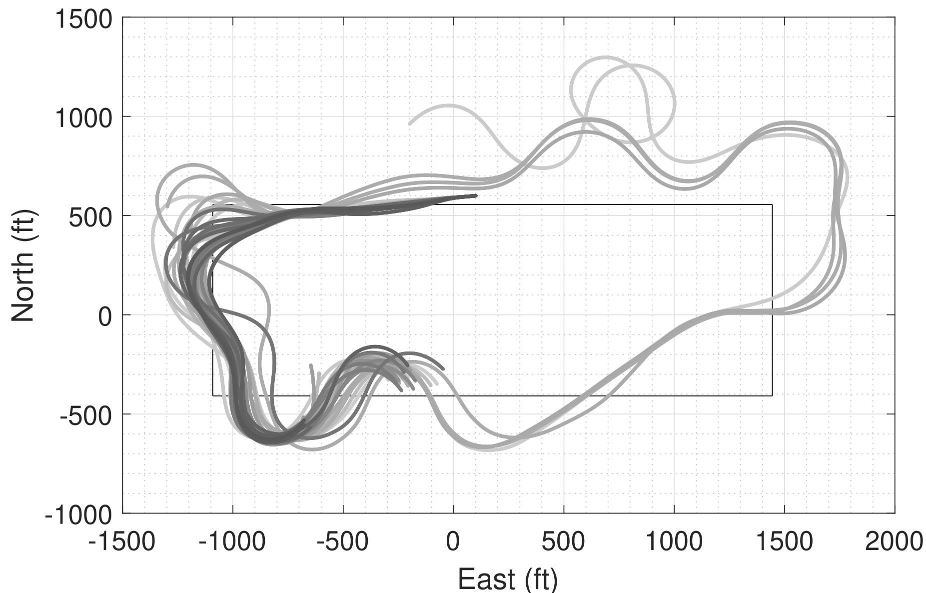


Figure 2.15: Sequential DAgger: Intermediate trajectories flown by ANN during data aggregation.

The modification of choosing a random time to initialize data collection, and thus a corresponding random state ($x^{(t)}$) of the aircraft that does not have the accumulated errors from all previous

flight decisions, leads to the terminology: Monte Carlo DAgger. Surprisingly, the flight times per McDAgger iteration were found to often be even shorter than for the Sequential DAgger autopilot but it is noted that the slope (e.g. the trend in flight length over training time) is at least positive (see the green line in Figure 2.14), indicating that with longer training or more computational power this model may have eventually converged to the required 750 seconds of flight.

GNC McDAgger Model: The "expert" GNC policy used to generate correct input-output mapping data, is designed in such a way that it does not react to errors abruptly and always ensures a very smooth transition if initialized from a large state or control error. When the simulation for GNC and 6-DOF, is initialized at a random state for the aircraft, it will always have a warm-up time (t_{delay}) to converge to the correct cruise altitude and hence the desired trajectory. In other words, even if the simulator is initialized at an exact point on a previously flown trajectory, the GNC outputted control values will not be the same as the controls on the original path, and hence the aircraft flying with GNC policy does not follow a particular pattern of flying when initialized randomly. This causes data distribution mismatch issues, because the neural network now is learning from policy decisions that are variant even for the same set of states. Hence the flight time for McDAgger autopilot stays around 40 seconds (Figure 2.14).

If data addition is desired to be initialized at a random location on the original trajectory, the GNC induced flight behavior needs to be kept effectively the same, or at least the flight trajectory the ANN is trying to learn (i.e. post initialization phase of the GNC). This is addressed by initializing the simulator from a fixed initial condition ($x^{(0)}$: aircraft state). The GNC then flies from this initial condition until a randomized time t which is at least delayed for the warm-up time (t_{delay}). After the GNC has flown the aircraft up to this random point, then the ANN is invoked to take over the flight and McDAgger data addition resumes. This modification to McDAgger algorithm ensures proper data aggregation and is mainly applied to work around this specific "expert" GNC policy. This modified version of McDAgger algorithm is termed as model: GNC McDagger and would require two main changes to Algorithm 3, namely t would be selected at random after adding delay time and the GNC would be initialized to run from $x^{(0)}$ until $t-1$. The slope of flight

times during GNC McDAgger iterations is shown in Figure 2.14 in blue color and indicates that this training procedure is allowing the ANN autopilot to increase flight time steadily by adding data that is useful for ANN learning. The flight times are consistently increasing and reaching up to 750 seconds which is the maximum flight path time. Additionally, while the model is not following the GNC trajectory exactly, the ANN learning curves, depicted in Figure 2.16 do suggest that the flights are getting longer and more closely aligned with the desired trajectory as training proceeds.

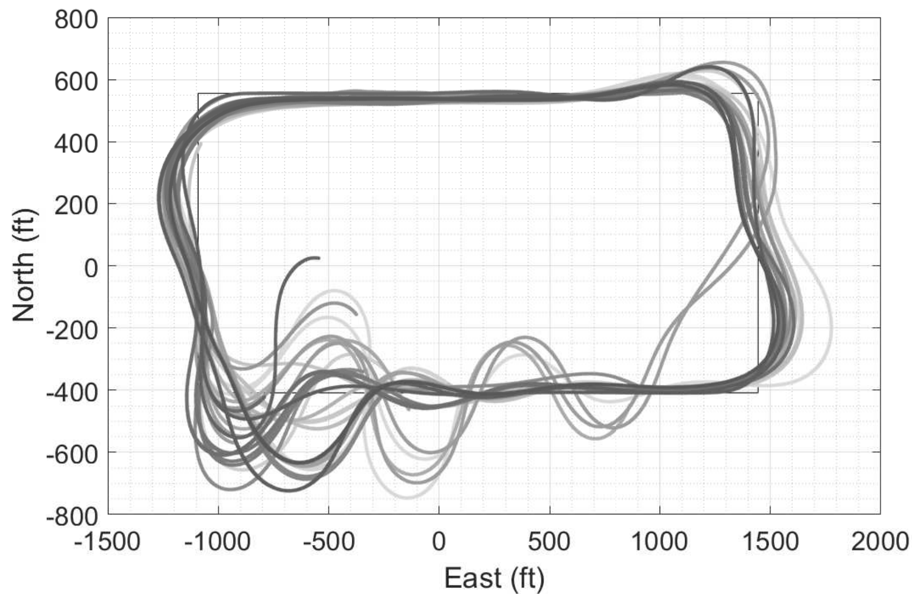


Figure 2.16: GNC McDAgger: Intermediate trajectories flown by ANN during training: ANN Learning Curves

On comparing the learning curves from models: Sequential Dagger and GNC McDAgger (Figures 2.15, 2.16), it can be clearly observed that the flight paths during learning are closer to the desired trajectory for the GNC McDAgger model. After training is completed, each ANN autopilot is subjected to a simulated flight test, starting the aircraft at the same initial state used for training and without any involvement of GNC. The aircraft 6-DOF equations are excited by ANN outputted control values in a time-series closed-loop simulation. We can then compare the trajectory as flown by all the ANN autopilot models (trained via DAgger variants) with the "expert" GNC trajectory. Results for all the five models are shown in Figure 2.11. It is observed that the

overall flown flight path for only the GNC McDAgger ANN autopilot is satisfactory with respect to GNC flown path. This ANN autopilot has approximately ± 20 ft of North distance errors along the East-West waypoint legs of the desired path, and approximately $\pm$ 80 ft of East distance errors along the North-South legs. The largest tracking errors are observed around the South-West corner of the desired trajectory, which is about 267 ft. These trajectory errors are not ideal in terms of a realistic flight test and must be improved before putting this ANN autopilot on an actual aircraft, but a measurable progress is shown in improving imitation learning in order to construct a ANN autopilot.

For the simulated flight tests with the Baseline, Sequential DAgger, McDAgger and GNC McDAgger neural networks, the aircraft states are analyzed in both longitudinal and lateral directions which are depicted in Figures 2.13 and 2.12, respectively. Figure 2.13 shows longitudinal states, namely: total velocity and pitch angle of the aircraft for all four ANN autopilot models. Oscillations are observed in both the aircraft states for the GNC McDAgger neural network. There are low frequency (≈ 1.2 Hz) pitch oscillations observed on the aircraft. It is also worth mentioning that the aircraft is able to hold the desired cruise altitude of 300 ft.

In the bottom panel of Figure 2.12, some lateral states, namely, roll angle and yaw (or $\approx$ heading) angle, are shown for the GNC McDAgger ANN autopilot and compared to the GNC simulation states. There are again low frequency oscillations observed in the roll angle of the aircraft, which are not ideal for a realistic flight, and must be addressed before putting this autopilot on a real aircraft.

Table 2.1 shows the statistics of training for the first four models. In terms of computational complexity, the GNC McDAgger is the most expensive, and it also uses the largest amount of data set to train the ANN autopilot.

Table 2.1: Comparison of ANN autopilots trained using different imitation learning paradigms. Model evaluation focuses on training feasibility and flight time during validation and testing.

Model	DAgger Epochs (Training)	Training Examples	Training Time	Slope of Flight Times	Flight Time (seconds)
Baseline	N.A.	1.50e4	0h 4m	N.A.	16.375
Seq. DAgger	239	6.08e5	7h 35m	-0.54132	750
McDAgger	279	1.27e5	1h 52m	0.12801	48.5
GNC McDAgger	325	$> 1e6$	9h 14m	0.57104	800+

2.7 Moving Window DAgger: Addressing Oscillations

The GNC McDAgger ANN autopilot was able to combine and imitate the complete functionality of the GNC policy. However, low frequency oscillations in the control surfaces (and consequently aircraft states) raised serious questions on the reliability and safety of using these imitation neural networks for real flight applications. These oscillations are imparted due to oscillatory neural network outputs, but it was found that the major shortcoming was in the data aggregation algorithm used.

The low frequency state oscillations indicate that the ANN autopilot was unable to make correct control decisions for small state perturbations, and would react only when the aircraft state (and hence 2D trajectory) would drift enough for the ANN to take corrective action. To mitigate this issue, the sequential data aggregation was applied to a fixed time-window, and the ANN was trained iteratively over this window until the stopping criterion was met. This aggregated data along a short time window is saved in a data buffer, and the ANN parameters are reinitialized. The time-window for data aggregation is increased, and new data is added to the data buffer, and the ANN is re-trained for this time-window with the whole data buffer. The process is repeated until the whole flight path is covered by the time-window, increasing up-to the maximum flight time. Algorithm 4 shows the details of the Moving Window DAgger (MwDagger) algorithm applied to a fixed-wing

aircraft with an "expert" GNC policy. It took 753 DAgger iterations, and the algorithm aggregates a total of 1.52e5 data points, taking a total of about 8 hours and 55 minutes. This ANN autopilot is able to fly indefinitely around the waypoint path, maintaining stable states at all times. Since this is the first neural network autopilot that can mimic "expert" policy decisions stable and accurately, the neural network is termed as "Imitation Model 1". The detailed results are presented in Section 4.2.1.

Algorithm 4 Moving Window DAgger for Aircraft

1: Initialize state $x^{(0)}$, D := [ ], T
2: **for** t = 0, T **do**
3: $y^{(t)}$ = **GNC**$\left(x^{(t)}\right)$
4: $x^{(t+1)}$ = **6-DOF**$\left(y^{(t)}\right)$
5: D := D $\cup \left\{x^{(t)}, y^{(t)}\right\}$
6: Initialize w and w_s
7: **end for**
8: **while** stopping criteria not met **do**
9: Initialize ANN
10: ANN = **train_model**$(X, Y \subset D)$
11: Initialize state $x^{(0)}$, $t = 0$ and $t_f = w$
12: **while safe**$\left(\hat{y}^{(t)}\right)$ & **safe**$\left(\hat{x}^{(t)}\right)$ & $t < t_f$ **do**
13: $\hat{y}^{(t)}$ = **ANN**$\left(x^{(t)}\right)$
14: $\hat{x}^{(t+1)}$ = **6-DOF**$\left(\hat{y}^{(t)}\right)$
15: D* := $\left\{\hat{x}^{(t+1)}, \textbf{GNC}\left(\hat{x}^{(t+1)}\right)\right\}$
16: $t = t + 1$
17: **end while**
18: **if** RMSE$\left\{\hat{y}^{(t)}, \textbf{GNC}\left(\hat{x}^{(t+1)}\right)\right\} \leq \Delta_c$ **then**
19: $w = w + w_s$
20: **end if**
21: D = D $\cup$ D*
22: **end while**

2.8 Grid Search for Optimal Neural Network Architectures

A basic grid search algorithm is designed for optimizing the neural network architecture. This search runs at the top level of a data aggregation and training procedure. The grid search is designed to mainly tune two important neural network parameters: (1) number of neurons or nodes

and (2) number of tanh hidden layers. An additional parameter is introduced in the overall training paradigm that defines the maximum number of DAgger training iterations per neural network architecture ($max_{i_{ann}}$). Before performing the training step in Algorithm 3 (line 6), a logic check is implemented that compares the number of training iterations with the maximum allowed iterations per neural network architecture. If the current training iteration (t_i) exceeds $max_{i_{ann}}$, then a neuron is added to the neural network architecture which is being trained currently. Subsequently, if the number of neurons in the current architecture exceeds maximum neurons (max_{nodes}), then one hidden layer is added to the overall architecture.

Algorithm 5 Grid Search: Optimal Neural Network

```
1:  Initialize min and max neurons: min_nodes, max_nodes
2:  Initialize hidden layers: HL
3:  Initialize max training iterations per ANN architecture: max_i_ann
4:  if t_i == max_i_ann then
5:      D = empty()
6:      if nodes < max_nodes then
7:          nodes = nodes + 1
8:      else
9:          HL = HL + 1
10:         nodes = min_nodes
11:     end if
12:     Restart Training and Data collection
13: end if
```

This straightforward grid search algorithm is designed around the data aggregation concept, meaning if different neural network architectures are used, then the data-set added to D will be different at each training iteration. Since the data is collected while the neural network is being trained and each neural network architecture, even with just one node addition, would output just slightly different controls, the overall data addition will become dissimilar. And, hence the training will have to be restarted for a new architecture, and previous data-set will have to be discarded (see line 5 in Algorithm 5). Due to this, a fixed data-set generated by any DAgger variant cannot be utilized to train and evaluate different neural network architectures, because the data generation itself is dependent on the neural network used.

Algorithm 5 can be used as a sub-routine inside Algorithm 3 before line 6 and also with other

variants of DAgger. The fundamental idea behind this algorithm is that if the data addition per training iteration supports better performance (given a neural network architecture), then the number of iterations are automatically reduced and hence a particular network is superior than its previous counterparts. The algorithm is not designed to reduce computation time, but to find the optimal neural network architecture based on a user defined tuning parameter: $max_{i_{ann}}$. Therefore, the optimal ANN is only as good as the pre-defined number of explored iterations and will be limited in performance if a relatively large value is used.

2.9 Reinforcement Learning

Reinforcement learning describes an approach to learning from data in which an agent learns directly from interactions with an environment. There are two main elements within the reinforcement learning problem framework: agent and environment. Agent is the entity that takes actions to control the environment and learns through this interaction process. Environment is a dynamical system that reacts to the agent's inputs and changes its state accordingly while providing a reward feedback signal to the agent. In this research, the environment consists of the equations of motion of the aircraft and reward function. For a control system problem, the environment can be thought of as a generic mapping function (discretized dynamics) that relates the current state of the system to the next state, given a control input that excites the system, see Equations 2.41a and 2.41b, while numerically evaluating the new state using a reward function.

$$X_{t+1} = f(X_t, U_t) \tag{2.41a}$$

$$r_{t+1} = R(X_t, U_t) \tag{2.41b}$$

Equation 2.41a is a dicretized representation of a nonlinear dynamic system which is the 6-DOF model of the aircraft, see Section 2.2, where state X_{t+1} is computed as a function of both the previous state (X_t) and the control input (U_t). The reward function computes a numerical scalar which is a function of current state and controls. The agent is a learner that can be as simple as a linear regression model or can be a more general function approximation element

such as an artificial neural network. The agent learns its internal representations of parameters through maximization of long term rewards from the environment. A general architecture of a reinforcement learning problem setting is shown in Figure 2.17.

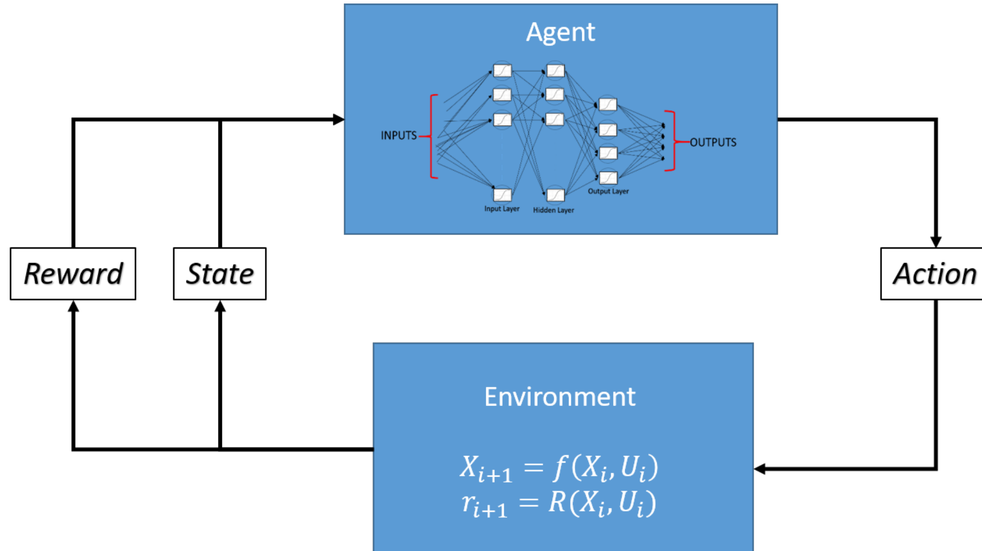


Figure 2.17: Reinforcement Learning Problem Framework

Reinforcement learning frameworks are typically applied to discrete time, stochastic sequential decision problems known as Markov Decision Processes (MDPs) [138]. Sequential decision problems that satisfy the "Markov Property" are known as MDPs. The Markov property states that the current state of a dynamical system depends only on the previous state, enabling the dynamics of the system to be a one-step process. This Markov property is a very important factor in reinforcement learning problems because the actions and rewards are dependent on current state only.

2.9.1 Preliminaries

In a typical reinforcement learning setting, the agent consists of a policy function (controller) that makes the decisions, and a value function (cumulative feedback) that quantifies the "goodness" of the decisions of the policy. The value function which is defined as the expectation of future rewards if a current policy is followed, arises from the learning goal of maximizing cumulative

long-term reward. The reward function (or immediate reward) is usually designed by the user utilizing the environment feedback available, and depends on the problem at hand. This reward function directly affects the immediate decisions providing feedback to the controller or policy function. For a deterministic problem setup, the value function is defined as:

$$V(s_0) = R(s_0) + \gamma_d R(s_1) + \gamma_d^2 R(s_2) + \gamma_d^3 R(s_3) + ... \tag{2.42}$$

Here, V is the value function, $R(s_t)$ is the immediate reward received from entering state s_t, γ_d is a discount factor (not to be confused with γ, the flight path angle) that controls how much to discount (expected) future rewards. The value of γ_d should lie between $0 \sim 1$, depending on the problem setting. A higher value of γ_d represents higher weighting on future rewards, and vice-versa. The Bellman equation readily provides an efficient way to solve the above problem [138]. It aggregates the long term reward function and separates it elegantly into two terms as shown in Equation 2.43.

$$V(s_t) = R(s_t) + \gamma_d V(s_{t+1}) \tag{2.43}$$

Therefore, according to Bellman equation, the long term reward consists of immediate reward and sum of future discounted rewards. The next state s_{t+1} is achieved by following a policy, generally denoted by π if it is stochastic and μ if it is deterministic. In this research, since the aim is to develop a deterministic policy or flight controller, the policy notation used will be μ. Equation 2.43 can be efficiently used to solve for an optimal value function given that the problem or the dynamical system exhibits only a finite number of states. If the number of states $|S|$ is finite, then there can be $|S|$ equations written down that can in turn be solved iteratively to estimate the optimal value function ($V^*(s) = \max_\mu V(s)$). Thus the optimal policy can be calculated using $\mu^*(s) = \arg\max_a V^*(s')$, where s' is the next state of the system and a is a particular control action. This process is generally known as "value iteration" and works well for learning policies that are optimal for dynamical systems when there are a finite number of actions. Value iteration serves as a core part of the Q-learning algorithm [138], which basically solves the Bellman equation iteratively, Equation 2.43.

Particularly, it is often desired to estimate the action-value function, denoted by $Q(s,a)$. For a controlled dynamical system, the evolution of states is significantly dictated by the action input to the dynamical system. And, hence it is generally required to know the value of not only being in a state, but also what action to take, given a particular state, so as to maximize the value.

2.9.2 Deep Q-Network

In practice, many dynamic systems consist of a large number of continuous control actions, as well as states ($\sim\infty$), and hence the value iteration algorithm cannot be applied to such a system, for example an aircraft. To handle the large number of observed states, deep neural networks can be employed to estimate a nonlinear action-value function which learns an embedded state representation that can be easily compared to other, even unobserved states resulting in good generalization capabilities. Deep Q-network (DQN) [100] is one of the algorithms used to solve this problem. Traditionally, nonlinear function estimators such as deep neural networks have been avoided to approximate value functions. This is due to the fact that the training becomes highly unstable because of a large number of internal parameters and highly nonlinear behavior of neural networks. The core of this problem arises during training due to high correlations among sequential data and the dependence of target value on the original value function, see Equation 2.43. Reference [100] addresses these issues by introducing two important innovations. Firstly, a collection of experiential data is accumulated and then the agent is trained from samples randomly chosen in batches; thereby removing auto-correlations among data samples. Secondly, a target neural network is used while applying Bellman's iterative update (see Equation 2.43) to estimate the numerical "value", instead of the original value function neural network. This target neural network is referred to as a delayed temporal difference backup of the original value function neural network, whose parameters are generated using exponential moving averages of the original network's parameters. The second innovation imparts stability during training by providing consistent target values.

However, DQN can only handle a finite number of discrete actions. The continuous action space can be discretized to employ the DQN algorithm, however, the problem of the "curse of di-

mensionality" arises in such a setting. For an aircraft such as Skyhunter, with four control variables (throttle, elevator, aileron, and rudder), even with a practically viable coarse discretization around a trim point, the total action values would amount to ~ 1.45 million, see Equations 2.44.

$$\delta_{t_p} = -20:2:20(\%) = 21 values \tag{2.44a}$$

$$\delta_{e_p} = -2:0.1:2(^\circ) = 41 values \tag{2.44b}$$

$$\delta_{a_p} = -2:0.1:2(^\circ) = 41 values \tag{2.44c}$$

$$\delta_{r_p} = -2:0.1:2(^\circ) = 41 values \tag{2.44d}$$

Therefore, it becomes impractical to explore all these actions using iterative algorithms, and the problem only exacerbates when ranges of control surfaces are widened to cover routine aircraft maneuvers. The innovation of utilizing deep neural networks offers a potential solution for addressing continuous policy decisions. In conjunction with the Deep Deterministic Policy Gradients (DDPG) algorithm [88] deep neural networks can be trained to behave as policy functions, that directly map states to actions. With this solution of using neural networks as an optimal policy, the major challenge that arises is that of training, or that of correct input-output data generation. When training the deep Q-network, the output data or target values for this network are estimated using Bellman's equation, however, there is not an iterative procedure that directly estimates the target output values for the policy neural network. This problem is addressed using policy gradient algorithms [138], specifically deterministic policy gradients [130] which directly provides the gradients of the neural network policy parameters without the need for output target values.

2.9.3 Policy Gradients

The goal of a reinforcement learning problem is to maximize a long-term reward or an objective function. This objective function can be represented as in Equation 2.45a. Here, $Q(s,a)$ is the action-value function and it is the expectation of the future discounted rewards. If a deterministic policy $\mu : S \leftarrow A$ is followed the action-value function is as follows:

$$J = Q(s, a) \tag{2.45a}$$

$$Q^{\mu}(s_t, a_t) = E[r(s_t, a_t) + \gamma_d Q^{\mu}(s_{t+1}, \mu(s_{t+1}))] \tag{2.45b}$$

Equation 2.45b is the recursive relationship or the Bellman equation that is used to estimate the expected cumulative reward, also known as the action-value function. Sampling a batch of "N" data points, the target values can be set as follows:

$$y_t = r(s_t, a_t) + \gamma_d Q(s_{t+1}, \mu(s_{t+1})|\theta^Q) \tag{2.46}$$

Here, y_t are the estimated target values and note that the Q function is parameterized by θ^Q. The parameters of Q function are updated (via training) using the following loss function (L) that minimizes the difference between the target values and the values predicted by the Q approximation:

$$L = E[(Q(s_t, a_t) - y_t)^2] \tag{2.47}$$

Generally, once the Q function is updated, the optimal policy is computed using a Q-learning greedy policy update, where the single most likely action is chosen by the agent, which is equivalent to:

$$\mu(s) = \arg\max_a Q(s, a) \tag{2.48}$$

However, as described in Section 2.9, solving Equation 2.48 becomes impractical due to large and continuous action space, even if a feasible coarse discretization is used. To address this issue, deterministic policy gradients (DPG) [130] are utilized. Instead of solving Equation 2.48, the deterministic policy can be updated simply in the direction of the Q function gradient which reduces the total number of actions and states that must be considered, as only the general direction of the gradient must be calculated. If the policy μ is parameterized by θ^{μ}, then these parameters can be updated in proportion to the gradients of the action-value function, that is, $\nabla_{\theta} Q^{\mu}(s, \mu(s))$. Therefore, the policy parameters can be updated as follows:

$$\theta_{k+1}^{\mu} = \theta_k^{\mu} + \alpha_{lr} E[\nabla_\theta Q^{\mu}(s, \mu(s))] \quad (2.49)$$

Using chain rule, the term inside the expectation on right hand side of Equation 2.49 can be further decomposed into two terms: (1) Gradient of policy with respect to its parameters, and (2) Gradient of action-value function with respect to actions, as shown in Equation 2.50. The proofs can be found in Reference [130].

$$\nabla_\theta Q^{\mu}(s, \mu(s)) = \nabla_a Q^{\mu}(s,a) \nabla_\theta \mu(s) \quad (2.50)$$

Therefore, using Equations 2.45a and 2.50, the gradient of the objective function can be derived as follows:

$$\nabla J = \nabla_a Q^{\mu}(s,a) \nabla_\theta \mu(s) \quad (2.51)$$

This gradient of objective function can be utilized to update the policy neural network parameters.

2.9.4 Deep Deterministic Policy Gradients

The DDPG algorithm starts with initializing four ANNs: actor (μ), target actor (μ'), critic (Q) and target critic (Q'). Let θ denote the parameters of a neural network (weights and biases), then μ is parametrized by θ^{μ} and so on. Since the critic network depends on both state (s) and action (a), critic network notation is: $Q(s,a|\theta^{Q})$, and actor notation is: $\mu(s|\theta^{\mu})$. Initially, the parameters of the target neural networks (target actor and target critic) are set to be the same as their respective actor and critic networks, i.e., $\theta^{Q'} \leftarrow \theta^{Q}$ and $\theta^{\mu'} \leftarrow \theta^{\mu}$. The copies of the original neural networks (target networks) provide additional stability and increase the probability that the ANN models converge as discussed in Reference [100].

After initialization, the iterative learning procedure is started. At the start of each iteration, the noise process is initialized $\mathscr{N}$, and the dynamic system is reset to an initial state. The noise ($\mathscr{N}$) is a temporally correlated Gauss-Markov process, which is generally used for control system

problems that have inertia, and is called the Ornstein-Uhlenbeck process as discussed in Reference [88].

An episode proceeds with first obtaining an action a_t from the neural network policy $\mu(s_t|\theta^{\mu})$ and noise is added to this action to incentivize exploration. Exploration is a methodology used in reinforcement learning algorithms that explores new actions given a state, that could lead towards improving the current policy and potentially converge to an optimal policy. In reinforcement learning problems, there is always a dilemma between exploration and exploitation. Exploitation simply means the use of those actions (or current policy) that are known to produce good corresponding next states (or maximize value), given a current state of the system.

This noisy control action is then applied to the dynamic system to get the next state (s_{t+1}), a flag (done: d) is set if this next state is terminal, an immediate reward is received (r_t), and the data set (state, action, reward, next state, done) are saved in a data stream called the experience or replay buffer (*RB*). After collecting a series of initial data points, a random mini-batch of N experiences is selected and an estimate of state-action values ($y_t^{(i)}$) are computed using Equation 2.43, the respective immediate reward ($r_t^{(i)}$), the target critic (Q') and the target actor (μ'). Note here, that the target critic (Q') and target actor (μ') are used for calculating the next state-action value (next state = $s_{t+1}^{(i)}$, next action = $a_{t+1}^{(i)} = \mu'(s_{t+1}^{(i)}|\theta^{\mu'})$), that is required as the second expression in the Bellman's equation.

These state-action values (y_t) serve as the "true" or target values to compute the loss for training the critic ANN. Mean squared error loss is computed, between the target values (y_t) and the predicted output state-action values by the critic neural network (Q). This loss is utilized to compute gradients of the critic parameters (θ^{Q}) and the Adam gradient descent algorithm is used to update the critic parameters [80].

One of the major steps of the DDPG algorithm involves training the actor ANN using policy gradient method as outlined in Reference [130], also see Section 2.9.3. Gradients of the critic ANN with respect to actions ($\nabla_a Q$) and that of actor with respect to its own parameters ($\nabla_{\theta^{\mu}} \mu$), are utilized to find the gradients of the objective function for the actor ANN, which are in turn

used to update actor ANN parameters. Gradient ascent is used to update the actor neural network together with the Adam learning algorithm.

Algorithm 6 DDPG Algorithm

1: Initialize ANNs: $Q(s,a|\theta^{Q})$ and $\mu(s|\theta^{\mu})$
2: Initialize target ANNs: $Q'(s,a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$
3: Initialize replay buffer: RB
4: $\theta^{Q'} \leftarrow \theta^{Q}$
5: $\theta^{\mu'} \leftarrow \theta^{\mu}$
6: **for** E = 1, K **do**
7: Initialize $\mathcal{N}$
8: Initialize LTI state s_1
9: **for** t = 1, T **do**
10: $d = 0$
11: $a_t = \mu(s_t|\theta^{\mu}) + \mathcal{N}_t$
12: $s_{t+1} = LTI(s_t, a_t)$
13: **if** s_{t+1} is terminal **then**
14: $d = 1$
15: **end if**
16: $r_t = R(s_t, a_t)$
17: $RB \equiv RB \cup \{s_t, a_t, r_t, s_{t+1}\}$
18: Sample a mini-batch N transitions from RB
19: **for** i = 1, N **do**
20: $y_t^{(i)} = r_t^{(i)} + \gamma_d Q'(s_{t+1}^{(i)}, \mu'(s_{t+1}^{(i)}|\theta^{\mu'})|\theta^{Q'})$
21: **end for**
22: $L = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - Q(s^{(i)}, a^{(i)}|\theta^{Q}))^2$
23: Minimize L, and update $Q(s,a|\theta^{Q})$
24: $\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_a Q(s^{(i)}, \mu(s^{(i)})|\theta^{Q}) \nabla_{\theta^{\mu}} \mu(s^{(i)}|\theta^{\mu})$
25: Update $\mu(s|\theta^{\mu})$ using $\nabla_{\theta^{\mu}} J$
26: $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$
27: $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
28: **end for**
29: **end for**

One of the most important and final steps of the DDPG algorithm is to perform soft updates to the target actor (μ') and target critic (Q') neural networks. This update method provides smoothed parameters and takes temporal difference backup of the original actor and critic neural networks. This provides stability in the overall training procedure, see Equation 2.52.

$$actor_{target} = \tau(actor) + (1-\tau)(actor_{target}) \tag{2.52}$$

The step-by-step process of DDPG is outlined in Algorithm 6.

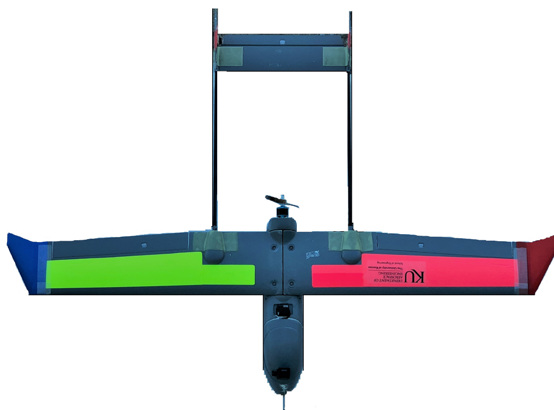
As shown in Algorithm 6, in line 12, the dynamic system is a linear time invariant (LTI) model of the aircraft that is discretized to produce next step states based on input controls and initial state, see Section 3.3. This mathematical model, see Section 3.1 is obtained for a specific flight condition around a trim point and hence predicts perturbed states while accepting perturbed controls. As shown in line 8, the initial state vector s_1 is randomly chosen from a range of "perturbed" states before the start of each episode.

Chapter 3

Aircraft Platform, Hardware and Software Development

3.1 Skyhunter Aircraft

The platform chosen for this research is a twin-boom, foam based structure aircraft known as a Skyhunter UAS. The Skyhunter features a 71 inch wingspan and a fuselage length of 25 inches. The operating weight of this aircraft with all avionics and components is approximately 8.4 pounds. This aircraft is propelled by a brushless DC (direct current) motor and is capable of sustaining a cruise speed of 50.63 ft/s. This UAS features three powered control surfaces, two differential ailerons and one elevator, which come pre-installed on the standard commercially off-the-shelf (COTS) aircraft.



(a) Top View Picture on the Ground

(b) Bottom View Picture in Flight

Figure 3.1: Skyhunter UAS Picture

A picture of one of the Skyhunters used can be seen in Figure 3.1, courtesy of the Department of Aerospace Engineering, The University of Kansas. The aircraft foam structure is modified

by reinforcing it with fiber glass laminates at several different important locations including the bottom surface of the fuselage, the aileron and elevator joints, and the horizontal and the vertical stabilizers. The COTS Skyhunter does not have landing gears and rudder control surfaces, therefore these are assembled in-house and installed on the aircraft.

To determine the aircraft's moments of inertia, a bifilar pendulum test is conducted on the full aircraft with all avionics onboard (see [70]). The resulting moment of inertia estimates are shown in Table 3.1. The I_{xz} component of the aircraft was assumed to be negligible due to a relatively small magnitude.

Moment of inertia	slug-ft^2
I_{xx}	0.1
I_{yy}	0.1
I_{zz}	0.35

Table 3.1: Moment of Inertia Estimates from Bifilar Pendulum Tests

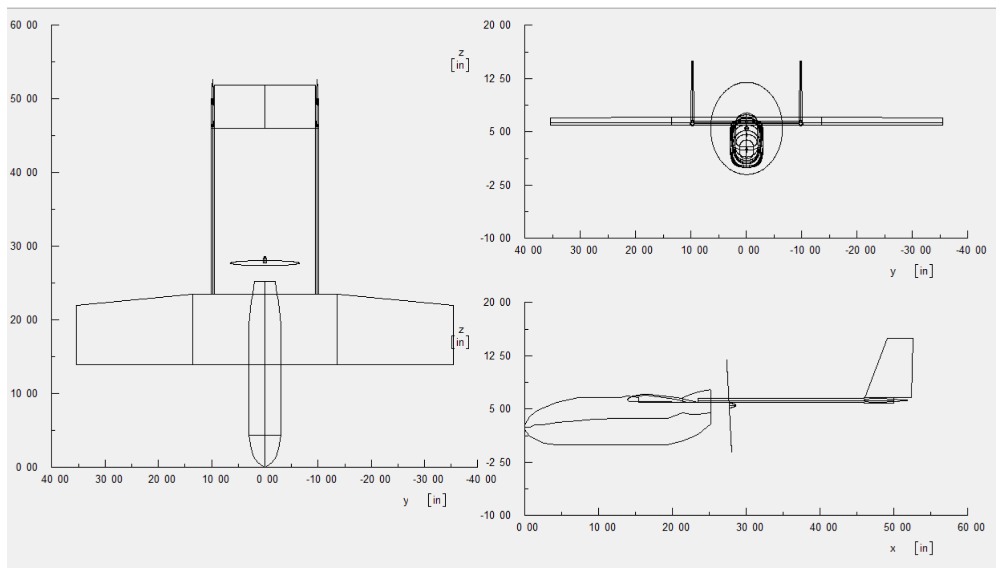


Figure 3.2: AAA Model: Top, Front, and Side Views

To characterize the motor dynamics of the system, engine testing is conducted at the Mal Harned Propulsion Lab at the Garrison Flight Research Center in Lawrence, Kansas. This provides

the static thrust values for the engine across a RPM (Revolutions Per Minute) range, as well as correlates the static thrust with the input pulse width modulation (PWM) signal for controller design. These static thrust values are then used to estimate the in-flight thrust values at the cruising velocity of 50.63 ft/s using methods proposed in Reference [77] that convert thrust values for a similar propeller across varying advance ratios [25]. Using the moment of inertia estimates and the motor dynamics, an aircraft modeling approach is taken to obtain the values of aircraft stability and control derivatives. The software used is Advanced Aircraft Analysis (AAA), a product of DARcorporation, which uses aircraft geometry and flight conditions to construct a dynamic model. The stability and control derivatives are estimated based on both the explicit equations and by correlating variables using historical data across a vast aircraft database. The AAA geometric model is shown in Figure 3.2. Flight tests are conducted and the dynamic model is fine-tuned by making slight adjustments to stability and control derivatives until the recorded flight data and the physics-based model converge across large portions of the flight. The notation for these stability and control derivatives are developed and outlined in Reference [117]. The aircraft geometry data is shown in Table 3.2. The dynamic model, stability and control derivatives are developed with the help of the Flight Research Laboratory's team member Mr. Hady Benyamen at The University of Kansas. His contribution towards developing a practical model is greatly appreciated.

Planform Area: S (ft^2)	MGC: $\bar{C}$ (inches)	Span:b (ft)	Aspect Ratio ($AR = b^2/S$)
4.82	8.78	5.92	9.81

Table 3.2: Aircraft Main Wing Geometric Parameters

3.2 Skyhunter 6-DOF Equations

The aircraft equations of motion (see Section 2.2) are integrated in time using the classical Runge–Kutta fourth order method [82]. In order to compute the derivatives in Equations 2.1, first the lift and drag coefficients, and the aerodynamic forces and moments coefficients need to be computed. The lift and drag coefficients used for the calculation of non-dimensional aerodynamic force coefficients,

are shown in Equations 3.1.

$$C_L = C_{L_1} + C_{L_\alpha}(\alpha - \alpha_{trim}) + \frac{C_{L_q}(Q - Q_{trim})\bar{c}}{2u_{trim}} + \frac{C_{L_{\dot{\alpha}}}\dot{\alpha}\bar{c}}{2u_{trim}} + \frac{C_{L_u}(u - u_{trim})}{u_{trim}} + C_{L_{\delta_e}}(\delta_{e_x} - \delta_{e_{trim}}) \tag{3.1a}$$

$$C_D = \overline{C_{D_0}} + \frac{C_L^2}{\pi(AR)0.88} \tag{3.1b}$$

The non-dimensional aerodynamic moment coefficients are shown in Equations 3.2. These are calculated in the aircraft stability axis of motion.

$$C_{l_s} = C_{l_\beta}\beta + \frac{C_{l_p}(P - P_{trim})b}{2u_{trim}} + \frac{C_{l_r}(R - R_{trim})b}{2u_{trim}} + C_{l_{\delta_a}}\delta_{a_x} + C_{l_{\delta_r}}\delta_{r_x} \tag{3.2a}$$

$$C_{m_s} = C_{m_1} + C_{m_\alpha}(\alpha - \alpha_{trim}) + \frac{C_{m_q}(Q - Q_{trim})\bar{c}}{2u_{trim}} + \frac{C_{m_{\dot{\alpha}}}\dot{\alpha}\bar{c}}{2u_{trim}} + C_{m_u}\frac{(u - u_{trim})}{u_{trim}} + C_{m_{\delta_e}}(\delta_{e_x} - \delta_{e_{trim}}) + 2C_{m_1}\frac{(u - u_{trim})}{u_{trim}} + (C_{m_{t_u}} + 2C_{m_{t_1}})\frac{(u - u_{trim})}{u_{trim}} + C_{m_{t_\alpha}}(\alpha - \alpha_{trim}) \tag{3.2b}$$

$$C_{n_s} = C_{n_\beta}\beta + C_{n_p}\frac{(P - P_{trim})b}{2u_{trim}} + C_{n_r}\frac{(R - R_{trim})b}{2u_{trim}} + C_{n_{\delta_a}}\delta_{a_x} + C_{n_{\delta_r}}\delta_{r_x} \tag{3.2c}$$

The non-dimensional aerodynamic force coefficients are shown in Equations 3.3.

$$C_{x_a} = C_L \sin\alpha - C_D \cos\alpha \tag{3.3a}$$

$$C_{y_a} = C_{y_\beta}\beta + \frac{C_{y_p}(P - P_{trim})b}{2u_{trim}} + \frac{C_{y_r}(R - R_{trim})b}{2u_{trim}} + C_{y_{\delta_a}}\delta_{a_x} + C_{y_{\delta_r}}\delta_{r_x} \tag{3.3b}$$

$$C_{z_a} = -C_L \cos\alpha - C_D \sin\alpha \tag{3.3c}$$

The non-dimensional aerodynamic moment coefficients in the aircraft body frame are shown in Equations 3.4

$$C_l = C_{l_s}\cos\alpha - C_{n_s}\sin\alpha \tag{3.4a}$$

$$C_m = C_{m_s} \tag{3.4b}$$

$$C_n = C_{l_s}\sin\alpha + C_{n_s}\cos\alpha \tag{3.4c}$$

The aerodynamic force calculations are shown in Equations 3.5, where $\bar{q}$ is the dynamic pressure.

$$X_A = \bar{q}SC_{x_a} \tag{3.5a}$$

$$X_T = x_{t_0} + x_{t_1}(100\delta_{t_x}) + x_{t2}(100\delta_{t_x})^2 \tag{3.5b}$$

$$Y_A = \bar{q}SC_{y_a} \tag{3.5c}$$

$$Z_A = \bar{q}SC_{z_a} \tag{3.5d}$$

Aerodynamic moments can be computed as shown in Equations 3.6, where dT is the moment arm between the aircraft center of gravity and the motor (thrust) location.

$$L = C_l\bar{q}Sb \tag{3.6a}$$

$$M = C_m\bar{q}S\bar{c} - X_T dT \tag{3.6b}$$

$$N = C_n\bar{q}Sb \tag{3.6c}$$

Using the above equations and the equations of motion defined in Section 2.2, the 6-DOF simulator is implemented by propagating the states using RK4 integrator and control inputs.

3.3 Skyhunter LTI Model

The LTI model is developed by linearizing the equations of motion around a trim point for steady-state wings level condition. For longitudinal dynamics the trim point is, velocity: 50.63 ft/s, angle of attack: 0.73 degrees, pitch attitude: 0.73 degrees, throttle trim: 63.2% and elevator trim: 1.5 degrees. For lateral dynamics the trim point is zeros for all states and controls. The longitudinal LTI model of the Skyhunter aircraft is represented in Equation 3.7. The longitudinal state vector is defined as: $\mathbf{x_{lon}} = [V_T \;\; \alpha \;\; \theta \;\; Q]^T$ and the control vector is defined as: $\mathbf{u_{lon}} = [\delta_t \;\; \delta_e]^T$.

$$\begin{bmatrix} \dot{V}_T \\ \dot{\alpha} \\ \dot{\theta} \\ \dot{Q} \end{bmatrix} = \begin{bmatrix} -0.1240 & 19.0662 & -32.1974 & 0 \\ -0.0250 & -6.2646 & -0.0080 & 0.9405 \\ 0 & 0 & 0 & 1 \\ 0.0224 & -14.6920 & 0.0051 & -2.7085 \end{bmatrix} \begin{bmatrix} u \\ \alpha \\ \theta \\ q \end{bmatrix} + \begin{bmatrix} 5.9203 & -0.7755 \\ -0.0544 & -0.3158 \\ 0 & 0 \\ 0.2737 & -19.4782 \end{bmatrix} \begin{bmatrix} \delta_t \\ \delta_e \end{bmatrix} \tag{3.7}$$

The lateral LTI model is shown in Equation 3.8. The lateral state vector is defined as: $\mathbf{x_{lat}} = [\beta \;\; \phi \;\; P \;\; R]^T$ and the control vector is defined as: $\mathbf{u_{lat}} = [\delta_a \;\; \delta_r]^T$. Both the LTI model Equations 3.7 and 3.8 are shown in the form: $\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$.

$$\begin{bmatrix} \dot{\beta} \\ \dot{\phi} \\ \dot{p} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} -0.6048 & 0.6359 & -0.0153 & -0.9797 \\ 0 & 0 & 1.0000 & 0 \\ -35.6302 & 0 & -8.2173 & 2.4916 \\ 34.3722 & 0 & -1.3352 & -2.1524 \end{bmatrix} \begin{bmatrix} \beta \\ \phi \\ p \\ r \end{bmatrix} + \begin{bmatrix} 0 & 0.2484 \\ 0 & 0 \\ 74.0768 & 3.8109 \\ 0.2435 & -26.3714 \end{bmatrix} \begin{bmatrix} \delta_a \\ \delta_r \end{bmatrix} \tag{3.8}$$

3.4 Monte-Carlo Simulations: Off-Nominal Initial Condition Flights

One of the main tools and practices for validation and verification of developed guidance, navigation and control algorithms is Monte-Carlo analysis [19]. Software in the loop simulations with complete guidance, navigation and control algorithms combined with aircraft six-degrees-of-freedom equations of motion are conducted to test the reliability and robustness of the autopilot systems. These end-to-end experiments are performed to find any underlying flaws in the existing algorithms such as path planning, guidance or control. As discussed in Reference [19], Monte-Carlo analysis is a statistical tool that can help predict the behavior of an autopilot system prior to a flight test.

A Monte-Carlo simulation setup is similar to Figure 2.2, except an outer loop generates different initial conditions for the aircraft states and repeats the flights. The initial conditions for 12

Table 3.3: Ranges and Intervals for Aircraft States Used to Generate Initial Conditions for Monte-Carlo Flight Simulations

State	Minimum	Maximum	Interval	Number of Values
$V_{T_0}(ft/s)$	45	55	1	11
$\alpha_0(^\circ)$	-5	5	1	11
$\beta_0(^\circ)$	-5	5	1	11
$\phi_0(^\circ)$	-30	30	2	31
$\theta_0(^\circ)$	-10	20	2	16
$\psi_0(^\circ)$	-30	30	3	13
$P_0(^\circ/s)$	-50	50	2	51
$Q_0(^\circ/s)$	-30	30	2	31
$R_0(^\circ/s)$	-20	20	2	21
$N_0(ft)$	n_0 - 100	n_0 + 100	10	21
$E_0(ft)$	e_0 - 100	e_0 + 100	10	21
$H_0(ft)$	h_0 - 50	h_0 + 50	5	21

aircraft variables are varied, namely: total velocity, angle of attack, side-slip angle, roll, pitch and yaw angles, roll, pitch and yaw rates, and the inertial north, east and height. For each of these 12 variables, a vector consisting of a set of values is defined. The ranges and the resolution of these set values are based on flight test data. These ranges are shown in Table 3.3.

From each of the aircraft states' vector, a value is sampled uniformly at random without replacement and combined to create an initial aircraft state condition. Using this state, a randomly chosen combination of velocity, attitude angles, aircraft location with respect to the waypoint path, etc., a flight simulation is carried out. After, each simulated flight test, a monitoring algorithm checks if the flight test was a "success" or a "failure" case. Each simulated flight test result is assigned a boolean flag and saved in its respective success or failure directory, to analyze the data further.

The success or failure algorithm is mainly based on boundedness of all the aircraft states, the control rates and the shortest distance of aircraft from the waypoint path. Therefore, in essence the Monte-Carlo simulation checks the bounded input-output (BIBO) stability of the complete closed-loop system with the GNC algorithms. It also checks the reliability of the guidance algorithm, its behavior to off-nominal flight conditions and its convergence to waypoint flight path. The test of control rate boundedness provides information about the stability of the control system to some

Table 3.4: Ranges for Aircraft States, Controls and Control Rates Used to Test a Success or Failure Case for a Monte-Carlo Flight Simulation

State	Minimum	Maximum
$V_T(ft/s)$	40	60
$\alpha(^\circ)$	-10	10
$\beta(^\circ)$	-10	10
$\phi(^\circ)$	-45	45
$\theta(^\circ)$	-20	20
$P(^\circ)$	-40	40
$Q(^\circ)$	-20	20
$R(^\circ)$	-15	15
$\delta_t(\%)$	10	100
$\delta_e(^\circ)$	-10	10
$\delta_a(^\circ)$	-10	10
$\delta_r(^\circ)$	-10	10
$\delta_t/dt(\%/s)$	-30	30
$\delta_e/dt(^\circ/s)$	-10	10
$\delta_a/dt(^\circ/s)$	-10	10
$\delta_r/dt(^\circ/s)$	-10	10

extent and statistically signifies the validity of the controller. The bound conditions representing a successful simulation flight are shown in Table 3.4.

3.5 Aircraft Avionics

The advances in compact and computationally efficient computer systems, which are ideal for small unmanned aerial systems, has made a significant impact in the aerospace and robotics community. Computational platforms, such as NVIDIA Jetson-Nano, comprise the characteristics that are ideal for implementing autopilot software onboard a UAS. It consists of a powerful Quad-core Arm based A57 chip processor with 1.43 GHz clock speed. Combined with an efficient 128-core graphical processing unit (GPU), the computer has the potential for running vastly parallel systems, such as neural networks, with a potential for real-time learning. Another economic and computationally competitive computer unit is the Odroid-XU4. With heterogeneous processors composed of two quad-core processors, clock speeds of 2 GHz and 1.4 GHz respectively, Odroid-XU4 offers a computationally efficient platform to run real-time demanding software sys-

tems. Both these computer units (NVIDIA Jetson-Nano and Odroid-XU4) are fully equipped to run real-time processes at high update rates (tested at 20 Hz up-to 50 Hz), comprising of demanding autopilot software: guidance, navigation and control together with sensor peripheral devices, all in parallel with real-time data exchange and execution of high rate control on a UAS. However, the NVIDIA Jetson-Nano computer provides an edge over Odroid-XU4 due to its 128-core CUDA (Compute Unified Device Architecture) enabled GPU, providing parallel computing capabilities for computationally intensive machine learning applications.

For the aircraft platform used in this research (Skyhunter UAS), two different avionics systems are developed, fabricated and flight tested. The details of both avionics with a complete list of peripherals, interfacing, connections, data acquisition devices and sensor integration are given in the following paragraphs. But, here it is important to note the following major differences that contrast the two avionics systems:

- One of the avionics systems utilizes the Odroid-XU4 computer. Another major component that is a distinctive feature of this avionics, is the Microhard Nano - n920 telemetry module.
- The second avionics system consists of the NVIDIA Jetson-Nano as the main computing platform. It is interfaced with Microhard Pico 900 (P900) telemetry module.

The readily available accurate sensor fusion platforms with multiple redundancies, such as Pixhawk Flight Control Unit (FCU), have surged during the drone technological revolution within the last decade. Autopilot hardware devices such as Pixhawk 1 and Pixhawk 2.1 cube, readily provide reliable and economic platforms for sensor data collection with redundancies, control output processing and classical guidance and control algorithms. These boards can act as data acquisition and input-output control devices in conjunction with computationally powerful boards (NVIDIA Jetson-Nano and Odroid-XU4) that behave as companion computers. These companion computers can run computationally demanding algorithms such as, deep neural networks, robust and adaptive controllers, path planning and high-level decision making processes.

A complete, general avionics block diagram is shown in Figure 3.3. The main components

of the avionics are the companion computer, Pixhawk version 2.1 FCU, a GPS, pressure sensor with a Pitot tube and an Electronic Speed Controller (ESC). The Pixhawk board acts as the sensor interface or the data acquisition board for the whole system and it receives external control inputs from the companion computer via a Serial-Mavlink interface. Pixhawk provides IMU data, interfaces with GPS, processes I2C signal from the pressure sensor, processes PPM (Pulse Position Modulation) input signal from a remote control (RC) receiver (via PPM encoder) and outputs PWM signals to control servo motors (Elevator, Aileron and Rudder) and also to ESC to control an electric engine (forward thrust). The companion board acts as the main onboard computer on the aircraft. It runs a Linux based operating system (OS) Ubuntu, and a middleware meta OS called: Robot Operating System (ROS) [114, 133]. The ROS framework runs all the guidance, navigation and control sub-systems, sensor data acquisition process and the wireless communication node. For wireless communication between the aircraft and the ground control station (GCS) [3] a Microhard telemetry module is used that operates around 900 MHz frequency.

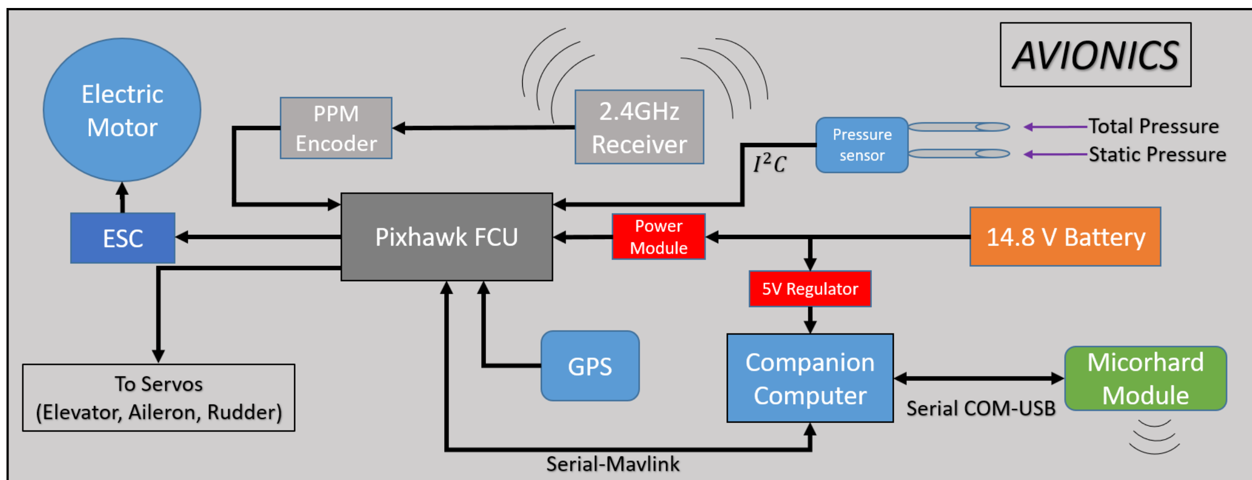


Figure 3.3: Skyhunter Avionics Setup

The Pixhawk units can run two different types of firmware or software operating systems with autopilot functionalities: (1) Ardupilot, and (2) PX4. Each software sends out data over Mavlink software protocol, which consists of all sensor variables (airspeed, GPS, IMU, RC and Flight mode data) and controller values can also be received over this protocol. The two-way data transmission

has different properties such as update rates, for the two distinct firmware. This data can be harnessed by a companion computer over USB-COM or a serial interface cable and parsed by an open source ROS package called MAVROS. The MAVROS package gives access to published data topics for the sensor variables and advertising capabilities for the controller values. Ardupilot firmware uses an RC override functionality through MAVROS to change the flight mode and have the Pixhawk FCU accept controller values from the companion computer. On the other hand, PX4 firmware simply uses an "offboard" flight mode function to pass on controller values from the companion computer to the Pixhawk.

3.6 Onboard ROS-Autopilot Software Architecture

All the autopilot sub-systems are implemented as custom designed ROS processes, also called ROS nodes (executables), that interact with each other using the ROS master server. ROS provides a set of software tools and libraries, or drivers for various peripherals and sensors, that are open-source and can be readily used for real-time robotic applications. The ROS library provides state-of-the-art algorithms for sensor data processing, vision systems, robot navigation and path planning, motor control, etc.

The main advantage of using ROS framework is its flexibility to run on different platforms (software portability), such as NVIDIA Jetson-Nano, Odroid-XU4 or a desktop computer with standard CPU architecture (such as x86 or AMD) and a Linux operating system (Ubuntu), which readily supports development, testing and implementation on different sized UAS platforms. The major components or processes of this ROS-Autopilot software are: (1) 6-DOF node, (2) Pixhawk node, and (3) autopilot node, as shown in blue circles in Figure 3.4. The 6-DOF node computes and outputs the aircraft states for the next time step. It integrates the aircraft differential equations of motion, based on an initial state value and control excitation. The servo dynamics are modeled as a part of these differential equations, thereby the servo time delays affect the control surface deflections and hence the corresponding states in this step, as described in Section 2.2. The Pixhawk node has a similar output functionality, except it does not integrate the equations of mo-

tion, rather it just takes in measured, filtered and estimated sensor values from the Pixhawk FCU through Serial-Mavlink interface, parsed via MAVROS. The aircraft states' output ROS topic is of the same data type for both the 6-DOF and the Pixhawk nodes. Based on the software mode: HiTL or Flight, this "states" topic is populated, represented in a green box in Figure 3.4.

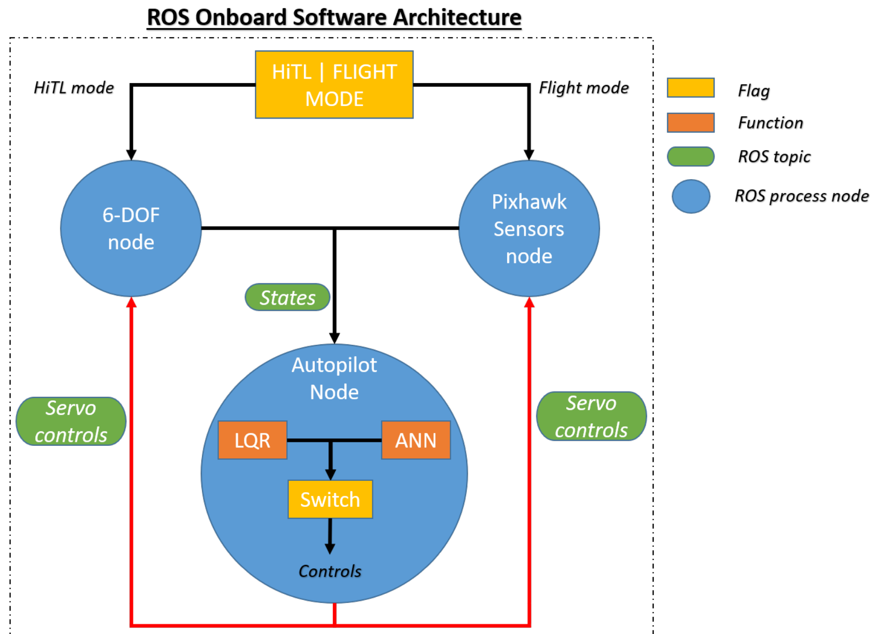


Figure 3.4: ROS Onboard Software Architecture: HiTL and Flight Modes

This states topic then serves as the input to the Autopilot node. The Autopilot node runs all the guidance and control functions of the system, with additional functionalities including GPS data conversions and path follower (waypoint line switching). As shown in Figure 3.4, the LQR and ANN (orange boxes) control values are computed in parallel and are analyzed by the safety-critical switch (yellow box). The switch chooses the appropriate stable control outputs and sends them through servo controls topics to the 6-DOF and Pixhawk nodes, depending on the HiTL or flight test mode flag. It is important to note that during HiTL mode, the topic is subscribed by both the 6-DOF and the Pixhawk nodes in parallel. This allows for an end-to-end systems test for the whole avionics, software and hardware, in which case the control surfaces are deflected based on the control output values via the servo controls topic. In the flight test mode, only the servo controls topic is sent to the Pixhawk node, and the 6-DOF node is completely disabled.

The Pixhawk and the 6-DOF nodes are coded in the C++ programming language [137]. On the other hand, the functions inside the autopilot node are written in Python programming language [112]. The autopilot node handles waypoint data parsing and their conversion to local coordinates; sensor data variable assignment and their passing to guidance and control functions; aggregation of all variables: state, control, and various flags for a time stamp synchronized logging of all flight data. The autopilot node also contains a functionality that allows the user to modify certain flags and variables within the autopilot node. This parameter modification functionality is invoked whenever there is a waypoint data upload from the ground control station. The parameter modification functionality allows a user to change various guidance parameters, stability augmentation system (SAS) gains, LQR's Longitudinal and Lateral gains, IMU filter parameters, ANN activation and deactivation flags, airspeed and altitude commands, trim values for all control variables, etc. Through this parameter modification functionality, the user can change these variables dynamically in a real flight while the aircraft is flying, in either RC (manual) or auto modes. Note that these modes (RC and auto) are different than the HiTL and Flight modes. The RC and auto modes are sub-modes of the Flight mode, and can be toggled in real flight, allowing a pilot to take off the aircraft in RC mode and turn the auto mode "ON" after reaching the approximate desired altitude and airspeed.

During the HiTL testing, the aircraft 6-DOF equations of motion are propagated through a ROS node. This ROS node consists of all the stability and control derivatives of the Skyhunter aircraft developed using the AAA software [21]. These derivatives are used to compute respective aerodynamic coefficients that in turn determine the aircraft forces and moments. The equations of motion are integrated using standard Runge-Kutta integrator after augmenting servo dynamics, and hence states are emulated at 20 Hz frequency. The GNC node subscribes to these states' topic and computes controller outputs which are sent back to the 6-DOF node. The whole process is implemented in a feedback manner, exchanging data via subscribing and advertising ROS topics. A ROS graph for the whole software implementation is shown in the right part of Figure 3.5.

During HiTL process run, as shown in Figure 3.5, the controller outputs are sent to a topic,

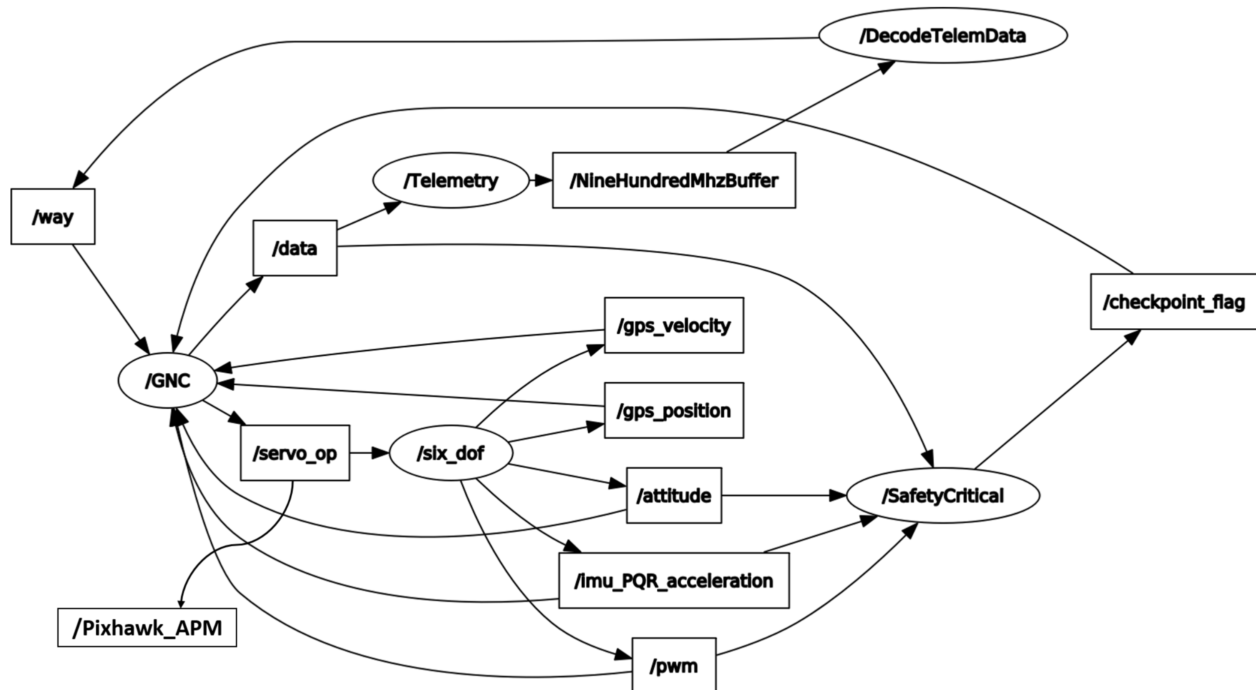


Figure 3.5: ROS Graph: Communication and Data Exchange Among the Autopilot Processes (nodes)

namely, $/servo_op$. The $/servo_op$ topic sends the control surface commands to the $Pixhawk_PX4$ node, which acts as an interface node between onboard controller and MAVROS package that in turn communicates to Pixhawk hardware directly via Serial-Mavlink interface. Hence, the control angles and throttle percentage, after conversion to PWM values are transferred to Pixhawk to be sent out to the control surface servos and the ESC. All the aircraft states and controller values are aggregated via the $/data$ topic and are sent to the Microhard telemetry module (via Mavlink protocol) to show real-time updates of aircraft location on the GCS. Therefore, HiTL validates the whole system in an end-to-end sense, by verifying the software, hardware, telemetry and data communication.

3.7 Evolution: Learning From Flight Data - TensorFlow Software

A separate software is developed using Python programming language and TensorFlow (tf) machine learning system [4], to handle custom learning through Deep Deterministic Policy Gradients

(DDPG) algorithm and real flight data. For details of the DDPG algorithm see Section 2.9.4. Since the goal here is to learn directly from flight data and re-train the actor neural network, several steps of the original DDPG algorithm are skipped. The actor neural network does not need to be trained from scratch, rather it has to be fine-tuned using real flight data, which represents relatively more accurate aircraft dynamics. Therefore, the learning rates selected for the evolution task are very small ($1e-10$).

The DDPG algorithm used for evolution does not need to run a time-series episode for data collection. Rather, the experience buffer is readily made available using the flight data. The episode concept changes to simply mini-batch training from the replay buffer data. For completeness, the DDPG steps used for evolution are shown in Algorithm 7.

Algorithm 7 DDPG Algorithm for Evolution

1: Copy pre-trained ANNs: $Q(s,a|\theta^Q)$ and $\mu(s|\theta^\mu)$
2: Copy pre-trained target ANNs: $Q'(s,a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$
3: Populate replay buffer with flight data: FD
4: **for** B = 1, K **do**
5:     Sample a mini-batch N transitions from FD
6:     **for** i = 1, N **do**
7:         $y_t^{(i)} = r_t^{(i)} + \gamma_d Q'(s_{t+1}^{(i)}, \mu'(s_{t+1}^{(i)}|\theta^{\mu'})|\theta^{Q'})$
8:     **end for**
9:     $L = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - Q(s^{(i)}, a^{(i)}|\theta^Q))^2$
10:     Minimize L, and update $Q(s,a|\theta^Q)$
11:     $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_a Q(s^{(i)}, \mu(s^{(i)})|\theta^Q)\nabla_{\theta^\mu}\mu(s^{(i)}|\theta^\mu)$
12:     Update $\mu(s|\theta^\mu)$ using $\nabla_{\theta^\mu} J$
13:     $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
14:     $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
15: **end for**

The neural networks' parameters are assigned using *tf.variables* and the values are copied from the pre-trained and flight tested neural networks. The target networks' parameters are assigned the "apply" operation from *tf.train.ExponentialMovingAverage* method to update these parameters using temporal difference backups.

The critic optimizer is defined in the TensorFlow operation graph using the Adam optimizer object: *tf.compat.v1.train.AdamOptimizer* and the parameter updates are carried out using the "min-

imize" method directly applied to cost (estimated vs target cumulative rewards). For the actor network, the training is done using policy gradients, therefore a cost cannot be directly minimized (or maximized). First, a gradient operation is defined using "tf.gradients" object that computes gradients of Q with respect to actions a, where Q is the graph variable output from the critic neural network and a is the input "tf.variable" to the critic network. This gradient operation is denoted by $\nabla_a Q(s^{(i)}, \mu(s^{(i)})|\theta^Q)$ in the Algorithm 7.

Another "tf.gradients" object is defined to compute the gradients of a or policy with respect to actor's parameters θ^μ, where a is the output "tf.variable" from the actor network. This gradient operation is denoted by $\nabla_{\theta^\mu} \mu(s^{(i)}|\theta^\mu)$. While defining this gradient operation, the third argument to the "tf.gradients" is passed containing the negative of the numerical values of the $\nabla_a Q(s^{(i)}, \mu(s^{(i)})|\theta^Q)$ operation. The gradients $\nabla_{\theta^\mu} \mu(s^{(i)}|\theta^\mu)$ are clipped before passing to the "apply_gradients" method of the *tf.compat.v1.train.AdamOptimizer* class that is used to update the actor parameters.

Chapter 4

Validation, Verification, and Discussions

4.1 Base Autopilot - LQR Controller

Implementation: The base autopilot software was implemented using MATLAB scripts, and was optimized by using matrix computations wherever possible. Due to this new implementation of base autopilot, the computational efficiency improved by at least 31 times, as compared to the SIMULINK based GNC simulator [76]. For 500 seconds of flight simulation using the SIMULINK based GNC system, the time taken is about 25 seconds; however the time taken for the same flight (500 seconds) simulation using the base autopilot takes only 0.8 seconds. The training time for the imitation neural networks also improves drastically if this base autopilot policy is used for data generation, as discussed in Sections 4.2.2 and 4.2.3.

LQR Stability Analysis: The decoupled longitudinal and lateral LQR controllers developed as described in Section 2.3.2 are tested for stability. The controller gain matrices are used to find the closed-loop system matrices for the linear time invariant dynamic model of the Skyhunter aircraft, as shown in Equation 4.1.

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} \tag{4.1a}$$

$$\mathbf{u} = -K\mathbf{x} \quad \implies \quad \dot{\mathbf{x}} = (A - BK)\mathbf{x} \tag{4.1b}$$

Here, $A - BK$ is called the closed-loop system matrix (A_{CL}) [83] and can be directly used to analyze the absolute stability of the controller with respect to the LTI system dynamics. Using the *damp* function in MATLAB, the poles, natural frequency, and damping ratio of the closed-loop

Table 4.1: Lateral Controller Closed-Loop System Properties

Poles	Damping (ζ)	Natural Frequency (rad/s)
-10.31	1	10.31
-2.05 $\pm$ 5.96i	0.325	6.30
-0.614	1	0.614

Table 4.2: Longitudinal Controller Closed-Loop System Properties

Poles	Damping (ζ)	Natural Frequency (rad/s)
-6.51 $\pm$ 3.07i	0.904	7.20
-0.418 $\pm$ 0.649i	0.542	0.772
-0.0362	1	0.0362
-0.368	1	0.368

system matrix are computed. The lateral closed-loop system's properties are summarized in Table 4.1. As it can be observed from the table, all the poles are negative, that is they are located in the left-half complex plane, and therefore the system exhibits stable behavior. The damping ratio for the second order Dutch Roll mode is 0.325, which increased from 0.174 for the open-loop system matrix.

The closed-loop system properties for the longitudinal controller are shown in Table 4.2. It can be seen that since all the poles are negative, the closed-loop system is stable. Also, the damping ratio for the second order Short Period mode increased from 0.806 (open-loop) to 0.915 for the closed-loop system, and that for the Phugoid mode, it changed from 0.155 to 0.643.

Monte-Carlo 6-DOF Simulations: The base autopilot (LQR controller) was subjected to Monte-Carlo flight test simulations with multiple random initial conditions (aircraft states) as described in Section 3.4. Different combinations of aircraft states consisting of 100,000 cases were chosen to test the statistical reliability of the complete GNC system. A standard race-track waypoint path consisting of four waypoints was set up for the flight simulations. Every test case is subjected to a "success" and "failure" monitoring algorithm, the results of which are shown in Table 4.3.

Out of a 100,000 flight test simulations, more than 99% of cases were successful, demonstrating that statistically the LQR-GNC system is very reliable and most likely will perform well in a

Table 4.3: Success and Failure Rates for Monte-Carlo Flight Test Simulations Conducted on the Base Autopilot

Total Cases	Success	Failure	Percentage Success	Failure Conditions
100,000	99,690	310	99.69%	$\delta_a/dt : Max = 15.89^\circ/s$

real flight test. Rigorous hardware in the loop tests were also conducted with the base autopilot software, checking its end-to-end system reliability.

Skyhunter Flight Tests: Real flight tests were conducted using the base autopilot, with matching guidance parameters and controller gain matrices as used for 100,000 Monte-Carlo flight simulations. It was found that the Skyhunter aircraft went into both lateral and longitudinal oscillations, as soon as the autopilot mode was engaged in-flight. As described in Section 2.3.2, the lateral controller gain matrix K_{lat} is a 2×4 matrix as shown in Table 4.4. The table shows the matrix elements that are associated with particular states and their respective control variable.

Table 4.4: Base Autopilot Lateral Controller Gain Matrix

	β	ϕ	P	R
δ_a	K_{11}	K_{12}	K_{13}	K_{14}
δ_r	K_{21}	K_{22}	K_{23}	K_{24}

In order to reduce the lateral oscillations and achieve the desired tracking performance, the elements of K_{lat} matrix were fine-tuned dynamically in-flight via the ground control station. The element K_{13}, which represents the gain from roll rate to aileron control, was reduced by about 86.69% of the original value. Also, the element K_{24}, which represents the gain from yaw rate to rudder control, was reduced by about 56.68% with respect to its original value. To further improve lateral tracking performance, the element K_{12}, which represents the gain from roll angle to aileron control, was increased by about 15.18%. The summary of these necessary changes for flight testing is shown in Table 4.5.

Similarly, the elements of the longitudinal controller gain matrix (K_{lon}) were tuned in-flight to obtain smooth performance and mitigate oscillations. The longitudinal gain matrix K_{lon} is shown

Table 4.5: Base Autopilot Lateral Controller Gain Tuning: Flight Test

	K_{12}	K_{13}	K_{24}
Variation	+15.18%	-86.69%	-56.68%
Description	ϕ/δ_a	P/δ_a	R/δ_r

Table 4.6: Base Autopilot Longitudinal Controller Gain Matrix

	V_T	α	θ	Q	$\int V_{T_{cmd}} - V_T$	$\int \theta_{cmd} - \theta$
δ_t	K_{11}	K_{12}	K_{13}	K_{14}	K_{15}	K_{16}
δ_e	K_{21}	K_{22}	K_{23}	K_{24}	K_{25}	K_{26}

in Table 4.6.

During the flight test, very fast pitch oscillations were observed, therefore the gain matrix element K_{24}, which represents gain from pitch rate to elevator control, was tuned in-flight through the ground control station. This gain (K_{24}) was reduced by 77.61% to mitigate longitudinal oscillations and achieve a steady-state flight. Also, the element K_{23}, which represents gain from pitch angle to elevator control, was reduced by 80.17%. Further, to improve altitude tracking performance, the gain element K_{26} was tuned. K_{26}, which represents gain from the error integral of pitch angle to elevator control, was increased by 5.45%. The summary of the changes to longitudinal gain matrix elements is shown in Table 4.7.

Although the LQR controller was tested rigorously via 100,000 Monte-Carlo simulations, in which it was shown that the controller always performs well, tracks guidance commands with no oscillations, produces bounded states, controls and control rates, this controller was unable to perform well in a real flight test. This shows that most likely the dynamic model used (LTI) was either inaccurate or this particular LQR controller was not robust to un-modeled dynamics that could very well occur during real flight tests. After tuning the base autopilot's LQR controller, a total of 36 successful flight tests were conducted, in which the base autopilot functions as the main

Table 4.7: Base Autopilot Longitudinal Controller Gain Tuning: Flight Test

	K_{23}	K_{24}	K_{26}
Variation	-80.17%	-77.61%	+5.45%
Description	θ/δ_e	Q/δ_e	$\int(\theta_{cmd} - \theta)/\delta_e$

or background autopilot system, while testing more risk prone guidance and control algorithms.

4.2 Imitation Autopilots Trained via Moving Window DAgger

Using moving window data aggregation methodology (MwDAgger), see Section 2.7, three different ANN autopilot models are developed. They are named as Imitation Models 1, 2, and 3. Imitation Model 1 is trained using supervised data generated from the "expert" GNC policy as detailed in Reference [76]. And, the Imitation Models 2 and 3 are trained using the Base Autopilot "expert" policy that combines guidance and control algorithms, as described in Section 2.3. The reason for switching to the base autopilot as the "expert" policy was the computational efficiency of the software implementation for the base autopilot's guidance and control, see Section 4.1. As seen in Tables 2.1 and 4.8, the training times for each of the ANN autopilot models is relatively high (7 to 9 hours), and therefore the base autopilot is used as the "expert" policy for further development of ANN autopilot models.

4.2.1 Imitation Model 1: Unification of Guidance, Navigation and Control

The neural network autopilot used to unify guidance, navigation and control is the same architecture as described in Section 2.6.1, see Figure 2.9. This ANN architecture is named as: Imitation Model 1. The neural network was trained for 750 seconds of trajectory flown by the aircraft, which is about four and a half loops around the waypoints. After training, separate flight simulations are conducted for the "expert" GNC policy and ANN autopilot using 6-DOF EOM. The results are shown in the following figures. In Figure 4.1, the ANN flown trajectory and GNC flown trajectory are compared, the 2D race-track trajectory as flown by the neural network is shown in color cyan (greenish blue) which practically overlaps the 2D trajectory in gray color that is flown by the GNC policy. The training statistics for this model are shown in Table 4.8, see Section 2.7.

Table 4.8: Imitation Model 1 - Training Statistics

Model	DAgger Epochs (Training)	Training Examples	Training Time	Slope of Flight Times	Flight Time (seconds)
MwDAgger	753	1.52e5	8h 55m	N.A.	800+

The lateral-directional and the longitudinal states and control are shown in Figures 4.2 and 4.3 respectively. From Figure 4.2, it can be seen that the roll and yaw rates are almost identical for the GNC and ANN based closed-loop 6-DOF flight simulations, with slight overshoots showing that the ANN control rates are relatively higher by a very marginal amount. For instance, at around 75 seconds, the GNC flight roll rate is $\approx -6°/s$, but for the ANN flight it is $\approx -11°/s$, which is within reasonable limits. This over-prediction only occurs twice within a period of 240 seconds of ANN flight. The ANN control outputs, aileron and rudder are bounded and exhibit no oscillatory behavior.

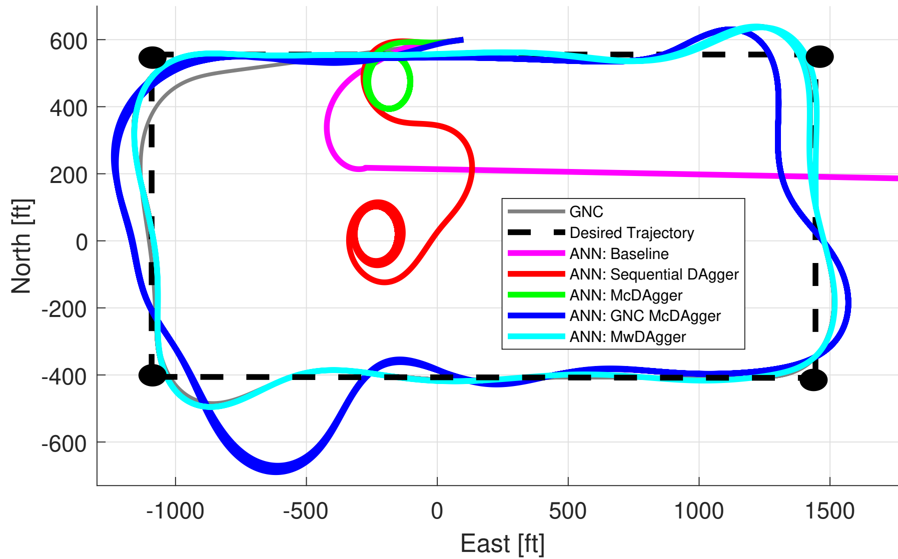


Figure 4.1: Trajectory Comparison: Expert Policy and ANNs: DAgger, McDAgger, GNC McDAgger and MwDAgger

From Figure 4.3, representing longitudinal states and control, a very interesting behavior for ANN predicted throttle control is observed. The ANN learns the appropriate value of steady-state

throttle needed to maintain the airspeed, while the GNC longitudinal controller takes time to react to error buildup and slowly reaches the required throttle. As a result, from Figure 4.2, it can be seen that the airspeed drops to about 48 ft/s while the GNC is controlling the aircraft and reaches steady-state (≈ 50.62 ft/s) only after about 30 seconds. Both the throttle and elevator controls predicted by the ANN through the closed-loop simulated flight exhibit no oscillatory behavior. Velocity tracking is reasonable and within an error margin of 0.5 ft/s, which is exactly the same behavior of the expert GNC policy exhibited.

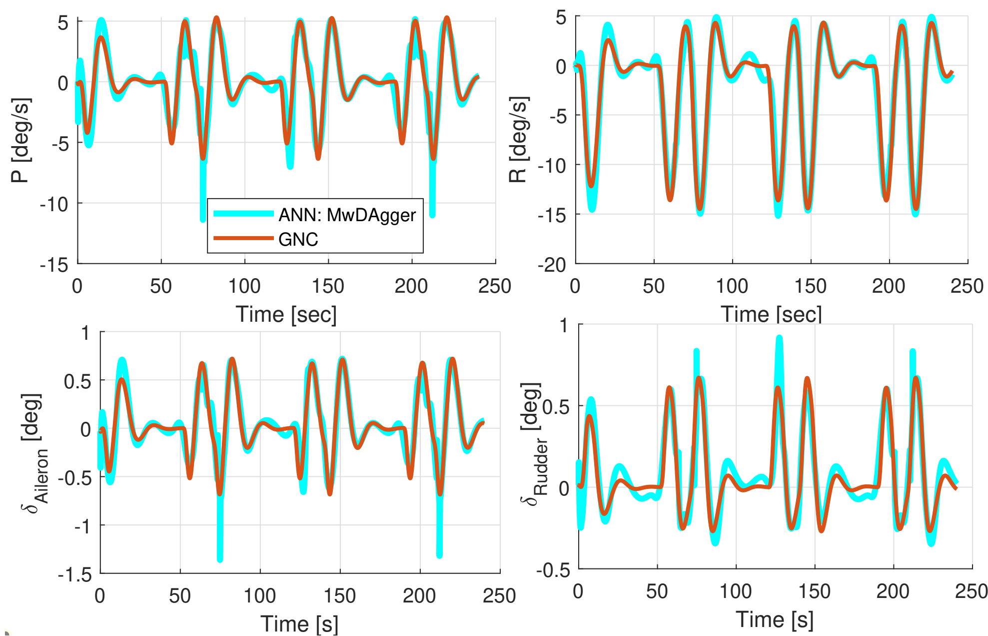


Figure 4.2: Moving Window DAgger ANN-Simulation Flight Test: Lateral States and Controls

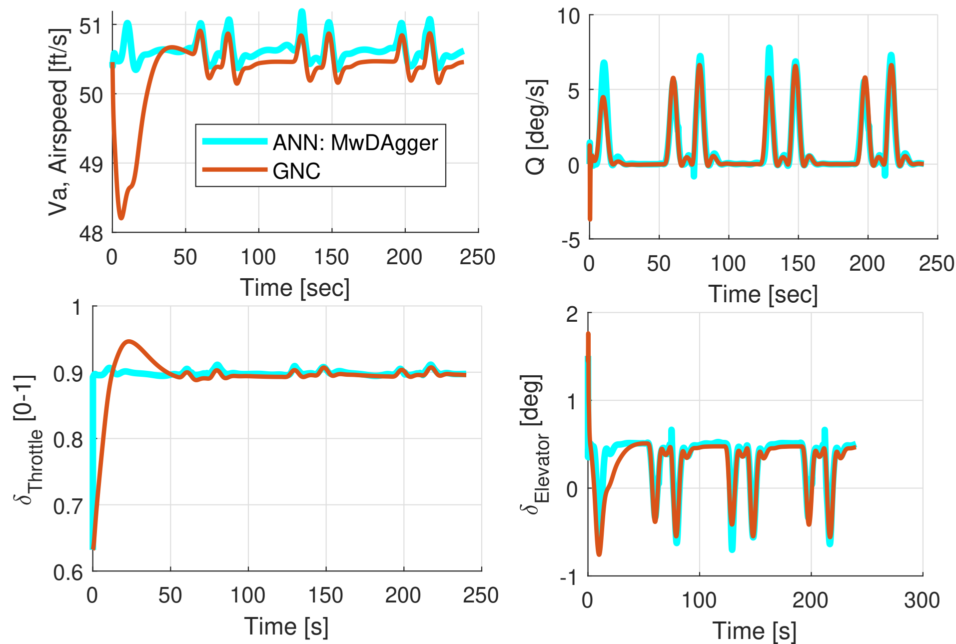


Figure 4.3: Moving Window DAgger ANN-Simulation Flight Test: Longitudinal States and Controls

In Figure 4.4, 3-Dimensional trajectories for GNC and ANN flying the aircraft are compared. It is very interesting to note that in GNC flight the aircraft loses altitude by about 15 ft in the beginning, but the ANN is able to maintain the altitude throughout the flight. From Figure 4.3, particularly the total velocity plot shows that an even higher velocity is maintained by the ANN at turning maneuvers which indicates the learning capability of the neural network in terms of altitude hold. This clearly shows the compensation of lost lift force due to loss of wing surface area at turns, by increasing throttle control by the ANN.

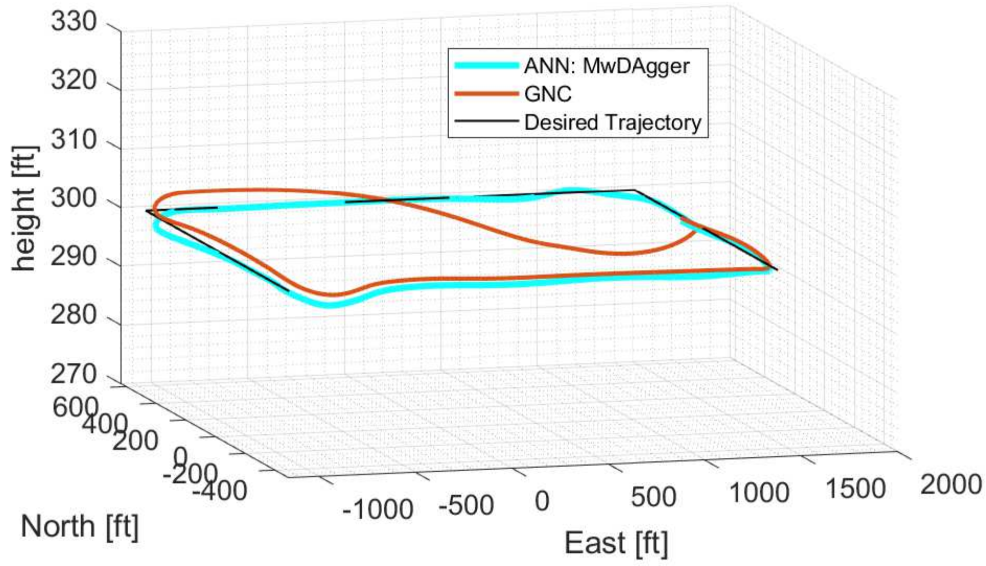


Figure 4.4: Moving Window DAgger - 3-Dimensional Trajectory Comparison with GNC

The goal to imitate an "expert" GNC policy is achieved, and note that the lateral tracking errors (see Figure 4.1) that are caused by this GNC policy are not rectified by the imitation ANN. This shows that the imitation neural network can be only as good as the GNC policy, or the expert from which the ANN is learning and cannot improve or rectify errors without some external feedback.

4.2.2 Imitation Model 2: Unification of Guidance and Control

The moving window DAgger algorithm successfully trains a neural network to imitate an expert policy while unifying the complete sub-systems of a GNC policy. The ANN can safely fly in closed-loop 6-DOF simulations while maintaining aircraft stability at all times, achieving long flights without crashing (no error accumulation). However, there is an inherent drawback in the design of this particular neural network input architecture. This ANN, detailed in Sections 2.6.1 and 4.2.1, does not have flexible inputs in terms of "number" of waypoints. For completeness, the inputs to Imitation Model 1 are shown here, in Equation 4.2.

$$\mathbf{x}^{(\mathbf{i})} = \left[\Delta_{N_j}^{(i)}, \Delta_{E_j}^{(i)}, \Delta_{D_j}^{(i)}, \phi^{(i)}, \theta^{(i)}, \psi^{(i)}, V_T^{(i)}\right], \quad j \in [1,4] \tag{4.2}$$

The first 12 inputs (NED distances between the aircraft and the waypoints) limit the application of this neural network to only four waypoints. Once trained, the input state vector dimension cannot be changed and hence the number of desired waypoints cannot be dynamically updated by the user for different flight test missions. Rather, the ANN weights and biases would have to be re-initialized and then trained to incorporate any more or less number of waypoints. To address this issue, the ANN input architecture is modified and the inputs are made independent of the number of waypoints.

Instead of using the distances between the aircraft and all the waypoints, the perpendicular or the shortest distance from the aircraft to a particular waypoint line (the line being tracked currently) is used as an input. This way, behaviorally the neural network will be trained to track a line, instead of waypoints. By introducing this perpendicular distance as an input, and by removing the NED distances, the directional information (desired heading) is inherently lost from the input state vector. To account for this direction information, the error in heading angle (aircraft vs desired heading) is used as an additional input, see Equation 4.3. This input state vector allows the user to configure and dynamically update to any number of waypoints, and hence the neural network becomes mission agnostic. The only extra effort in this architecture then would be to implement an outer loop that provides the perpendicular distance, heading error and also switches waypoint lines as the aircraft progresses in its flight mission. Another major modification to the neural network input architecture is to utilize error in states, such as airspeed, heading and altitude, so as to represent guidance type inputs to the network.

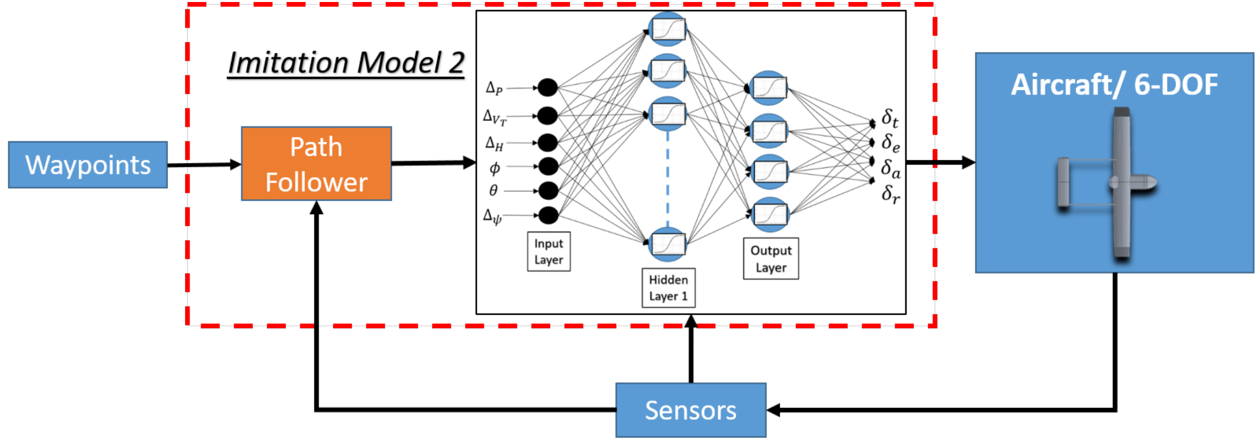


Figure 4.5: Imitation Model 2: Unification of Guidance and Control via Neural Network

Moreover, it was found that the incorporation of navigation filters as internal representations of the neural network makes the training relatively complex and slow, and hence the focus is shifted to unification of guidance and control functionalities only. The complete block diagram showing the ANN implementation and unification is represented in Figure 4.5. This block diagram represents ANN in place of the guidance and control parts of the base autopilot (see Figure 2.2, Section 2.3 for more details).

One input data point for this ANN is defined as follows:

$$\mathbf{x^{(i)}} = \left[\Delta_P^{(i)}, \Delta_{V_T}^{(i)}, \Delta_H^{(i)}, \phi^{(i)}, \theta^{(i)}, \Delta_\psi^{(i)}\right] \tag{4.3}$$

Perpendicular distance from the waypoint line is denoted as Δ_P and the errors in total airspeed, altitude and heading are denoted as: Δ_{V_T}, Δ_H, and Δ_ψ, respectively. One output data point for the Imitation Model 2 neural network is the same as Imitation Model 1, that is the neural network outputs throttle, elevator, aileron, and rudder controls.

A waypoint switching logic is used that changes the waypoint line based on aircraft proximity to the next waypoint, see Section 2.3. Moving window DAgger algorithm is used for ANN data collection and the scaled conjugate gradient descent algorithm is used for training. Training is carried out in mini-batches without the need for reinitialization of the neural network. This neural network is able to generalize much faster (as compared to Imitation Model 1) to stable flights and

complex waypoint paths. Trajectory comparison results are shown in Figure 4.6 for a complex waypoint path that resembles a figure-8 pattern.

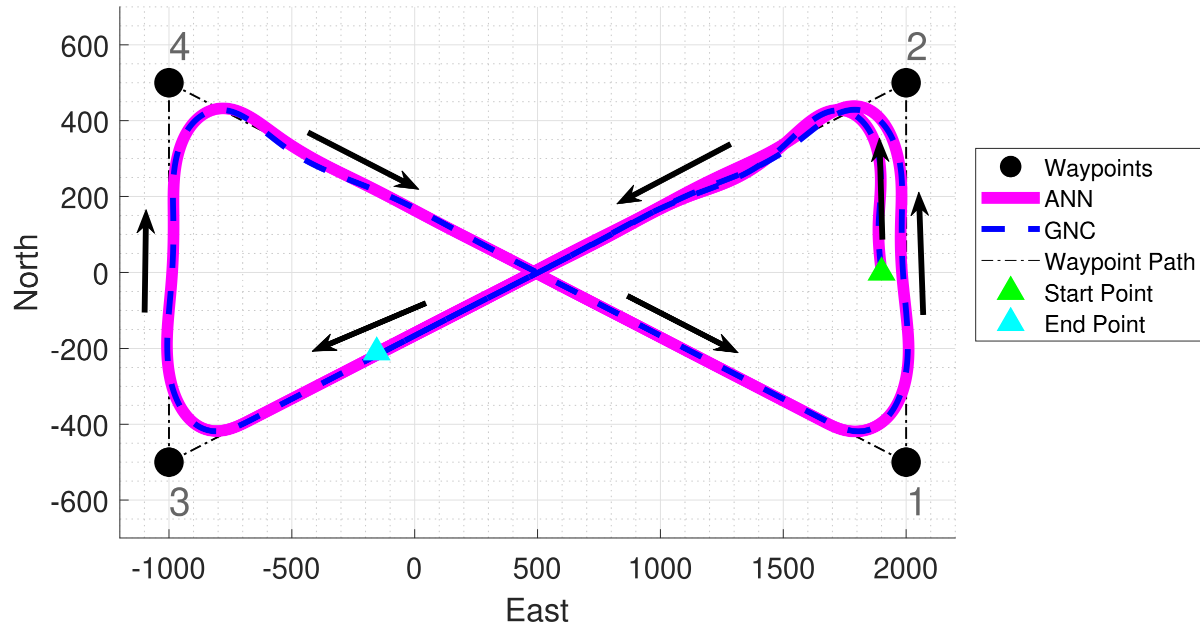


Figure 4.6: Moving Window DAgger - Trajectory Comparison Imitation Model 2 and GNC

The Imitation Model 2 is trained around a trajectory containing 4 waypoints, but instead of following a race-track, the flight path is a figure-8 pattern. As shown in Figure 4.6, the waypoints: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ are arranged from South-North-South-North pattern respectively. This ensures that the data used for training consists of both right and left maneuvers so that the learning is unbiased for lateral tracking. All the longitudinal and lateral states match between the GNC and ANN simulated flights.

The training for Imitation Model 2 is carried out in conjunction with the grid search neural network architecture optimization algorithm, see Section 2.8. And therefore, the MwDAgger training progresses efficiently and very fast, while finding an optimal neural network architecture for this problem. The architecture obtained consists of one hidden layer with 24 nodes or neurons and one output layer consisting of four nodes. All nodes are assigned tanh activation function. The time taken for this training is about 4 hours and 11 minutes and it takes 86 MwDAgger iterations to complete.

4.2.3 Imitation Model 3: Decoupled Longitudinal and Lateral Controllers

The Imitation Model 2 exhibits excellent properties in terms of mimicking the expert GNC policy, generalizing over right and left maneuvers and is waypoint mission agnostic. However, it was found that the Imitation Model 2 cannot generalize over certain difficult initial conditions (aircraft states) mainly pertaining to large drift from the input magnitudes with respect to the data that was used to train it. For example, the first input used to train the neural network, consisting of the perpendicular distance from the waypoint line, is mostly concentrated in the range of $0 \sim 20$ ft with an average value of 34.77 ft. Due to this bias in data, the neural network used in Imitation Model 2 is unable to predict and extrapolate to perpendicular distances that are out of this range; this is also true for other inputs, such as the error in velocity and the error in altitude.

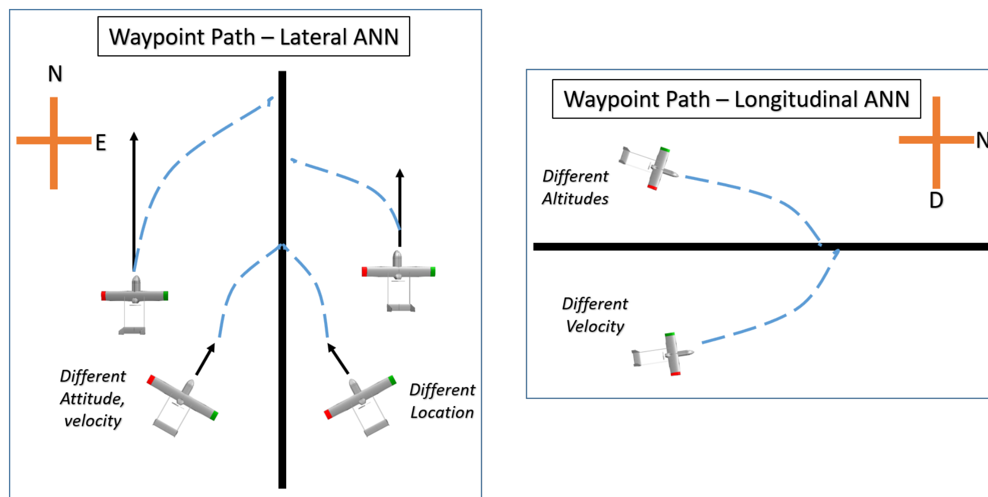


Figure 4.7: Imitation Model 3 Training Concept

In order to generalize over a large range of inputs, a new methodology is developed in which a randomized initial condition for the aircraft state is selected before the start of MwDAgger training. Moreover, the training does not need to be carried out around a fixed waypoint pattern, instead the training can be focused on achieving steady-state level wings flight and tracking convergence to just one waypoint path given that the dataset consists of multiple initialization points. These multiple random initial conditions consist of aircraft location with respect to the waypoint path (straight line), different airspeeds and airflow angles, different attitudes and attitude rates. There-

fore, in essence an outer loop for training is implemented that runs as a Monte-Carlo random initial condition generator. Also, the Imitation Model 3 is divided into decoupled lateral and longitudinal neural networks that are mainly used as controllers. The concept of this method is depicted in Figure 4.7.

Lateral Neural Network: One input data point to this neural network consists of three variables as shown in Equation 4.4. First input variable is the angle (η_{Lat}) between the aircraft's ground speed and an imaginary $\overrightarrow{L_{lat}}$ vector as defined in Section 2.3.1. Other inputs include the ground speed (V_{XY}) and the roll angle (ϕ). As described in Section 2.3.1, the η_{Lat} angle is used to generate the lateral acceleration required to fly the aircraft towards the waypoint path, which in turn generates the required roll angle (ϕ_{cmd}). Hence, the neural network learns to directly map from this η_{Lat} angle, the ground speed and the roll angle, to aileron and rudder controls.

$$\mathbf{x}^{(\mathbf{i})} = \left[\eta_{Lat}^{(i)}, V_{XY}^{(i)}, \phi^{(i)}\right] \tag{4.4}$$

The lateral ANN is trained to output perturbed values for aileron and rudder controls instead of total values as shown in Equation 4.5. Therefore, the trim values need to be added to these perturbed controls before exciting the aircraft model. This ANN consists of one hidden layer with 20 nodes or neurons, and one output layer with two nodes. Both layers' nodes consist of tanh activation functions.

$$\mathbf{y}^{(\mathbf{i})} = \left[\delta_{a_p}, \delta_{r_p}\right] \tag{4.5}$$

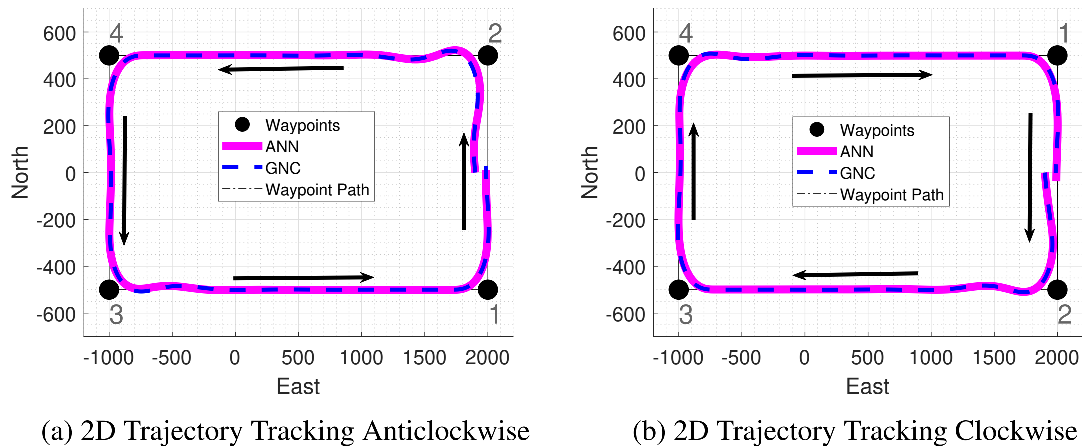
Longitudinal Neural Network: One input data point for this neural network consists of four inputs as shown in Equation 4.6. The input variables are: error in pitch angle, error in velocity and the integrals of these error states.

$$\mathbf{x}^{(\mathbf{i})} = \left[\Delta\theta^{(i)}, \Delta V_T^{(i)}, \int \Delta\theta^{(i)}, \int \Delta V_T^{(i)}\right] \tag{4.6}$$

The longitudinal neural network is trained to output perturbed throttle and elevator controls as

shown in Equation 4.7. This ANN consists of three hidden layers with 50 neurons each, and one output layer with two neurons. Both layers' nodes consist of tanh activation functions.

$$\mathbf{y}^{(\mathbf{i})} = \left[\delta_{t_p}, \delta_{e_p}\right] \tag{4.7}$$



(a) 2D Trajectory Tracking Anticlockwise

(b) 2D Trajectory Tracking Clockwise

Figure 4.8: Imitation Model 3

Both lateral and longitudinal neural networks are trained using MwDAgger data aggregation with scaled conjugate gradient descent algorithm. For the lateral neural network, the outer loop Monte-Carlo random initial condition generator is run 10 times to provide sufficient data for generalization, and it is run 100 times for the longitudinal neural network. The time taken for training the lateral neural network is only about 2 minutes and 5 seconds; and for the longitudinal network it is approximately 3 hours 21 minutes. When these neural networks are tested together in closed-loop 6-DOF flight simulations, the combined controllers are able to generalize to different initial conditions, clockwise, anticlockwise and figure-8 trajectories, see Figures 4.8a, 4.8b, and 4.9.

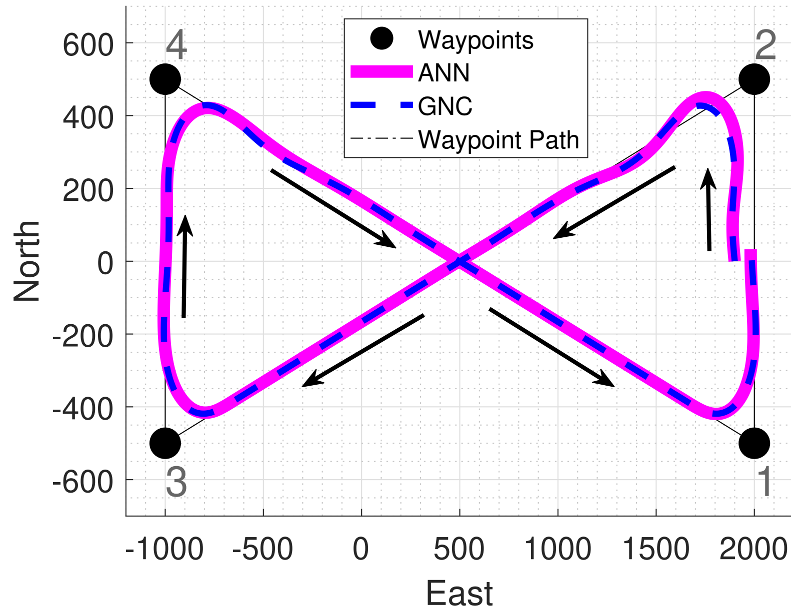


Figure 4.9: Imitation Model 3: 2D Trajectory Tracking Figure-8

After training completion, both the neural networks are subjected to closed-loop 6-DOF flight simulations with multiple random initial conditions. These Monte-Carlo evaluation simulations consist of 10,000 different cases of flights, for each longitudinal and lateral neural networks, in which the aircraft states at the start of the simulation are varied as described in Section 3.4. The success and failure rates for these neural networks is shown in Table 4.9.

Table 4.9: Success and Failure Rates for Monte-Carlo Flight Test Simulations Conducted on Imitation Model 3

ANN	Total Cases	Success	Failure	Percentage Success	Failure Conditions
Longitudinal	10,000	9988	12	99.88%	$\delta_a/dt : Max = 24.61°/s$
Lateral	10,000	9997	3	99.97%	$\delta_a/dt : Max = 18.08°/s$

With more than 99% success rate in each case for longitudinal and lateral neural networks, the reliability of these neural network controllers are proven to be effective in closed-loop 6-DOF flight simulations. However, as shown in Section 4.1, the initial (real) flight tests for the base autopilot were unsuccessful even after passing more than 99% Monte-Carlo flight test simula-

tions, the reliability of these simulated flight tests highly depend on the aircraft 6-DOF model used (physics-based). Since the controllers in the base autopilot were specifically designed around a low-fidelity aircraft model, they are only reliable and optimal with respect to that aircraft model. If the actual aircraft dynamics are significantly different than the estimated model parameters, then the controller may fail in real flight tests. Therefore, a similar case is observed with the real flight tests for Imitation Model 3, as described in the next section.

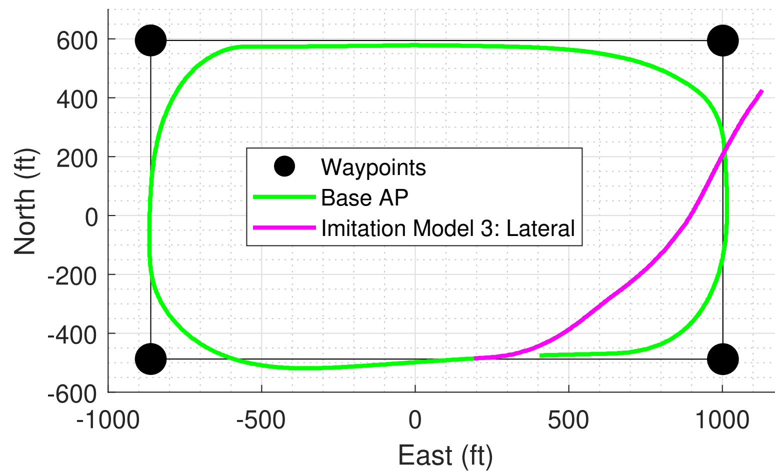


Figure 4.10: Imitation Model 3: Lateral Neural Network - Trajectory Tracking

Flight Test Results for Imitation Model 3: The flight test results for the Imitation Model 3 are shown in Figures 4.10 and 4.11. From Figure 4.10, it can be seen that the controllers were switched in-flight from the base autopilot (Base AP) to the Imitation Model 3 lateral neural network, at the South waypoint leg around 200 ft East local frame coordinate. From this switching point, the lateral neural network controller cannot track the waypoint path, however it maintains stability throughout its flight time of about 18.5 seconds, after which the autopilot mode is disabled by the pilot and manual control is taken of the aircraft. The Imitation Model 3 longitudinal neural network was engaged twice, dynamically in-flight, as shown in Figure 4.11. During each engagement, the longitudinal neural network controller reacted abruptly resulting in a loss of altitude of about 32 ft in just 3.5 seconds. The longitudinal neural network controller was kept engaged longer the second time and it did try to recover in altitude, however at the cost of very fast control rates, due to which

it was disabled.

Monte-Carlo simulation results for reliability analysis and the initial flight test results for both the base autopilot and the Imitation Model 3 clearly show that the GNC autopilot designs are inherently dependent on the physics-based aircraft dynamic model reliability and accuracy. The traditional controllers, such as LQR, simply lack the structure to encompass all the non-linearity associated with actual aircraft, if low quality dynamic model of an aircraft is used. On the other hand, it has been shown that neural networks can effectively behave as standalone GNC systems or just controllers; however, if they are trained on existing GNC systems, their reliability is again questionable and they can be only as good as the expert policy.

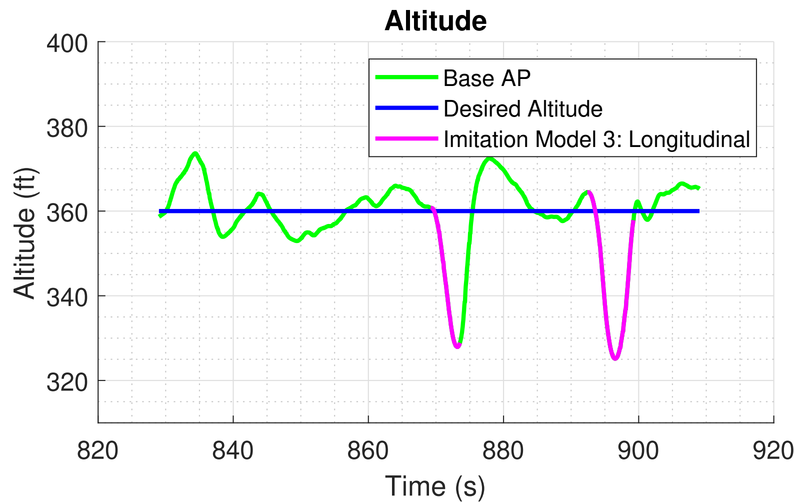


Figure 4.11: Imitation Model 3: Lateral Neural Network - Altitude Tracking

In light of this conclusion, the research question that lies ahead is that of designing reliable controllers that possess the mathematical structure which can safely extrapolate to unseen flight conditions, and also learn from low-fidelity aircraft models and in turn generalize over real aircraft flights. A mathematical structure that not only generalizes but also learns a behavior or "policy" that is agnostic to large variations in aircraft model parameters and can continue to learn such behaviors is needed. Such a design of a neural network controller is shown to surpass expectations in real flight tests which is trained using Deep Deterministic Policy Gradients algorithm, as detailed in the next section.

4.3 DDPG Trained Neural Network

After success with imitation learning, learning from environment methodology (reinforcement learning) is explored. Since a flight controller is one of the most difficult parts from the sub algorithms of a GNC system, the focus here is shifted to designing a neural network controller policy using the reinforcement learning paradigm. Decoupled longitudinal and lateral controllers are trained using aircraft's respective LTI models with the goal of minimizing perturbations in states and maintaining a steady-state wings-level flight. Validation and verification for the longitudinal neural network controller is carried out in closed-loop simulations as well as flight tests. In real flight tests, it is shown that this neural network is able to generalize predictions over large perturbations or errors in airspeed and pitch angle inputs, and safely produce required perturbed throttle and elevator outputs. The lateral controller is developed using a unique reward scheduling technique and is validated in LTI and 6-DOF closed-loop simulation flights.

In this research, MATLAB's reinforcement learning toolbox [96] is used, as it readily provides the Deep Deterministic Policy Gradients (DDPG) learning algorithm. The basic idea behind this algorithm is to simultaneously train two deep neural networks representing the actor (policy) and critic functions. To use MATLAB's reinforcement learning toolbox the user has to set up a few functions and variables. A one step dynamic model (aircraft model) needs to be set up that provides next step states based on control input excitation, in turn providing a crucial part of data for training. This dynamic model is defined inside a "step" function, an object property of MATLAB's reinforcement learning toolbox. A "reset" function is also set up by the user that resets the initial state of the dynamic system before start of every episode. The reward function (immediate reward) needs to be carefully designed by the user depending on the control problem, and is implemented as part of the "step" function. Other significant properties for the "step" and "reset" functions that are set up by the user include: ranges for terminal states, ranges for selecting randomized states, and control variable ranges. Apart from these standard functions, the user defines and tunes some crucial parameters, such as the learning rates for the actor and the critic neural network functions, the discount factor, mini batch size, neural network architectures for both actor and critic (layers,

neurons and activation functions) and the noise parameters.

4.3.1 Longitudinal Environment: Aircraft Model and Reward Function

The environment is formulated using the longitudinal Linear Time Invariant (LTI) model of the Skyhunter UAS, see Section 3.1. A discrete model is created using aircraft continuous A and B matrices, see Equation 4.8.

$$X_{k+1} = A_D X_k + B_D U_k \tag{4.8}$$

Here, X represents the longitudinal state of the aircraft, consisting of $\{V_T, \alpha, \theta, Q\}$, and U consists of the control variables: $\{\delta_t, \delta_e\}$. A_D and B_D are the discrete system matrices of the LTI model.

The environment-based reward function that is used as the learning signal for the reinforcement learning procedure must be defined in such a way that it can represent the desired goals effectively and also not restrict data collection. The goal for the longitudinal controller is straightforward: maintain a steady-state flight with airspeed and altitude holds. Since, in the LTI model the total velocity of the aircraft is available, it can be directly incorporated in the reward function. However, altitude is not part of the LTI model. Hence, the flight path angle (γ) is used as an indirect measurement for the altitude of the aircraft, and a flight path angle of zero indicates an altitude hold. Note that the flight path angle is denoted by (γ) and the discount factor is denoted by γ_d see Equation 4.9, with a subscript of d to differentiate them from each other. Other goals for this controller involve forcing the dynamic system (aircraft) to its trim point while minimizing the energy input into the system (minimize controls). Therefore, the reward function consists of three major elements divided as discrete and continuous rewards:

- The first element consists of a discrete reward which gives a high positive value each for the perturbed velocity (V_{T_p}) and the γ angle, when each of these variable values are less than certain independent thresholds. Since perturbed velocity is used, ideally the threshold

for velocity should be close to zero. In a real flight the total velocity is estimated using the Pitot tube - differential pressure airspeed measurement. It would be prudent to model the uncertainty bounds from this sensor and use that value as the threshold for perturbed velocity when giving positive reward to the system.

Similarly, the flight path angle which is estimated using vertical GPS velocity (V_z) and the Z-acceleration (a_z) from the IMU, the uncertainty bounds from these sensors can indicate the uncertainty in the flight path angle measurement, which can be utilized as the threshold for providing reward to the system based on the flight path angle.

- Second element represents a high negative value when V_{T_p} and γ are above certain independent thresholds. The perturbed velocity is a value representing how much off trim the aircraft velocity is, or in other words, it represents the difference between actual and trim velocities. The minimum airspeed, which is 1.3 times the stall speed ($V_{min} = 1.3V_{stall}$) must be maintained to generate enough lift throughout the flight. The difference between the aircraft trim speed and this minimum velocity is used as the threshold to give a high penalty to the system.

  The maximum rate of climb (or rate of descent) can be determined for a specific aircraft depending on the flying qualities desired and also its structural integrity. Depending on this factor ($\dot{h}_{max}$), the maximum flight path angle can be computed using the direct relationship:

$$\dot{h} = V_T \sin\gamma \tag{4.9}$$

- Third element represents a quadratic continuous valued reward function combining V_{T_p}, γ, and perturbed controls: δ_{t_p} and δ_{e_p}. Quadratic rewards (penalties) for states ensure that the data collected with its corresponding reward values exhibits a smooth transition from a non-steady to a steady-state. The quadratic penalties for the control variables force the controller to use less energy input to the system and also puts a limit (indirectly) on the control rates: sudden changes in control values.

The reward function is represented in Equations 4.10. The first reward element (R_1) returns a reward value of r_{11} if the square of the perturbed velocity ($V_{T_p}^2$) is less than t_{11} threshold, and it also adds a reward value of r_{12} if the absolute value of the flight path angle is less than t_{12} threshold. The second reward element (R_2) returns a high negative value ($-r_{21}$), if the absolute value of perturbed velocity exceeds t_{21} threshold, and also adds to it a high negative penalty ($-r_{22}$), if the absolute value of the flight path angle exceeds t_{22} threshold. The third element (R_3) simply adds the weighted ($w_{31}, w_{32}, w_{33}, w_{34}$) and squared values for the perturbed velocity, flight path angle, perturbed throttle and perturbed elevator.

The total reward (R) is the summation of the three reward elements (R_1, R_2, R_3).

$$R_1 = r_{11}\left[V_{T_p}^2 < t_{11}\right] + r_{12}\left[|\gamma| < t_{12}\right] \tag{4.10a}$$

$$R_2 = -r_{21}\left[|V_{T_p}| >= t_{21}\right] - r_{22}\left[|\gamma| > t_{22}\right] \tag{4.10b}$$

$$R_3 = -w_{31}V_{T_p}^2 - w_{32}\gamma^2 - w_{33}\delta_{t_p}^2 - w_{34}\delta_{e_p}^2 \tag{4.10c}$$

$$R = R_1 + R_2 + R_3 \tag{4.10d}$$

Here, r, t, and w are discrete rewards, state thresholds and quadratic function weights respectively. All these three elements of the reward function are added together to provide a scalar reward feedback for training. Therefore, the reward function ensures smooth changes in states and controls for the aircraft while maintaining steady-state straight line flight, with airspeed and altitude holds.

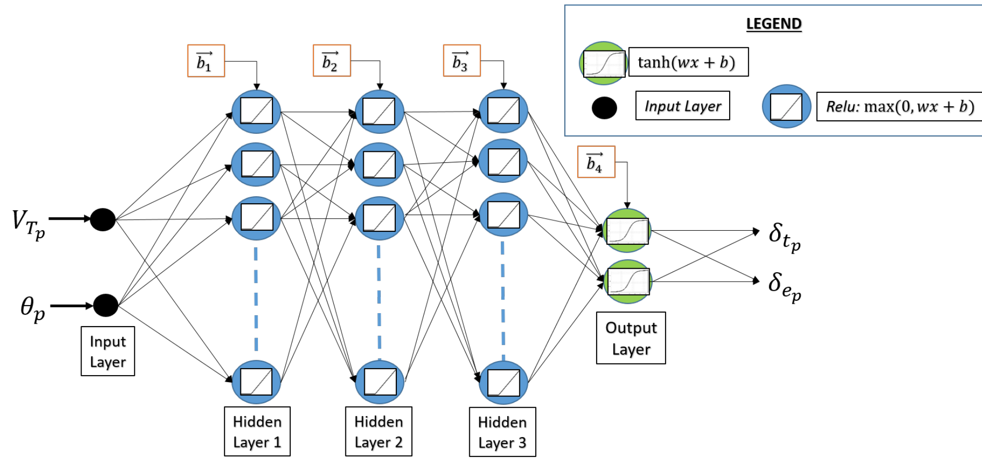


Figure 4.12: Longitudinal Neural Network Controller Architecture

4.3.2 Actor and Critic Neural Networks - Longitudinal Controller

Using reinforcement learning terminology, the ANNs consisting of the actor and critic functions, combined together as one entity are called as the reinforcement learning "agent". One neural network behaves as the controller policy, called the "actor" and the other neural network provides cumulative reward feedback and is called the "critic". For the Longitudinal LTI model, the actor neural network is trained to output perturbed throttle and elevator controls. The inputs to the neural network consist of perturbed states: total velocity and pitch angle. It is a two input and two output neural network consisting of three hidden layers and one output layer. The three hidden layers consist of rectified linear units (relu) as the activations for the neurons and the output layer activation is tanh function. All layers are fully connected with weights and biases in each node. Each fully connected hidden layer consists of 50 nodes (or neurons). The total learnable parameters consisting of weights and biases sum to 5,352, which shows that the complexity of ANN controllers is orders of magnitude more than LQR controllers as designed in Section 2.3.2. The actor ANN architecture is represented in Figure 4.12.

The critic ANN architecture is shown in Figure 4.13. The critic ANN takes in the observation (perturbed states) passes them through one fully connected layer with relu activations and 50 nodes (Hidden Layer 1, see Figure 4.13). The outputs of Hidden Layer 1 are passed through a linear

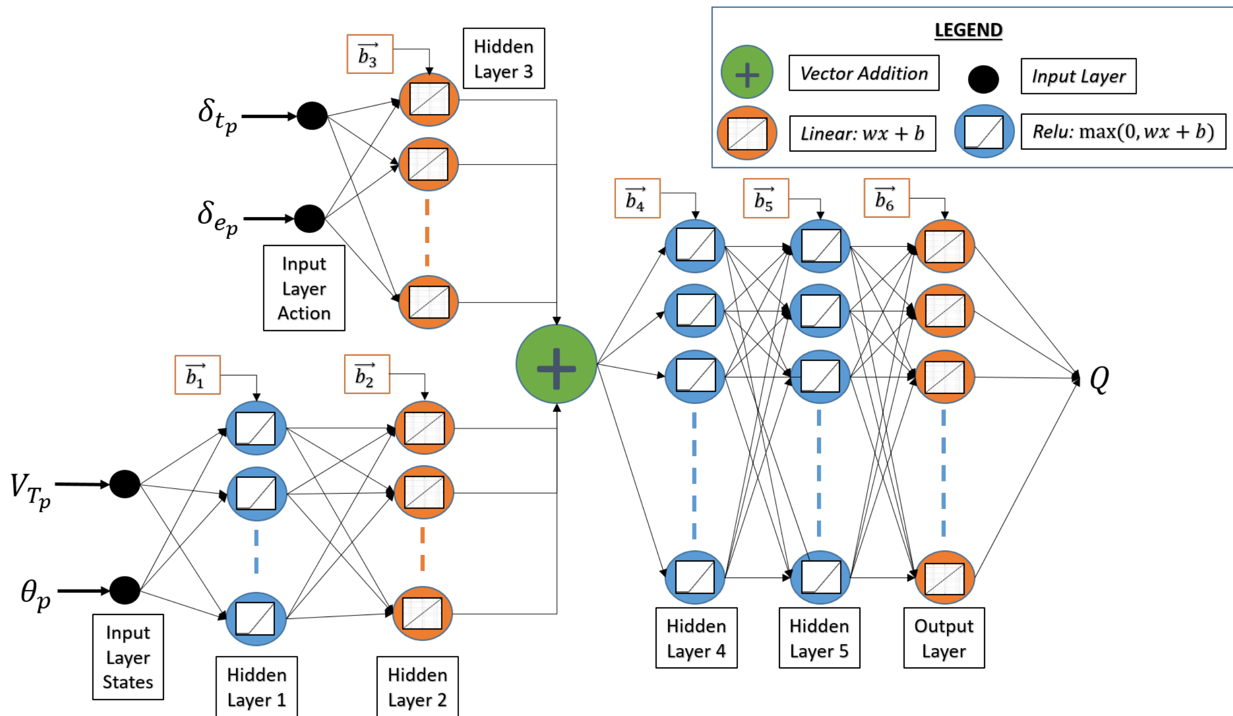


Figure 4.13: Critic Neural Network Architecture: For Longitudinal Controller

fully connected layer (Hidden Layer 2). On a different input path to the critic neural network, the perturbed control values are passed through a fully connected linear activation layer (Hidden Layer 3). The outputs of Hidden Layers 2 and 3 are added element-wise before passing them through a relu fully connected layer (Hidden Layer 4). The outputs from Hidden Layer 4 are passed through another relu fully connected layer (Hidden Layer 5). The Output Layer of the critic neural network consists of a fully connected linear layer, and outputs the Q value (cumulative reward). Each hidden layer in this neural network consists of 50 nodes (or neurons). In total, the critic network consists of five hidden layers and one output layer, with a total of 5,451 learnable parameters.

4.3.3 Training Results - Longitudinal Controller

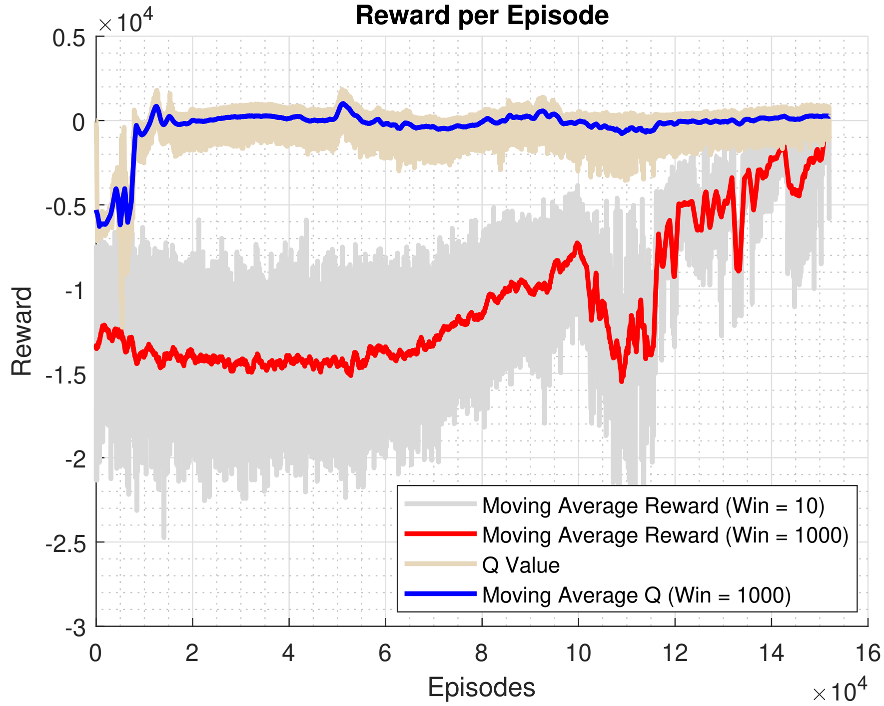


Figure 4.14: Cumulative Rewards during training: Longitudinal Neural Network

Using the reward function and the discretized aircraft dynamics, the agent is subjected to training. The maximum number of training episodes are set to 200,000. An episode is defined as the time period in which the agent makes control decisions, receives reward and collects data for training until the states reach an unstable or impractical threshold value or the maximum episode time is reached. The cumulative reward or the value depends on the episode length and the immediate reward attained at each step. An experience buffer size of 100,000 data points is set, with initial learning rates of 0.001 each for actor and critic neural networks. For both networks, Adam optimizer (see Section 2.5.3) is chosen to update network weights and biases, which is based on adaptive momentum estimation [80]. A discount factor of 0.9 is chosen to account for future rewards.

The training is carried out on an Intel Core i7-8700 CPU, 3.20 GHz, with matrix computations offloaded to a NVIDIA GeForce GT740 GPU. The training stopping criterion was chosen as a high

positive value for the average cumulative reward. The moving average cumulative rewards and the predicted value (Q) are shown in Figure 4.14. This training duration was 50 hours and 17 minutes, with the stopping criterion being achieved at episode number 152,404. During training, agents with a specific positive threshold value for cumulative reward are saved intermediately.

4.3.4 LTI Simulations - Longitudinal Neural Network Controller

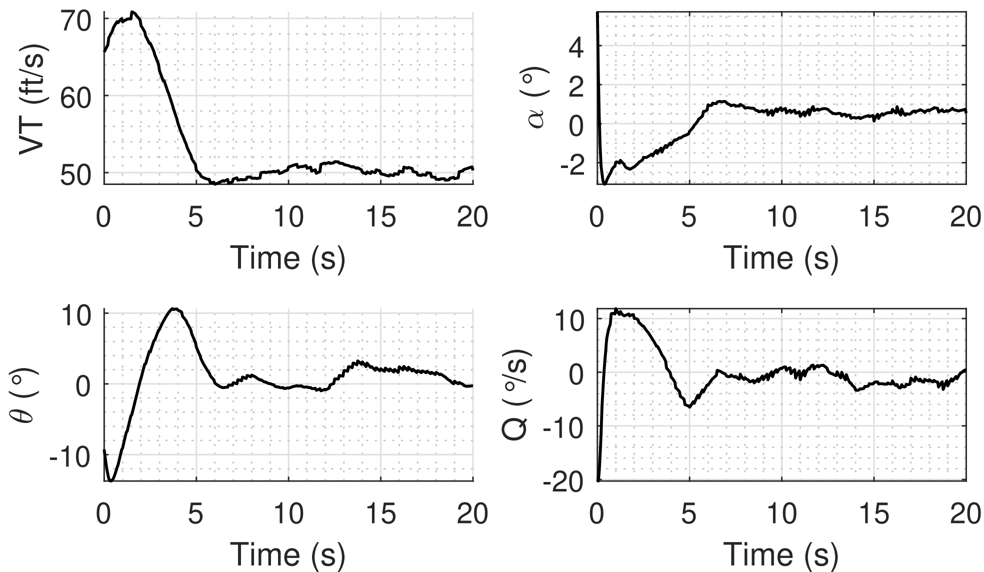


Figure 4.15: Longitudinal Neural Network Controller, Closed-Loop LTI Simulation Results: States

After training the agent, the actor ANN with highest cumulative reward is selected from all the saved agents and it is subjected to numerous closed-loop simulations using the LTI model of the aircraft at an update rate of 20 Hz (0.05 seconds discrete time steps). During LTI simulations, the states are subjected to a temporally correlated Gauss-Markov process noise (Ornstein-Uhlenbeck process [141]) to test the robustness of the ANN predictions. Results from one of the tests are shown in Figures 4.15 and 4.16. The Skyhunter aircraft has a cruise speed of 50.63 ft/s, a trim angle of attack of 0.73 degrees, and a trim throttle of about 60%. The LTI model is initialized with a set of difficult initial conditions: $V_T = 65.63$ ft/s, $\alpha = 5.73°$, $\theta = -10°$, and $Q = -20°/s$.

The ANN predicts throttle and elevator in such a way that it tries to bring the aircraft back to trim conditions. It can be clearly seen that the ANN controller predicts stable throttle and elevator controls, which are well bounded, and all the states converge to trim points.

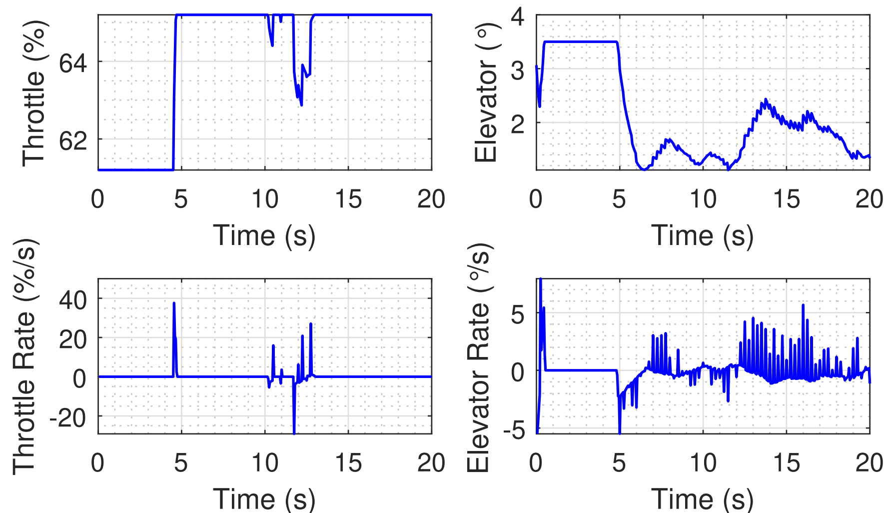


Figure 4.16: Longitudinal Neural Network Controller, Closed-Loop LTI Simulation Results: Control Variables and Rates

As seen in Figure 4.15, airspeed and angle of attack, both settle around the trim value, and the pitch angle and pitch rate settle around zeros. While running closed-loop LTI and ANN simulation, temporally correlated noise is added to all four states at every five time steps (4 Hz or 0.25 seconds). The mean for the noise is set as zero, and the variances for the states are: $\sigma_{V_T} = 5$ ft/s, $\sigma_\alpha = 1^\circ$, $\sigma_\theta = 2^\circ$, $\sigma_Q = 1^\circ/\text{s}$. It can be seen in Figure 4.16, that the control rates are well bounded, throttle being within $\pm 40\%/\text{s}$ and elevator rate within $\pm 8^\circ/\text{s}$.

4.3.5 Lateral Environment: Aircraft Model and Reward Function

As described in Section 4.3.1, the reinforcement learning environment is formulated using the aircraft LTI model. Referring to Equation 4.8, in this case it represents the lateral LTI model with $X \equiv \{\beta, \phi, P, R\}$ and $U \equiv \{\delta_a, \delta_r\}$. The goal for the Lateral neural network controller is to maintain a leveled flight with roll angle attitude of zero degrees, while minimizing the use of

aileron controls and its rate. In contrast to the general consensus that a Lateral controller design should be relatively less difficult than its Longitudinal counterpart, it was otherwise found that tuning the reward function and the neural network architecture was particularly very difficult for this problem. Therefore, a novel idea is developed, in which the reward function is "scheduled" based on the lateral state of the aircraft. This new terminology is defined as "Reward Scheduling" and was found to be very effective in training the DDPG neural network controller, in terms of both training time and achieving desired controller performance.

The reward function consists of two major parts based on the lateral states (ϕ, P) of the aircraft: (1) a positive discrete reward combined with a low negative quadratic continuous reward, and (2) a negative quadratic continuous reward. Note that only one of these rewards is applied in an episodic step. The first part of the reward function is further divided into seven different rewards based on roll angle and roll rate conditions. The basic idea behind these different sub rewards for the first part is that the discrete value of reward should be lower if there is a large error in roll angle and vice-versa. Also, each sub reward combines another necessary condition involving roll rate, which ensures that the reward is applied only if the roll rates are bounded.

The first part of the reward is mostly positive, which encourages the agent to be in these states. This first part of the reward is applied only if the error in roll angle and the roll rate state are of opposite sign, or in other words, if the aircraft is in a positive roll angle state and if the roll rate is negative, this condition is considered good, and a positive reward is given. On the other hand, if the aircraft is in a positive roll angle state and if the roll rate is also positive, this will lead to instability, and hence the second part of the reward is applied which is negative. The structure of this reward scheduling is detailed in Algorithm 8.

In the Algorithm 8, conditions C_0 and C_1 apply very small positive rewards in case of high roll perturbations and high roll rates. The four conditions (C_2 to C_5) ensure that if the roll angle is in a particular range and if the roll rate is also in some bounded value, then there is a positive reward. The discrete rewards (D_0 to D_5) are in increasing order, that is, $D_0 < D_1 < D_2 < D_3 < D_4 < D_5$, and the corresponding roll angle bounds are in decreasing order, that is, $\phi_0 > \phi_1 > \phi_2 > \phi_3 > \phi_4$.

Algorithm 8 Reward Scheduling for Lateral Neural Network Controller

1: $C_0 : |\phi| > \phi_0$ and $|P| \geq P_0$
2: $C_1 : |\phi| > \phi_0$ and $|P| < P_0$
3: $C_2 : |\phi| \in (\phi_1, \phi_0]$ and $|P| \in (0, P_1)$
4: $C_3 : |\phi| \in (\phi_2, \phi_1]$ and $|P| \in (0, P_2)$
5: $C_4 : |\phi| \in (\phi_3, \phi_2]$ and $|P| \in (0, P_3)$
6: $C_5 : |\phi| \in (\phi_4, \phi_3]$ and $|P| \in (0, P_4)$
7: $C_6 : \phi = 0$
8: **if** sign(ϕ) $\neq$ sign(P) **then**
9:     **if** C_0 **then**
10:         $R = -w(\phi^2 + P^2) + D_0$
11:     **else if** C_1 **then**
12:         $R = -w(\phi^2 + P^2) + D_1$
13:     **else if** C_2 **then**
14:         $R = -w(\phi^2 + P^2) + D_2$
15:     **else if** C_3 **then**
16:         $R = -w(\phi^2 + P^2) + D_3$
17:     **else if** C_4 **then**
18:         $R = -w(\phi^2 + P^2) + D_4$
19:     **else if** C_5 **then**
20:         $R = -w(\phi^2 + P^2) + D_5$
21:     **else if** C_6 **then**
22:         $R = -W_P P^2 + D_{\phi=0}$
23:     **end if**
24: **else**
25:     $R = -w(\phi^2 + P^2)$
26: **end if**

The roll rates associated with these conditions are also in decreasing order, that is, $P_0 > P_1 > P_2 > P_3 > P_4$. Therefore, these conditions dictate the behavior of roll rate control based on the roll angle state: if the roll angle perturbation is higher, then the controller can react fast to reduce the error and on the other hand if the roll angle perturbation is small then it will only be rewarded if the roll rates are very small in value. The last condition C_6, line 21 of Algorithm 8 applies when the roll perturbation is zero degrees and gives a highly weighted negative (W_P) reward to roll rate ($-W_P P^2$), combined with a high positive reward for achieving level wings condition ($D_{\phi=0}$).

The second part of the reward, line 25 of Algorithm 8 applies a quadratic continuous reward when the controlling behavior is opposite of what is expected.

4.3.6 Actor and Critic Neural Networks - Lateral Controller

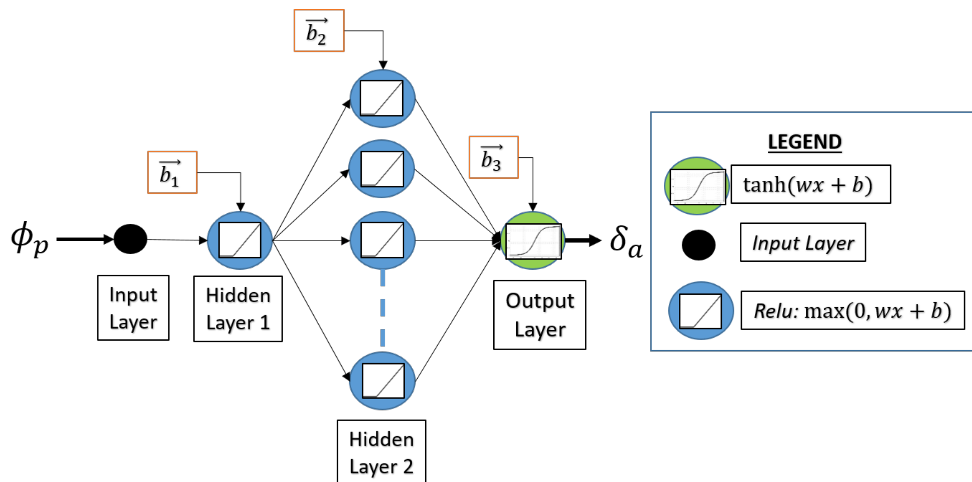


Figure 4.17: Lateral Neural Network Controller Architecture

The Lateral neural network controller (Actor) is trained to output aileron (δ_a), based on perturbed roll angle (ϕ) input. It is a single input and single output neural network consisting of two hidden layers and one output layer. All layers are fully connected with weights and biases in each node. The first hidden layer consists of one node (or neuron) and the second hidden layer consists of 20 nodes, with both layers having relu as their activation function for all nodes. The output layer consists of the tanh activation function with one output node. The total learnable parameters con-

sisting of weights and bases are 63. This Lateral neural network controller is represented in Figure 4.17.

For this Lateral controller, the critic neural network is shown in Figure 4.18. The architecture of this neural network critic is very similar to the one used for the Critic for Longitudinal controller. The number of hidden and output layers are same for this critic (Lateral controller) as of the critic for the Longitudinal controller, and equal to five hidden layers and one output layer. The number of nodes or neurons in each hidden layer is equal to 20. The total learnable parameters for this critic neural network are 545.

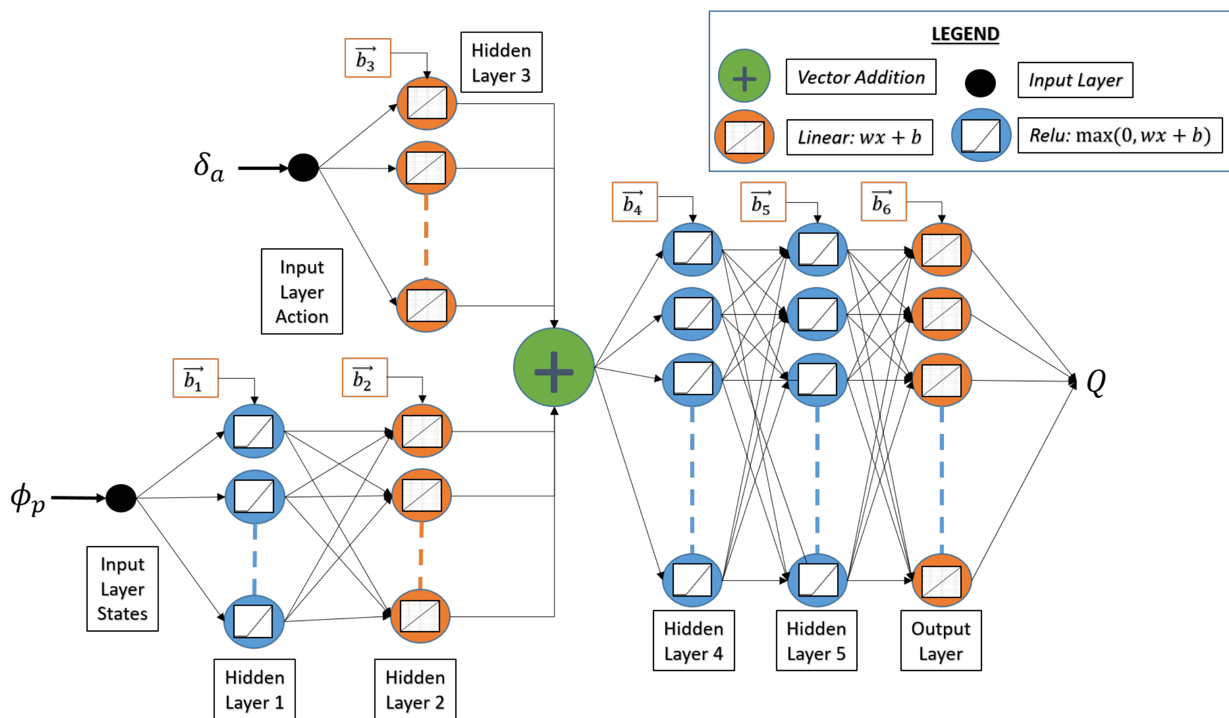


Figure 4.18: Critic Neural Network Architecture: For Lateral Controller

4.3.7 Training Results - Lateral Controller

Using the reward function, Lateral environment and the neural network agent, the training is carried out on an Intel Core i7-8700 CPU, 3.20 GHz, with matrix computations offloaded to a NVIDIA GeForce GT740 GPU. The maximum number of episodes are set to 10,000, with both actor and critic learning rates of 0.001, and a discount factor value of 0.99 is chosen. For both neural net-

works, Adam optimizer [80] is used. The moving average cumulative rewards and the predicted Q value are shown in Figure 4.19 up to episode 3000.

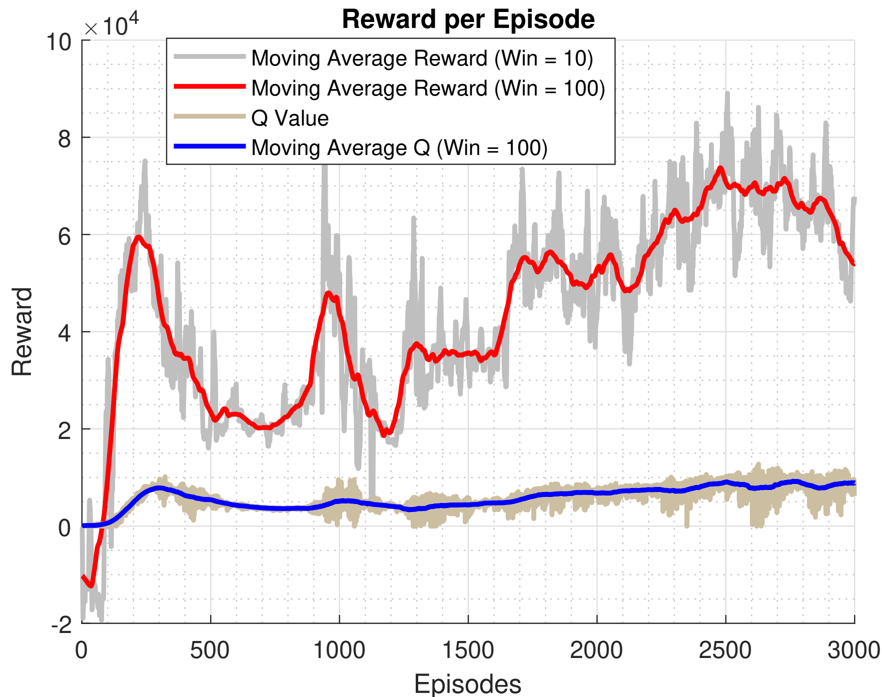


Figure 4.19: Cumulative Rewards During Training: Lateral Neural Network

This training duration was 58 hours and 37 minutes for all 10,000 episodes; however, the training could be stopped manually around episode 2,100 because the best neural network performance is achieved at training episode 2,037. The time taken for training up to episode 2,037 is approximately 16 hours and 32 minutes.

4.3.8 LTI Simulations - Lateral Neural Network Controller

The lateral neural network controller with the best performance (highest cumulative reward) is selected for evaluation using closed-loop LTI simulations with different initial conditions. The closed-loop simulations are carried out at 20 Hz update rate with discretized LTI Lateral model. A temporally correlated noise (Ornstein-Ulenbeck) is also implemented and applied to the lateral states at every 5 time steps (4 Hz or 0.25 seconds). One of the initial conditions used for the lateral

states is very difficult: $\beta = -4°, \phi = -20°, P = -30°/\text{s}, R = 15°/\text{s}$, the results of which are shown in Figures 4.20 and 4.21.

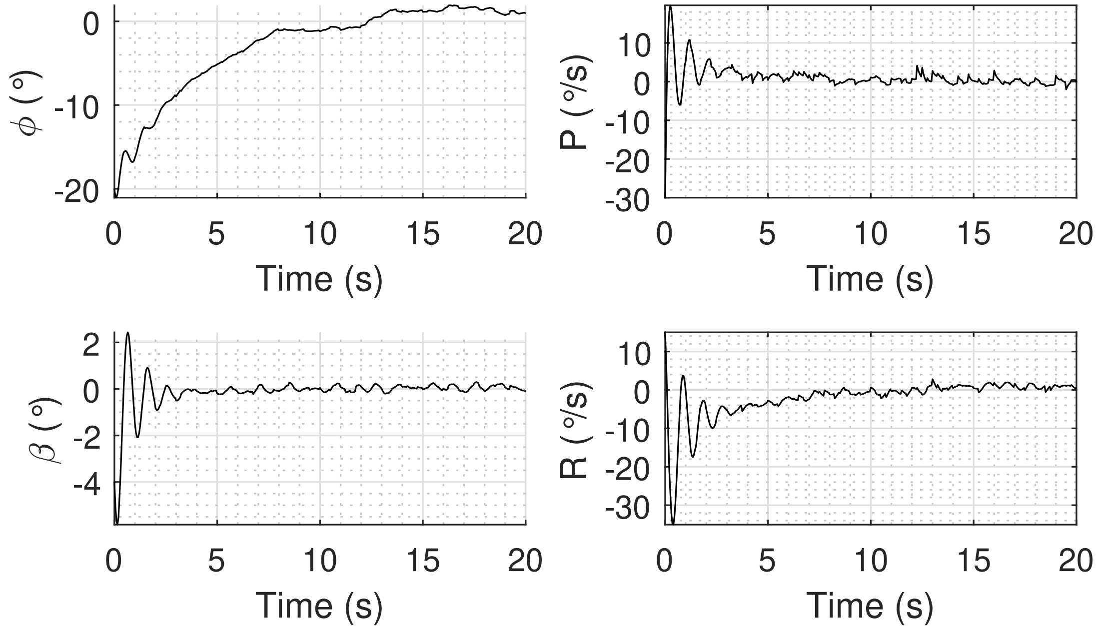


Figure 4.20: Lateral Neural Network Controller, Closed-Loop LTI Simulation Results: States

The mean for the noise used is zero, and the variances for the states are: $\sigma_\beta = 1°, \sigma_\phi = 2°, \sigma_P = 15°/\text{s}, \sigma_R = 10°/\text{s}$. The Lateral neural network controller predicts aileron controls based on perturbed roll angle input. From Figure 4.20, it can be clearly seen that the neural network is able to produce effective control outputs to drive the roll angle to zero degrees, while the roll and yaw rates are small and bounded. During the initial transient phase of state propagation in time, there are minor oscillations as seen in Figure 4.20.

As seen in Figure 4.21, the aileron output is very smooth, changing from about positive 0.8° and settling down to zero. The rate of change of aileron (control rate) is well bounded between $\pm 0.01°$ after the initialization phase.

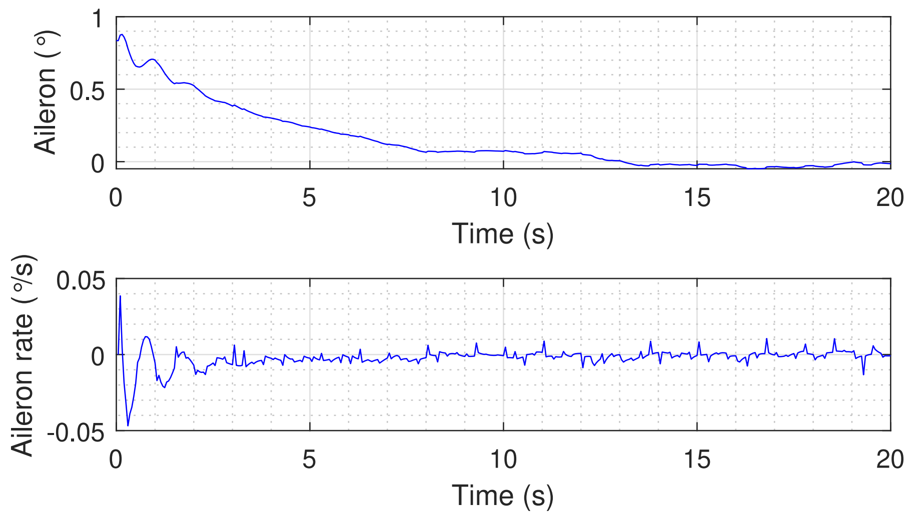


Figure 4.21: Lateral Neural Network Controller, Closed-Loop LTI Simulation Results: Control Variables and Rates

4.4 Flight Test Validation - Longitudinal Neural Network Controller

Flight tests were conducted at the Clinton International Model Airport in Lawrence, Kansas. Four waypoints are set by the ground station operator representing a racetrack flight path pattern. A total of 15 distinct flight tests were conducted with varying wind conditions (from low ideal to high extreme), different trim airspeeds, different altitudes, and most importantly, challenging test scenarios consisting of climb and descent maneuvers. Tests consisted of dynamically changing the airspeed command or trim velocity of the aircraft while the aircraft was performing autonomous flight controlled by the DDPG Longitudinal neural network. The climb and descent maneuvers were also tested dynamically while in autonomous flight. The flight test software incorporates a safety-critical switch, implemented in the ROS framework that monitors the neural network control outputs in real time. It outputs a stable or unstable flag that either allows the neural network to be in control or switches to command tracking LQR Longitudinal controller, respectively. The complete system level software architecture implemented onboard the aircraft is represented in a block diagram in Figure 4.22.

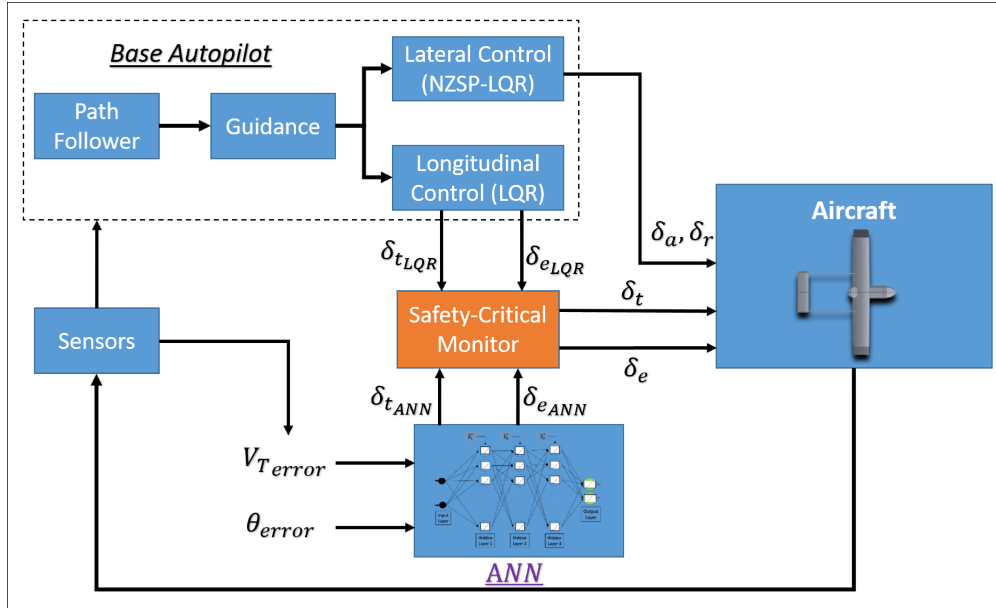


Figure 4.22: Aircraft Software-System Architecture: Flight Test

All flight test scenarios can be divided into 21 distinct categories or flight conditions. Most of the flights consisted of an anticlockwise (ACW) North-East pattern around the waypoints, with one test being clockwise (CW). The test scenarios with different aircraft conditions and complete statistics are detailed in Table 4.10. A total flight time added over all the flight tests was recorded at about 1 hour and 8 minutes. A total distance of about 40 miles was flown by the aircraft with altitude ranging from 328 ft to about 500 ft - Above Ground Level (AGL). The flight tests were conducted on different days spread out over 5 months (November 2019 to March 2020). Therefore flight tests encountered a large range of weather conditions: wind and temperature that in turn affect air density which directly affects the dynamic pressure over the aircraft and hence its dynamics characteristics.

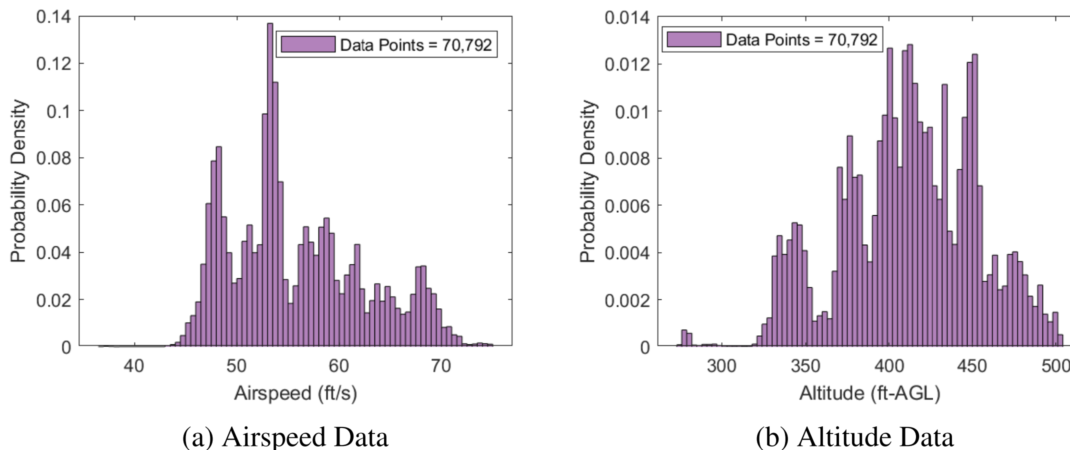
All the flights conducted using the DDPG longitudinal neural network exhibited stable behavior and desired performance characteristics. There was no need for re-training or tuning the neural network for any flight condition or scenario, and it performed equivalently well in each case, even though it was trained on Skyhunter's linear time invariant model.

The trim speed of the Skyhunter aircraft is 50.63 ft/s, and according to standard system dy-

Table 4.10: General Flight Test Statistics: DDPG Longitudinal Neural Network

Flight Scenarios	21
Flight Time	1 hr, 8 m, 48 s
Number of Flights	15
Total Flight Distance	214,114 ft (40.5 miles)
Race-Track Loops [1]	42.65
Wind Range (MPH)	0 - 13.9 (Gust to 17.4)
Altitude Range (ft-AGL)	328 - 500
Airspeed Range (ft/s)	41 - 78

namics and control theory, the controllers perform well as long as the perturbation around the trim points are small. The DDPG longitudinal neural network was flight tested with "large" perturbations and variations in flight airspeed, ranging from 41 ft/s to about 78 ft/s. The data distribution for airspeed collected over all 15 flight tests and 70,792 data points, is shown in Figure 4.23a. The longitudinal neural network was also flight tested at different altitudes, the data distribution of which is shown in Figure 4.23b.



(a) Airspeed Data

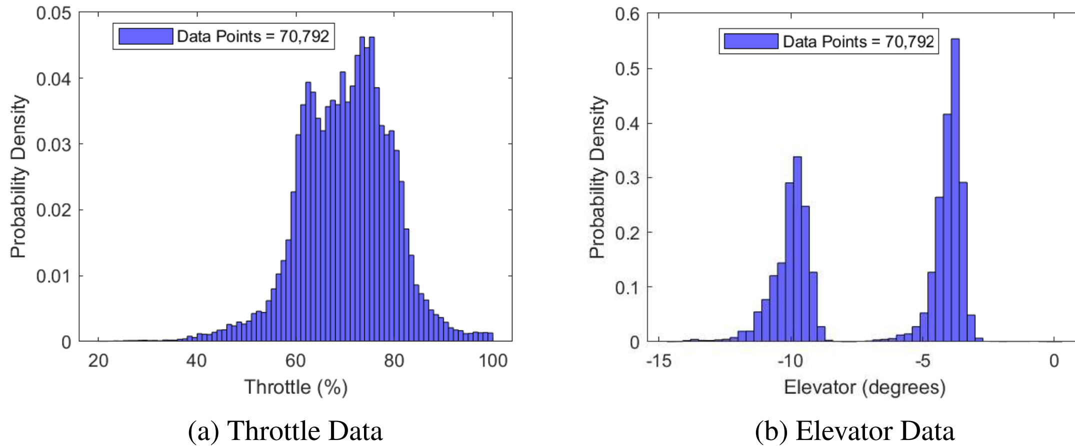
(b) Altitude Data

Figure 4.23: Airspeed and Altitude Data Distributions for all Flights with DDPG Trained Neural Network Longitudinal Controller

Figures 4.24a and 4.24b show the variations of control outputs generated by the neural network controller across all flight tests, for throttle and elevator respectively. The throttle trim for the Skyhunter aircraft is around 50% $\sim$ 60% and the throttle outputs in flight tests were changed from

[1] One Race-Track Loop $\approx$ 5020 ft

about 40% to 100% in different flight tests showing large perturbations. The data distribution for elevator controls, Figure 4.24b, likely shows two distinct elevator trim points, indicating that the aircraft mass distribution and the overall weight was changed across different flights.



(a) Throttle Data

(b) Elevator Data

Figure 4.24: Throttle and Elevator Controls Data Distributions for all Flights with DDPG Trained Neural Network Longitudinal Controller

The flight test scenarios are mainly divided into three sub-categories: (1) flight tests at different altitudes, with different airspeed commands and varying wind and gust condition (see Table 4.11), (2) flight tests with specific missions in which the airspeed command was changed dynamically from the ground control station (see Table 4.12), and (3) flight tests with altitude climb and descent missions in which the desired altitude is changed dynamically (see Table 4.13).

Flight scenarios in Table 4.11 are arranged in the increasing order of wind speed. From Table 4.11, the flight test results for test scenario 1.9 (marked with an asterisk) that showcase worst possible wind conditions, are presented in Section 4.4.1 with lateral and longitudinal states and controls, and 2D trajectory tracking.

Flight scenarios in Table 4.12 are arranged in the increasing order of altitude. From Table 4.12, the results from flight test scenario 2.5 (marked with an asterisk) are presented in Section 4.4.2. These results showcase DDPG longitudinal neural network flight controlling airspeed commands which are dynamically changing from the ground control station.

Flight scenario 2.1 (marked with a pound sign) is a special case of airspeed commands mission.

Table 4.11: Flight Test Scenarios with Varying Altitude, Airspeed and Wind Conditions: DDPG Longitudinal Neural Network

F.S.	Wind (MPH)	Alt. (ft-AGL)	$V_{a_{cmd}}$ (ft/s)	$\delta_{e_{trim}}$ (°)	F.T. (s)
1.1	$\approx$ 0	370	62	-9.2	12
1.2	5.6 $\rightarrow$ 7.4	328	58	-9.8	44
1.3	5.8 $\rightarrow$ 9.8	347	61	-9.4	100
1.4	6.7 $\rightarrow$ 7.4	360	73	-10.0	40
1.5	7 $\rightarrow$ 7	411	54	-1.3	272
1.6	9.2 $\rightarrow$ 12.3	380	45	-5	129
1.7	9.8 $\rightarrow$ 12.3	460	78	-10.4	123
1.8	12 $\rightarrow$ 15	428	69	-9.3	216
1.9*	13.9 $\rightarrow$ 17.4	370	65	-9.2	60

Table 4.12: Flight Test Scenarios with Airspeed Command Maneuvers: DDPG Longitudinal Neural Network

F.S.	Wind (MPH)	Alt. (ft-AGL)	$V_{a_{cmd}}$ (ft/s)	$\delta_{e_{trim}}$ (°)	F.T. (s)
2.1 #	8.1 $\rightarrow$ 9.8	380	50:55:50:45	-3.8	287
2.2	9.2 $\rightarrow$ 9.8	400	46:51:46:41	-4	373
2.3	4 $\rightarrow$ 5	400	52:48:45	-5	150
2.4	4.9 $\rightarrow$ 4.9	400	52:57:52	-10.0	182
2.5*	7.2 $\rightarrow$ 9.8	450	63:68:60	-9.6	236
2.6	6 $\rightarrow$ 10	466	55:60:52	-4	655

This flight test had airspeed commands programmed to change automatically in-flight based on the inertial location of the aircraft. This was done in order to compare LQR controller with the DDPG trained longitudinal neural network controller under almost similar flight conditions. In the same flight, while LQR controller is active, the mission is deployed from the ground control station and the mission program automatically commands different airspeeds at pre-defined inertial locations. After this mission with LQR controller is completed, the longitudinal neural network controller is activated and the mission is deployed again. This way the airspeed command maneuvers are performed under approximately same conditions for both LQR and the DDPG neural network controllers. The results of this flight test scenario are presented, discussed and analyzed in detail in Section 4.4.4.

Flight scenarios in Table 4.13 are arranged in the increasing order of airspeed. The flight test results from flight scenario 3.6 (marked with an asterisk) are presented in detail in Section 4.4.3.

Table 4.13: Flight Test Scenarios with Altitude Command Maneuvers: DDPG Longitudinal Neural Network

F.S.	Wind (MPH)	Alt. (ft-AGL)	$V_{a_{cmd}}$ (ft/s)	$\delta_{e_{trim}}$ (°)	F.T. (s)
3.1#	11.4 → 12.3	380:330:380	45	-4.2	175
3.2	4 → 5	400:450	45	-5	254
3.3	13.9 → 17.4	400:330:400	46	-4.5	174
3.4	4.9 → 4.9	415:350:400	52	-10.0	299
3.5	4 → 5	450:400	52	-5	296
3.6*	8.1 → 9.8	450:500:450	60	-9.6	111

Flight scenario 3.1 (marked with a pound sign) is a specifically designed flight mission in which altitude descend and climb maneuvers are tested while changing the desired altitude commands automatically based on the aircraft's inertial position. The flight mission is performed with both LQR controller and the DDPG trained neural network controller in approximately matching flight conditions. The detailed analysis of the results from this flight test scenario are presented in Section 4.4.4.

A total of five different flight test scenarios are presented in detail:

1. Scenario 1.9: This flight test was conducted in extremely high wind conditions with wind speed of 13.9 miles per hour (MPH) or $\approx$ 20 ft/s. Since the trim speed for Skyhunter aircraft is 50.63 ft/s and its stall speed is around 30 ft/s, these wind conditions present very difficult conditions for the flight controller in terms of maintaining stability throughout the autonomous flight. In this scenario, the aircraft was commanded to fly at 65 ft/s which is a large perturbation from the actual trim speed.

2. Scenario number 2.5: In this test, initially the desired airspeed was set at 63 ft/s. During the flight, the airspeed command was changed dynamically by sending commands through the ground control station via wireless telemetry. The aircraft was able to adjust the airspeed accordingly while maintaining stability and altitude.

3. Scenario number 3.6: This test scenario consisted of changing the reference altitude dynamically through the ground control station. Initially, the autopilot was engaged at 450 ft altitude

above ground level (AGL) and then changed to 500 ft and back to 450 ft via ground control station.

4. Scenario 2.1: This flight test scenario is an airspeed command variation test for performance comparison of LQR and DDPG trained neural network controller.

5. Scenario 3.1: This flight test scenario is an altitude command variation test for performance comparison of LQR and DDPG trained neural network controller.

In all the flight test scenarios, a pilot takes off the aircraft from ground using a remote control (RC) and continues to control it through climb phase, and once at the desired altitude and in a trimmed, steady-state level-wing flight condition, the base autopilot is engaged. Using onboard sensor information and desired waypoints, the aircraft flies around the waypoint path in a racetrack pattern. Once the autonomous flight has completed a few racetrack loops around the waypoints, the DDPG trained longitudinal neural network controller is engaged by sending an activation flag from the ground control station. For each flight test, the switching takes place in a smooth manner, and the aircraft throttle and elevator are commanded by the neural network.

4.4.1 Flight Scenario 1.9: Extreme Wind Conditions

This test was a high risk flight due to high steady wind conditions with a speed recorded at 13.9 MPH and a gust to 17.4 MPH. The pilot monitored the aircraft continuously in his line of sight, in case of any catastrophic failure or aircraft becoming unstable.

Longitudinal states for this flight test scenario are shown in Figure 4.25. The green portion of the parameter plots represent autonomous flight and the magenta color portion represents when the ANN Longitudinal controller was commanding throttle and elevator variables. The blue portions represent the desired or commanded variables. The airspeed was trimmed around 65 ft/s and the reference or desired altitude was around 367 ft. The ANN flew the aircraft for about 60 seconds before the switching monitor predicted an unstable flag, thereby automatically switching control to

LQR controller. As can be seen from the state plots, Figures 4.25 and 4.26, the aircraft was stable and was able to maintain autonomous path tracking.

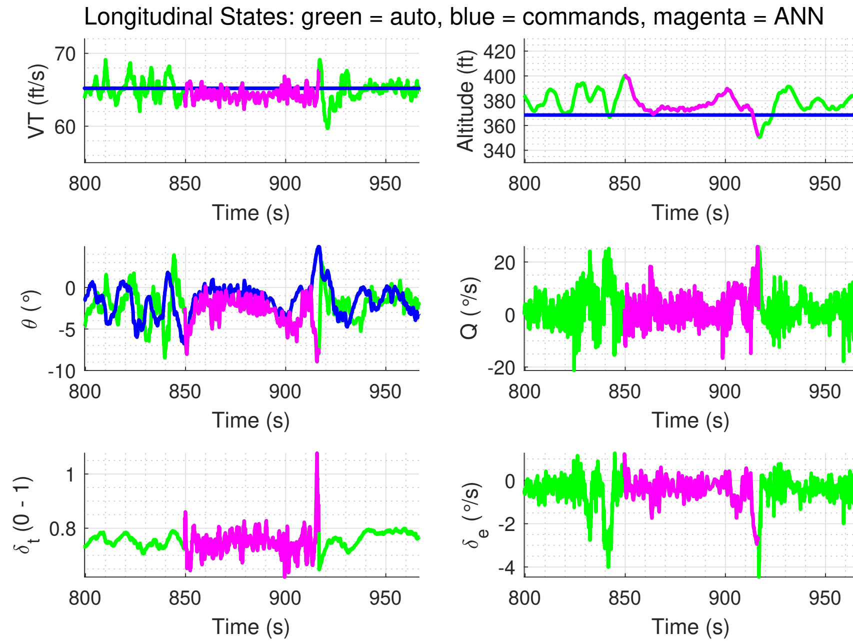


Figure 4.25: Flight Scenario 1.9: Longitudinal States for Flight Test Under Extreme Wind Condition

Note that from the Longitudinal state plots, Figure 4.25, it can be observed that the airspeed and pitch command tracking are better when Longitudinal ANN controller was active. And as a result, we see a better altitude hold, within ± 5 ft at least for the straight line flight. During the straight line flight for ANN controller, the airspeed hold is within ± 2.8 ft/s whereas when base autopilot is active, the airspeed hold is ± 5 ft/s. The pitch command tracking when the base autopilot is active is adequate but there is a considerable and variable delay of about 2 to 4 seconds. But the pitch tracking for the ANN controller has negligible delays and magnitude errors during straight line flight.

The 2-dimensional (2D) trajectory tracking for this flight is shown in Figure 4.27. The aircraft was able to complete almost one full loop around the waypoint path. The lateral tracking for when the ANN controller is active is very similar to that of base autopilot, indicating that the Longitudinal ANN controller was stable enough so as not to impart coupled lateral state variations.

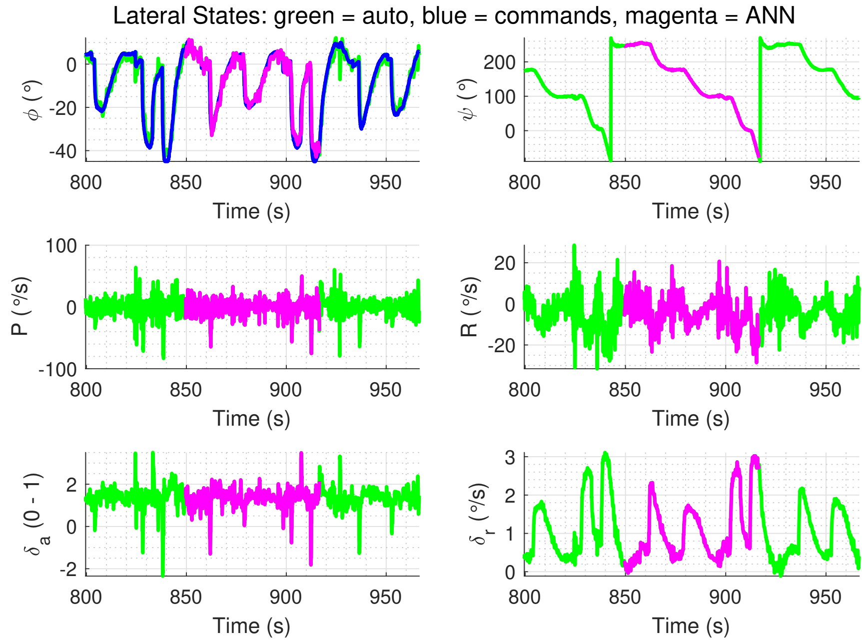


Figure 4.26: Flight Scenario 1.9: Lateral States for Flight Test Under Extreme Wind Condition

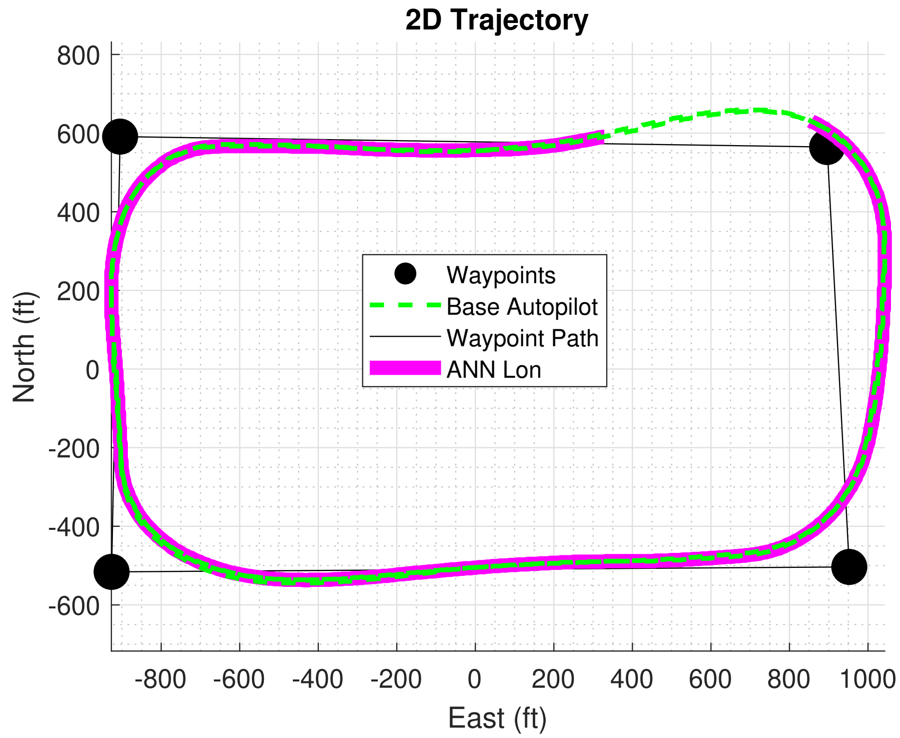


Figure 4.27: Flight Scenario 1.9: 2D Trajectory for Flight Test Under Extreme Wind Condition

4.4.2 Flight Scenario 2.5: Airspeed Command

In this test scenario, large perturbations from the trimmed airspeed are introduced as inputs to the Longitudinal neural network to test its generalization and robustness characteristics. The aircraft is flown typically starting from base autopilot and the ANN is activated mid-flight through the ground control station. Initially the airspeed hold commands are set at approximately 63 ft/s for both base autopilot and the ANN controller. After flying for about 100 seconds (≈ one loop), the airspeed command is dynamically changed from 63 ft/s to 68 ft/s.

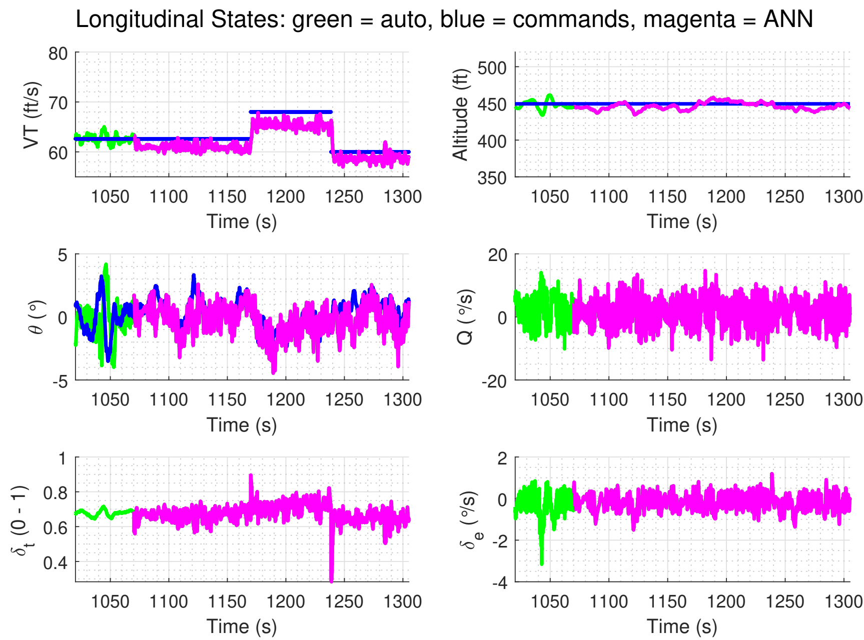


Figure 4.28: Flight Scenario 2.5: Longitudinal States for Flight Test with Different Airspeed Commands

The Longitudinal states for this test scenario are shown in Figure 4.28, in which it can be observed that the first transition from airspeed of 63 ft/s to 68 ft/s takes place in about 3 seconds and this airspeed hold (68 ft/s) persists for about 68 seconds, before the command is changed again. During this period (time: 1171 to 1239 seconds), while the airspeed is commanded from 63 ft/s to 68 ft/s, the altitude hold is kept almost a constant by adjusting the elevator controls via ANN Longitudinal controller. The ANN controller is capable of generalizing this complex coupling

between throttle and elevator tied to both the states: airspeed and pitch angle, and hence only the airspeed is changed while the altitude is maintained.

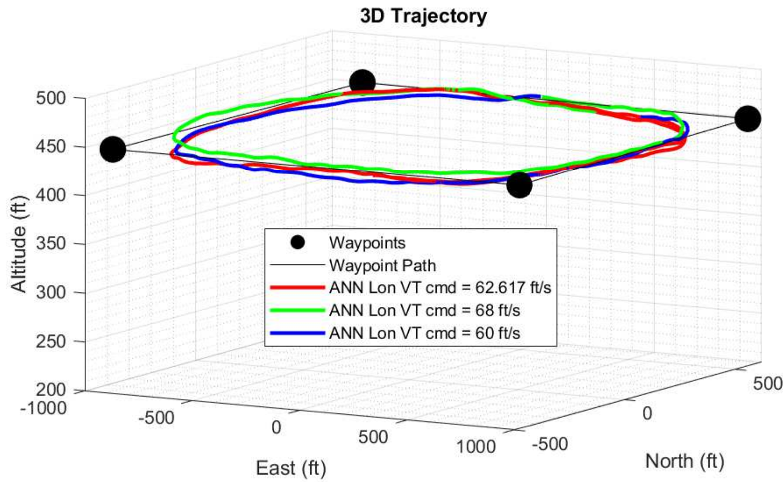


Figure 4.29: Flight Scenario 2.5: Three dimensional trajectory - three different airspeed commands

Figure 4.29 shows the 3D trajectory of flight path when the longitudinal neural network controller was active and the airspeed commands were dynamically changed from the ground control station. The different airspeed commands are represented in colors red (62.617 ft/s), green (68 ft/s), and blue (60 ft/s). During all the airspeed transitions, altitude was maintained within $\pm$ 15 ft.

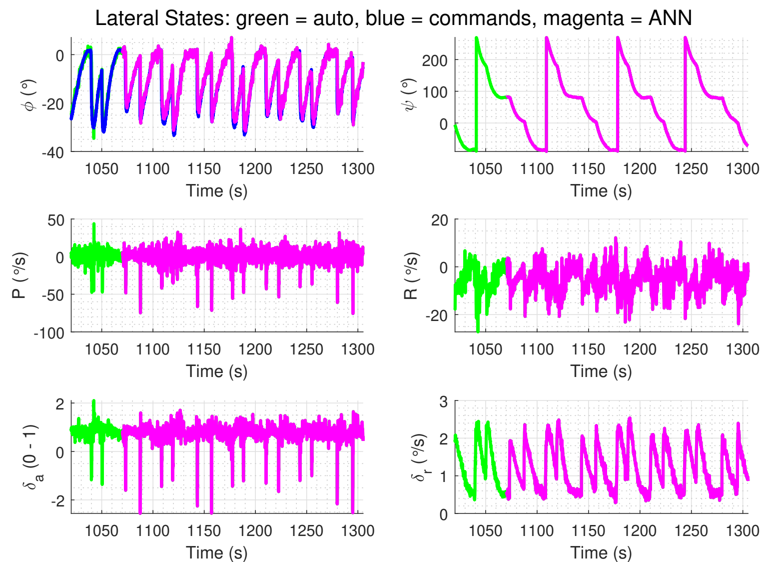


Figure 4.30: Flight Scenario 2.5: Lateral States for Flight Test with Different Airspeed Commands

The second airspeed command from 68 ft/s to 60 ft/s is sent to the aircraft at around time 1239 seconds. Again the transition takes place in about 2 seconds without causing any instability to any of the aircraft states. The altitude is maintained while the airspeed is adjusted by the ANN controller. The lateral states for this test scenario are shown in Figure 4.30. The 2D trajectory for this test scenario is shown in Figure 4.31.

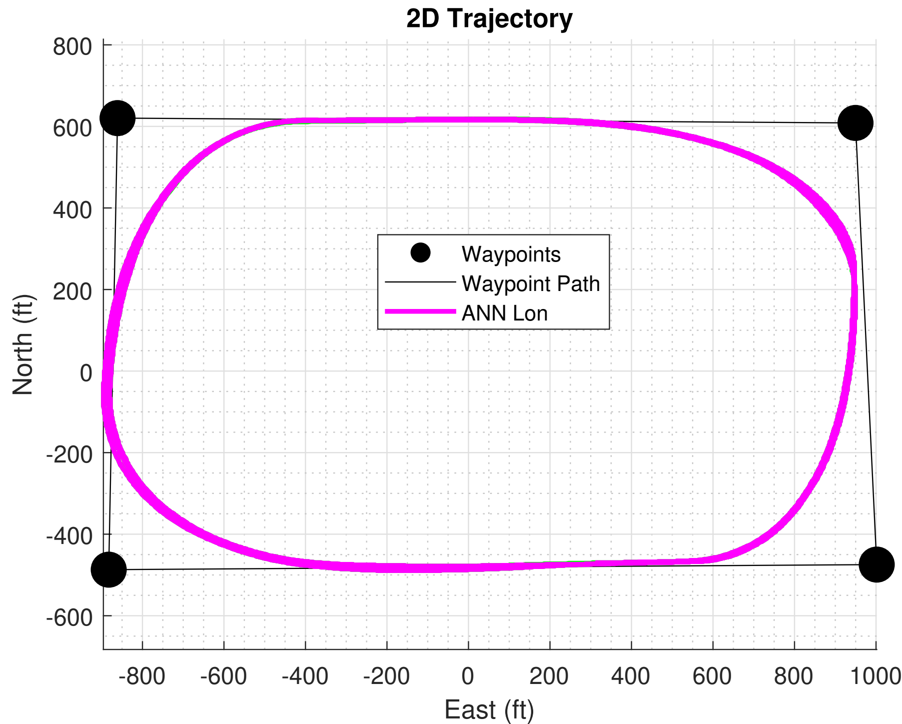


Figure 4.31: Flight Scenario 2.5: 2D Trajectory for Flight Test with Different Airspeed Commands

4.4.3 Flight Scenario 3.6: Altitude Command

In this test scenario, the reference or desired altitude is changed dynamically from the ground control station. The goal of this test is to introduce large perturbations from the trimmed pitch angle as inputs to the longitudinal neural network and test its generalization and robustness characteristics. Initially the desired altitude hold commands are at approximately 450 ft above ground level for the ANN controller flight. The altitude command is dynamically changed from 450 to 500 ft at about time 1306 seconds as shown in Figure 4.32. The longitudinal states for this test scenario are shown in Figure 4.32, in which it can be observed that the first transition from altitude of 450 ft to 500 ft

takes place in about 12 seconds and this altitude hold (500 ft) persists for about 72 seconds, before the reference altitude is changed again.

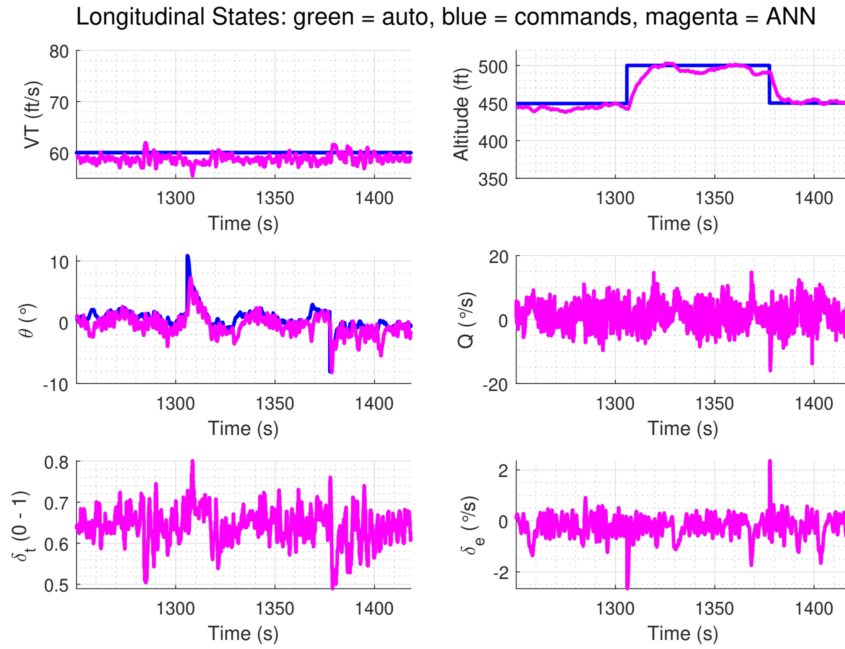


Figure 4.32: Flight Scenario 3.6: Longitudinal States for Flight Test with Different Altitude Commands

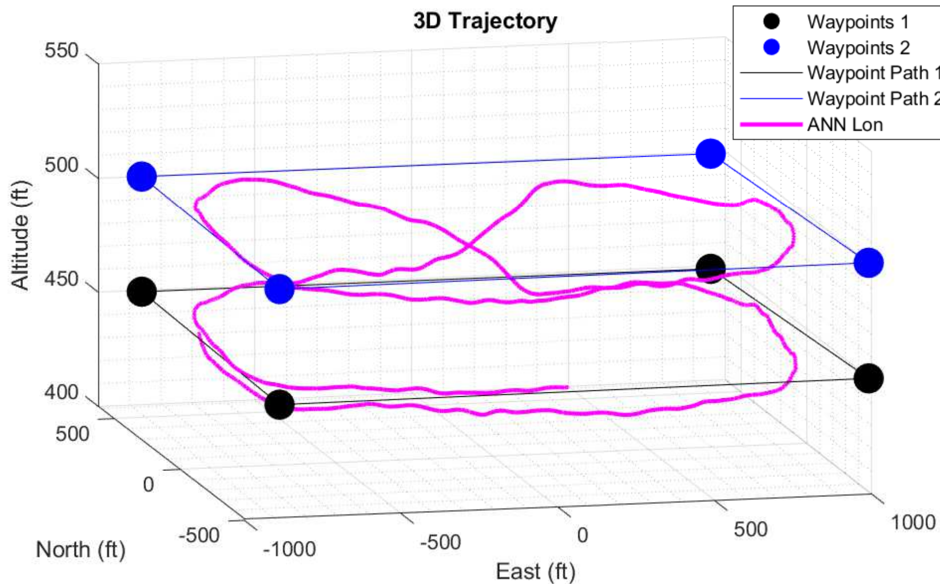


Figure 4.33: Flight Scenario 3.6: Three dimensional trajectory - two different altitude commands

The 3D trajectory for the flight path showing altitude transitions is depicted in Figure 4.33, first the aircraft climbs to 500 ft and then descends back to 450 ft. During this period (time: 1306 to 1378 seconds), while the altitude is commanded from 450 ft to 500 ft, the airspeed hold is kept almost a constant by adjusting the throttle controls via ANN Longitudinal controller. This again shows that the ANN controller is capable of generalizing the complex coupling between throttle and elevator tied to both the states: airspeed and pitch angle, and hence only the altitude is changed while the airspeed is maintained.

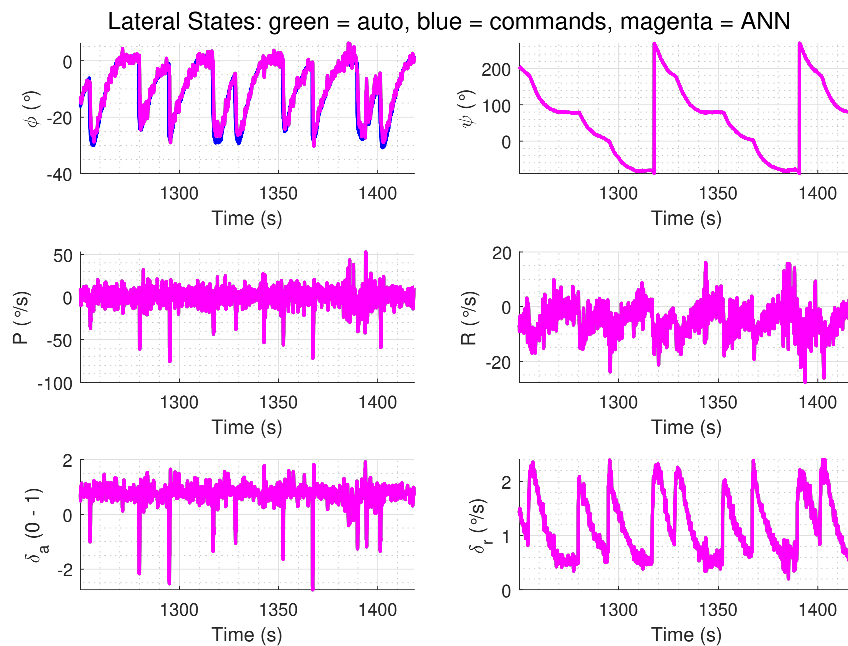


Figure 4.34: Flight Scenario 3.6: Lateral States for Flight Test with Different Altitude Commands

The second altitude command from 500 ft to 450 ft is sent to the aircraft at around time 1378 seconds. Again the transition takes place in about 7 seconds without causing any instability to any of the aircraft states. The airspeed is maintained while the altitude is adjusted by the ANN controller. The lateral states for this test scenario are shown in Figure 4.34.

4.4.4 Flight Scenario 2.1 and 3.1: Flight Test Comparison LQR vs ANN

These flight scenarios are automatic flight missions that are activated through telemetry commands via the ground control station. A flight mission is activated with a specific identification parameter and an "ON" flag, sent simultaneously via the ground control station. After take-off, base autopilot engagement and achieving steady racetrack flight, each of these missions are activated manually when the aircraft is around the mid-point of the South waypoint leg, see the top left corner plot in Figure 4.35. The approximate location of mission activation is around North: -500 ft and East: 0 ft. After activating the mission, the mission does not start (deploy) until it reaches a point on the East waypoint leg, where the switching algorithm automatically switches to the next waypoint line which is the North waypoint leg. The idea behind this logic is that the mission is deployed automatically based on an approximate inertial location of the aircraft (North-East corner) which repeats itself if the aircraft speed is maintained. This way comparison tests can be conducted between different controllers within the same flight test, approximately 5 minutes apart, so that the wind conditions do not change significantly.

Table 4.14: Flight Test Statistics: DDPG Longitudinal Neural Network vs LQR Base Autopilot

F.S. 2.1	S.S. $V_{a_{err}}$	Total $V_{a_{err}}$	S.S. Alt_{err}	Total Alt_{err}	Transient Time
LQR: $V_{a_{cmds}}$	1.49 ft/s	1.80 ft/s	11.87 ft	11.76 ft	1.4 s
ANN: $V_{a_{cmds}}$	1.39 ft/s	1.53 ft/s	13.09 ft	13.24 ft	1.4 s
F.S. 3.1	S.S. $V_{a_{err}}$	Total $V_{a_{err}}$	S.S. Alt_{err}	Total Alt_{err}	Transient Time
LQR: Alt_{cmds}	0.76 ft/s	0.88 ft/s	11.34 ft	17.20 ft	9 s
ANN: Alt_{cmds}	2.73 ft/s	2.81 ft/s	9.83 ft	15.67 ft	9 s

Two different missions are designed for comparing the performance of LQR and the DDPG trained longitudinal controller. One mission changes the airspeed commands automatically and the other changes the altitude commands. The summary of results for these two scenarios is shown in Table 4.14. The top part, first three rows show the results for flight scenario 2.1 (airspeed

commands) and the bottom three rows show the results for flight scenario 3.1. The columns of the table represent steady-state airspeed errors, overall airspeed errors, steady-state altitude errors, overall altitude errors and the transient time chosen for transition from non-steady to a steady-state. All the values are the average over the whole flight mission. In both the cases, it can be seen that the performance of LQR and the neural network controllers are quite similar. For the flight scenario 2.1, airspeed commands, the neural network's average steady-state airspeed error is 1.39 ft/s and that of LQR is 1.49 ft/s. For flight scenario 3.1, altitude commands, the average steady-state altitude error for the neural network is 9.83 ft and that of LQR is 11.34 ft.

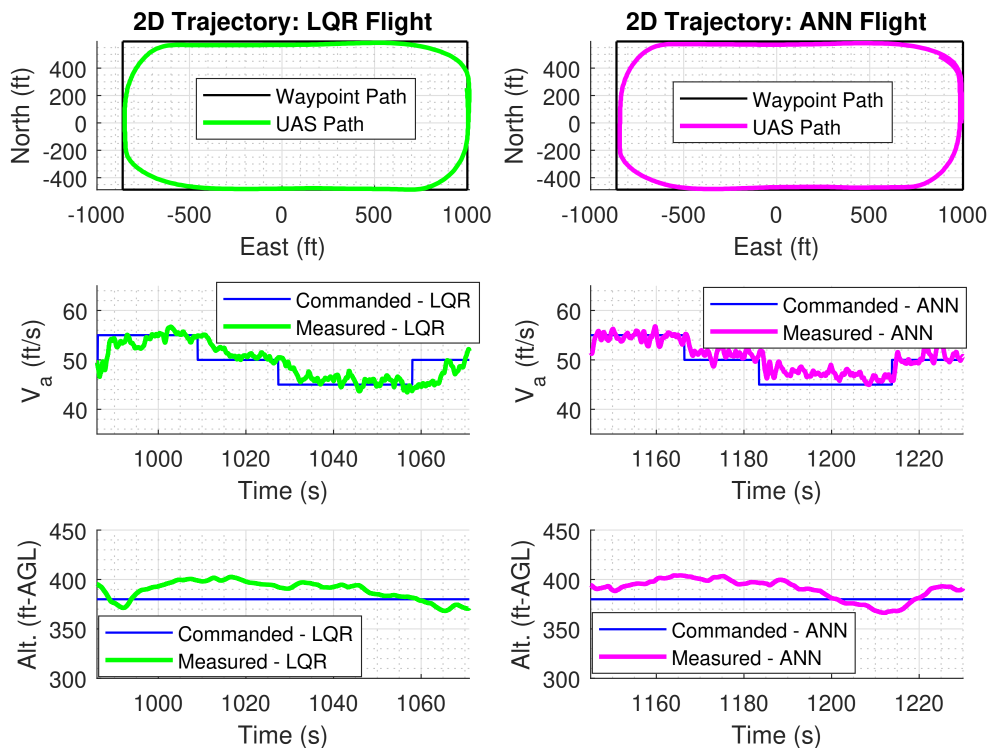


Figure 4.35: Airspeed Command, Flight Scenario 2.1: 2D Trajectory, Airspeed Commanded and Measured Values, and Altitude Commanded and Measured Values

Flight Scenario 2.1: The results from this scenario are shown in Figure 4.35 with green color used for LQR based flight and magenta color used for ANN based flight. When the mission is deployed close to the North-East corner, the airspeed command is set to 55 ft/s. As the aircraft progresses in its flight, the airspeed commands are changed three more times at each waypoint

corner. The airspeed commands are changed to 50 ft/s, then 45 ft/s and to 50 ft/s at the North-West, South-West and South-East corners respectively. The left column of Figure 4.35 shows the plots for LQR deployed mission and the right side column shows results for the neural network deployed mission. The 2D trajectory tracking in both cases is very similar. It can be seen that for airspeed commands of 55 ft/s the airspeed errors for the neural network flight are lower than for the LQR flight. The average airspeed error during 55 ft/s commands for the ANN flight is 0.78 ft/s, whereas for the LQR flight it is 1.15 ft/s. For both the cases, the altitude is maintained around the desired value of 380 ft-AGL within $\pm$23 ft.

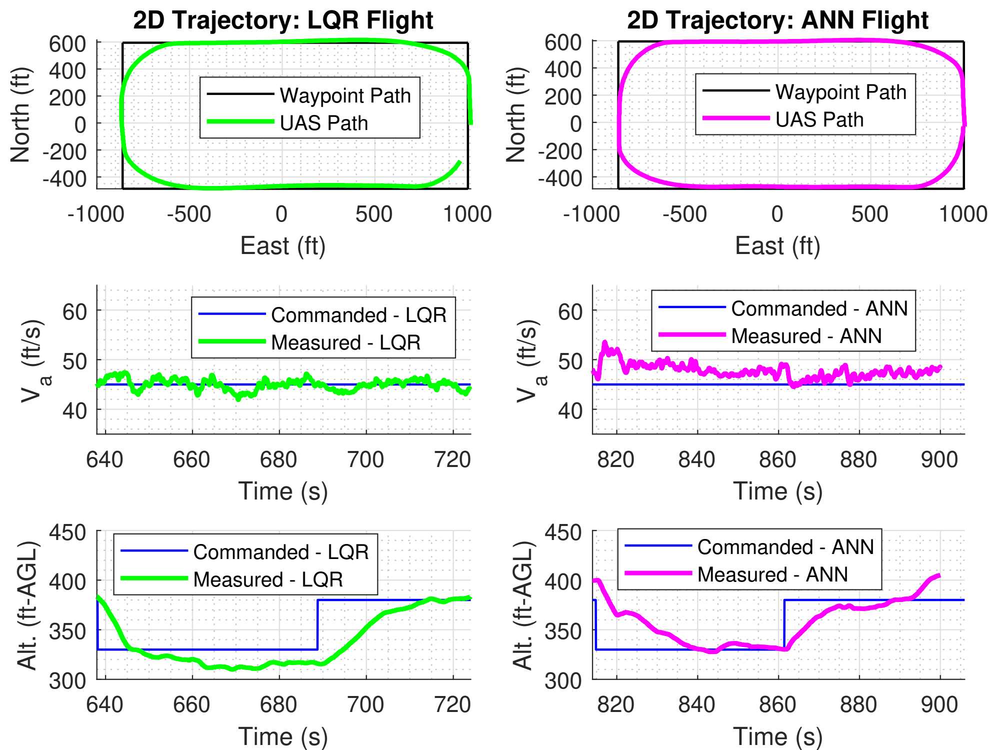
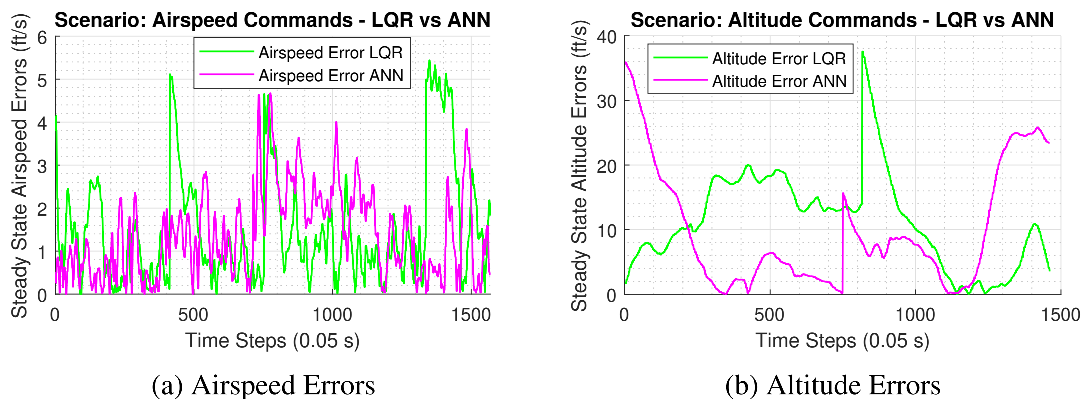


Figure 4.36: Altitude Command, Flight Scenario 3.1: 2D Trajectory, Altitude Commanded and Measured Values, and Airspeed Commanded and Measured Values

Flight Scenario 3.1: The results from this flight test scenario are shown in Figure 4.36. At the start of the mission around the North-East corner of the waypoint path, the altitude is set at 330 ft-AGL and then it is changed back to 380 ft-AGL at the South-West corner. For both cases, LQR and ANN the altitude tracking performance is similar, with average altitude errors for ANN at

steady-state as 9.83 ft as compared to 11.34 ft for the LQR flight. The overall and average airspeed errors for the LQR flight (S.S. 0.76 ft/s) are slightly better than the ANN (S.S. 2.73 ft/s). The 2D trajectory tracking performance is very similar in both the cases.

The steady-state airspeed errors plotted at 0.05 s of time steps for the airspeed command scenario are shown in Figure 4.37a, and the altitude errors for the altitude mission are plotted in Figure 4.37b. From the airspeed errors it can be seen that the performance of LQR and ANN were very similar in this flight scenario. And from the altitude errors plot, it can be seen that at most locations the ANN based flight outperforms the LQR based flight.



(a) Airspeed Errors

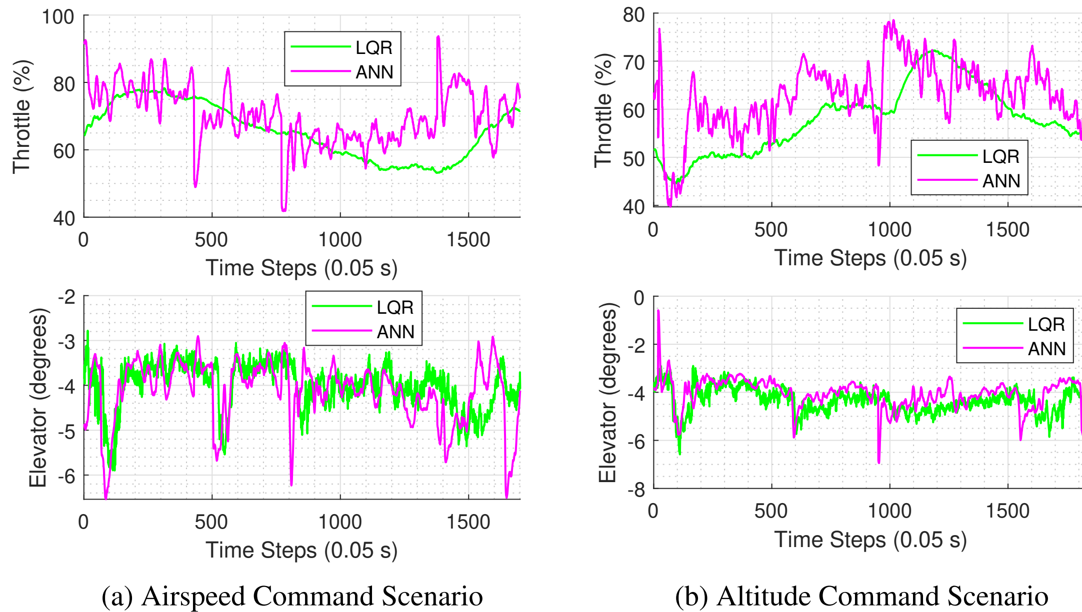
(b) Altitude Errors

Figure 4.37: Airspeed and Altitude Errors for Varying Airspeed and Altitude Commands, respectively

The throttle and elevator controls for both the missions 2.1 and 3.1 are shown in Figures 4.38a and 4.38b, respectively. The left column of these plots show the airspeed commands scenario and the right side shows the altitude commands case. In each mission, it is seen that the behavior of ANN is quite similar and the same can be said for the LQR flight. The ANN tends to use more throttle controls throughout the flight and varies it more as compared to LQR to achieve the same goals. Whereas, the LQR prefers to make more elevator variations as compared to ANN. This shows that the neural network controller has learned a novel control behavior in which it performs similarly to that of LQR but uses completely different combinations of throttle and elevator controls.

This type of novel behavior learned by the neural network controller cannot be extracted using

standard and existing control techniques. This novel result sets a series of new research questions about how many more possibilities of different types of controller behaviors can be adapted and explored using neural network learning methodologies. Moreover, with the possibility of tuning a neural network architecture in many different ways, the neural network is capable of learning a huge amount of coupled controllers generalizing over an ever increasing flight envelope.



(a) Airspeed Command Scenario

(b) Altitude Command Scenario

Figure 4.38: Throttle and Elevator Controls Comparison

4.5 Neural Network Evolution using Flight Test Data

The DDPG trained Longitudinal neural network controller is subjected to re-training using all the flight test data collected in the above flight tests, as explained in Section 3.7. This neural network was originally trained using aircraft LTI model, which is a perturbed model around a trim point as mentioned in Section 3.1. Therefore, the input states to the neural network are perturbed around an assumed trim point, consisting of the desired airspeed and desired pitch angle. The airspeed command is set initially as the measured airspeed at the time the autopilot is turned "ON" and the pitch attitude commands are outputted by the Longitudinal guidance algorithm as mentioned in Section 2.3. Hence, these input states to the neural network are taken as the difference between

the measured and the desired respective state. Similarly, the ANN outputs are taken as perturbed values, around the trim throttle and trim elevator controls during the flight. Inertial NED velocity from the GPS is also recorded to estimate (post-process) the flight path angle from the flight data.

Table 4.15: DDPG Longitudinal Neural Network Trained using Flight Data

Number of Experiences from Flight	67,393
Initial Conditions (LTI) for Validation (Random Selection) = 100	
V_T	$45 ft/s$ to $55 ft/s$
α	-5° to 5°
θ	-10° to 10°
Q	$-20^\circ/s$ to $20^\circ/s$
Training Episodes	1,149
Time Taken	10 hours, 57 minutes, 37.96 seconds
Average Validation Cumulative Reward Start	24,261.47
Average Validation Cumulative Reward End	51,980.79

The experience buffer (real flight data) should be saved in proper format before re-training the neural network. The desired format for the data is: (state, action, reward, next state, done), as explained in Section 2.9.4. Using the reward function as explained in Section 4.3.1, the immediate reward for each data point (state) is calculated. The done flag is set to zero for each next state data point, as the flight data collected always exhibits stable aircraft states.

The training statistics are shown in Table 4.15. The flight data, total number of experiences used is 67,393. After each training episode, the neural network is tested to control the aircraft longitudinal LTI model for 100 different initial conditions, randomly selected as a combination of V_T, α, θ, Q. The different initial conditions are uniformly sampled from the ranges shown in Table 4.15. The training is manually stopped at episode 1149, because the learning does not improve after this episode. The training duration is 10 hours and 57 minutes and the average cumulative reward for all 100 test cases at the start of first episode is 24,261.47, which reaches up to 51,980.79

Table 4.16: DDPG Longitudinal Neural Network Trained using Flight Data: Monte-Carlo Simulations

Monte-Carlo 6-DOF Simulations	10,000 each
Simulation Percentage with Higher Value	70.03%
Normalized Mean Cumulative Reward Difference (Evolved - Original)	0.0774%
Normalized Max Cumulative Reward Difference (Evolved - Original)	11.64 %
Normalized Original ANN: Mean Cumulative Reward	82.816 %
Normalized Evolved ANN: Mean Cumulative Reward	82.890 %

at the end of episode 1149. The cumulative rewards averaged over the 100 cases per episode are shown in Figure 4.39. It can be seen that the average cumulative reward is consistent across randomly selected 100 different test cases throughout the training, and increases steadily.

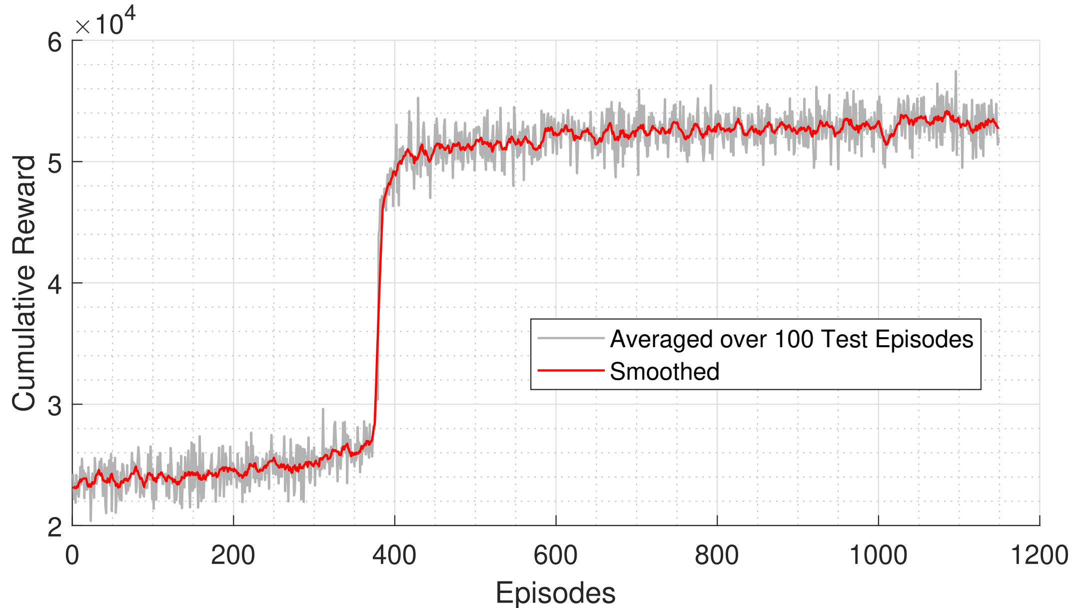


Figure 4.39: Cumulative Rewards Averaged Over 100 Test Episodes with Different Initial Conditions, During Re-Training of Longitudinal Neural Network from Flight Data

The re-trained or evolved neural network and the original flight tested neural network are subjected to 10,000 Monte-Carlo simulations (for details see Section 3.4). From all the 10,000 different conditions, it was found that the cumulative reward for the evolved ANN was higher in more than 7,000 cases of Monte-Carlo flight simulations. The normalized mean cumulative reward difference (Evolved - Original) improvement is relatively small at about 0.077%, and the maximum normalized cumulative reward difference is 11.64%.

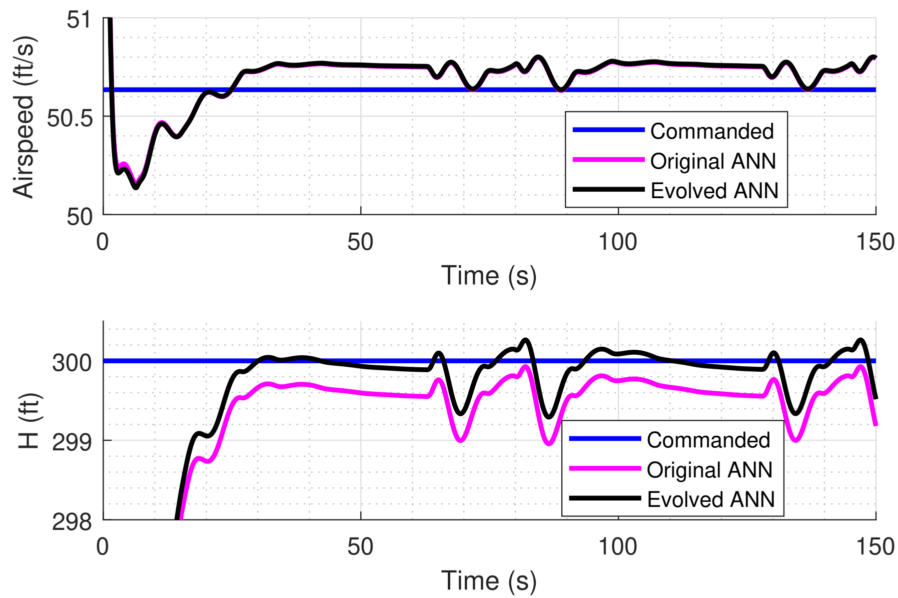


Figure 4.40: Airspeed and Altitude States Comparison for Original and Evolved Longitudinal Neural Network: 6-DOF Simulations

Airspeed and altitude tracking comparisons for the evolved and the original neural networks are shown in Figure 4.40, from one of the closed-loop 6-DOF flight simulations. From the airspeed plot it can be seen that the performance is very similar for both the evolved and the original ANNs. And, the altitude tracking has a very small improvement in the case of the evolved ANN.

Chapter 5

Conclusions

The research hypothesis initially focused on exploring whether an artificial neural network would be able to behave as a standalone flight controller and learn to fly while maintaining stability. A novel imitation learning methodology called moving window data aggregation was developed and used to train an artificial neural network to mimic a complete autopilot system consisting of guidance, navigation, and control algorithms. The moving window data aggregation was shown to be robust for different imitation learning models and “expert” policies. This algorithm successfully trained decoupled neural network controllers that were able to pass 10,000 Monte-Carlo flight test simulations. Due to lack of robustness of the LQR based autopilot, as seen in initial flights, the decoupled neural networks were inherently dependent on the “expert” policy and could not generalize and perform in real flight tests.

This work has demonstrated the potential of using artificial neural networks as flight controllers that can generalize across a wide range of flight envelope. Multiple validation and verification flight tests were conducted on the DDPG trained longitudinal neural network flight controller, showcasing its generalizing capabilities on a highly nonlinear Skyhunter aircraft and a broad range of flight conditions (wind, gust, altitude and airspeed). A total of 15 **successful** flight tests with 21 distinct scenarios were conducted over a period of 5 months (November 2019 to March 2020), with different goals and flight missions, proving the performance validity of the neural network flight controller. It was shown that the initial design of a standard LQR based controller did not perform well in a real flight test, even after passing 99.96% (99,960 out of 100,000) Monte-Carlo flight test simulations, and which had to be tuned in-flight to conduct a stable flight test. Both the LQR and the neural network controllers used the same linear time invariant model (LTI) of the Skyhunter

aircraft to find the control gains and train, respectively. Both controllers showed excellent performance in LTI and 6-DOF closed-loop simulations; however, only the neural network was able to perform a stable and robust flight in its first test using the real Skyhunter aircraft.

Through real flight tests, it is shown that the "dependency" of a standard flight controller design on a physics-based aircraft model can be mitigated through the use of artificial neural networks. The neural networks can be directly used as controllers, and trained to generalize a "policy" that breaks feedback correlations through the use of reinforcement learning. Moreover, a practical comparison of LQR and neural network controllers over the same flight missions, shows a significantly different control behavior. For performing the same types of maneuvers, the neural network chooses completely different combinations of throttle and elevator controls, indicating the potential of further learning and numerous possibilities for applying neural networks for flight controls.

This research produced a methodology that enables a neural network controller to learn from real flight data and evolve to perform better in a new flight condition. Training a neural network controller directly from flight data eliminates the requirement of developing a costly physics-based aircraft dynamic model. The re-trained neural network flight controller demonstrates better performance than its counterpart in more than 70% of flight test simulations (7,003 out of 10,000), showing signs of evolution and a prospect for outperforming and offering a competitive alternate to standard robust and optimal controllers.

References

[1] Csail's nick roy helms google's delivery-drone project. http://news.mit.edu/2014/csail-nick-roy-google-delivery-drone, 2014.

[2] Uavs learn to fly solo. http://news.mit.edu/2015/uavs-operate-autonomously-0824, 2015.

[3] Qgroundcontrol (qgc). qgroundcontrol.com/, 2017.

[4] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[5] Pieter Abbeel, Adam Coates, Timothy Hunter, and Andrew Y Ng. Autonomous autorotation of an rc helicopter. In *Experimental Robotics*, pages 385–394. Springer, 2009.

[6] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[7] Ahmed Abdelaziz, Mohamed Elhoseny, Ahmed S Salama, and AM Riad. A machine learning model for improving healthcare services on cloud computing environment. *Measurement*, 119:117–128, 2018.

[8] Manjula Ambur, Katherine G Schwartz, and Dimitri N Mavris. Machine learning technologies and their applications for science and engineering domains workshop–summary report. 2016.

[9] Manjula Y Ambur, Jeremy J Yagle, William Reith, and Edward McLarney. Big data analytics and machine intelligence capability development at nasa langley research center: Strategy, roadmap, and progress. 2016.

[10] Olov Andersson, Mariusz Wzorek, and Patrick Doherty. Deep learning quadcopter control via risk-aware active learning. In *AAAI*, pages 3812–3818, 2017.

[11] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[12] Okan Aşık, Binnur Görer, and H Levent Akın. End-to-end deep imitation learning: Robot soccer case study. *arXiv preprint arXiv:1807.09205*, 2018.

[13] J Andrew Bagnell and Jeff G Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1615–1620. IEEE, 2001.

[14] SN Balakrishnan and Victor Biega. Adaptive-critic-based neural networks for aircraft optimal control. *Journal of Guidance, Control, and Dynamics*, 19(4):893–898, 1996.

[15] SN Balakrishnan and RD Weil. Neurocontrol: A literature survey. *Mathematical and Computer Modelling*, 23(1-2):101–117, 1996.

[16] Gary J Balas, Ian Fialho, Andy Packard, Joe Renfrow, and Chris Mullaney. On the design of lpv controllers for the f-14 aircraft lateral-directional axis during powered approach. In *Proceedings of the 1997 American Control Conference (Cat. No. 97CH36041)*, volume 1, pages 123–127. IEEE, 1997.

[17] Haitham Baomar and Peter J Bentley. An intelligent autopilot system that learns piloting skills from human pilots by imitation. In *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, pages 1023–1031. IEEE, 2016.

[18] Haitham Baomar and Peter J Bentley. Autonomous navigation and landing of large jets using artificial neural networks and learning by imitation. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–10. IEEE, 2017.

[19] Ethan Baumann, Catherine Bahm, Brian Strovers, Roger Beck, and Michael Richard. The x-43a six degree of freedom monte carlo analysis. In *46th AIAA Aerospace Sciences Meeting and Exhibit*, page 203, 2008.

[20] A Aziz Bhatti. Neural network based control system design of an advanced fighter aircraft. In *Telesystems Conference, 1991. Proceedings. Vol. 1., NTC'91., National*, pages 1–6. IEEE, 1991.

[21] Aaron T Blevins, Hady Benyamen, Grant Godfrey, Daksh Shukla, and Bella Kim. Analysis and verification of cost-effective design modifications to commercially available fixed-wing unmanned aerial vehicle to improve performance, stability and control characteristics, and structural integrity. In *2018 AIAA Information Systems-AIAA Infotech@ Aerospace*, page 2261. AIAA, 2018.

[22] Aaron T Blevins, A R Kim, Daksh Shukla, Shawn S Keshmiri, and Weizhang Huang. Validation and verification flight tests of fixed-wing collaborative uass with high speeds and high inertias. In *2018 Flight Testing Conference*, page 4280, 2018.

[23] Eivind Bøhn, Erlend M Coates, Signe Moe, and Tor Ame Johansen. Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 523–533. IEEE, 2019.

[24] Haitham Bou-Ammar, Holger Voos, and Wolfgang Ertel. Controller design for quadrotor uavs using reinforcement learning. In *Control Applications (CCA), 2010 IEEE International Conference on*, pages 2130–2135. IEEE, 2010.

[25] John Brandt and Michael Selig. Propeller performance data at low reynolds numbers. *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, 2011.

[26] George H Burgin and Steven S Schnetzler. Artificial neural networks in flight control and flight management systems. In *Aerospace and Electronics Conference, 1990. NAECON 1990., Proceedings of the IEEE 1990 National*, pages 567–573. IEEE, 1990.

[27] Anthony J Calise and Rolf T Rysdyk. Nonlinear adaptive flight control using neural networks. *IEEE control systems*, 18(6):14–25, 1998.

[28] Giampiero Campa, Mario Luca Fravolini, Marcello Napolitano, and Brad Seanor. Neural networks-based sensor validation for the flight control system of a b777 research model. In *American Control Conference, 2002. Proceedings of the 2002*, volume 1, pages 412–417. IEEE, 2002.

[29] Min Chen, Yixue Hao, Kai Hwang, Lu Wang, and Lin Wang. Disease prediction by machine learning over big data from healthcare communities. *Ieee Access*, 5:8869–8879, 2017.

[30] Lin Cheng, Zhenbo Wang, Yu Song, and Fanghua Jiang. Real-time optimal control for irregular asteroid landings using deep neural networks. *arXiv preprint arXiv:1901.02210*, 2019.

[31] Seungwon Choi, Suseong Kim, and H Jin Kim. Inverse reinforcement learning control for trajectory tracking of a multirotor uav. *International Journal of Control, Automation and Systems*, 15(4):1826–1834, 2017.

[32] Sanjiban Choudhury, Ashish Kapoor, Gireeja Ranade, and Debadeepta Dey. Learning to gather information via imitation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 908–915. IEEE, 2017.

[33] Adam Coates, Pieter Abbeel, and Andrew Y Ng. Learning for control from multiple demonstrations. In *Proceedings of the 25th international conference on Machine learning*, pages 144–151. ACM, 2008.

[34] Adam Coates, Pieter Abbeel, and Andrew Y Ng. Autonomous helicopter flight using reinforcement learning. In *Encyclopedia of Machine Learning*, pages 53–61. Springer, 2011.

[35] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[36] Gerald C Cottrill. Neural network autopilot system for a mathematical model of the boeing 747. Technical report, WEST VIRGINIA UNIV MORGANTOWN, 1998.

[37] Robert Culkin and Sanjiv R Das. Machine learning in finance: the case of deep learning for option pricing. *Journal of Investment Management*, 15(4):92–100, 2017.

[38] Renwick Curry, Mariano Lizarraga, Bryant Mairs, and Gabriel Hugh Elkaim. L+ 2, an improved line of sight guidance law for uavs. In *2013 American Control Conference*, pages 1–6. IEEE, 2013.

[39] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[40] Konstantinos Dalamagkidis, Kimon P Valavanis, and Les A Piegl. Nonlinear model predictive control with neural network optimization for autonomous autorotation of small unmanned helicopters. *IEEE Transactions on Control Systems Technology*, 19(4):818–831, 2011.

[41] Jan De Spiegeleer, Dilip B Madan, Sofie Reyners, and Wim Schoutens. Machine learning for quantitative finance: fast derivative pricing, hedging and fitting. *Quantitative Finance*, 18(10):1635–1643, 2018.

[42] Analysis Design and Research Corporation. *Advanced Aircraft Analysis*. Design, Analysis and Research Corporation, Lawrence, KS, 3.7 edition, 2017.

[43] Travis Dierks and Sarangapani Jagannathan. Output feedback control of a quadrotor uav using neural networks. *IEEE transactions on neural networks*, 21(1):50–66, 2010.

[44] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[45] Dennis C Duro, Steven E Franklin, and Monique G Dubé. A comparison of pixel-based and object-based image analysis with selected machine learning algorithms for the classification of agricultural landscapes using spot-5 hrg imagery. *Remote sensing of environment*, 118:259–272, 2012.

[46] Ngozi Clara Eli-Chukwu. Applications of artificial intelligence in agriculture: A review. *Engineering, Technology & Applied Science Research*, 9(4):4377–4383, 2019.

[47] Nicholas Ernest, David Carroll, Corey Schumacher, Matthew Clark, Kelly Cohen, and Gene Lee. Genetic fuzzy based artificial intelligence for unmanned combat aerial vehicle control in simulated air combat missions. *Journal of Defense Management*, 6(1):2167–0374, 2016.

[48] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29, 2019.

[49] Jurgen Everaerts et al. The use of unmanned aerial vehicles (uavs) for remote sensing and mapping. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37(2008):1187–1192, 2008.

[50] Aleksandra Faust, Ivana Palunko, Patricio Cruz, Rafael Fierro, and Lydia Tapia. Learning

swing-free trajectories for uavs with a suspended load. In *2013 IEEE International Conference on Robotics and Automation*, pages 4902–4909. IEEE, 2013.

[51] Reeves Fletcher and Colin M Reeves. Function minimization by conjugate gradients. *The computer journal*, 7(2):149–154, 1964.

[52] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[53] King-Sun Fu. Learning control systems–review and outlook. *IEEE transactions on Automatic Control*, 15(2):210–221, 1970.

[54] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. *arXiv preprint arXiv:1704.05588*, 2017.

[55] Freddy Rafael Garces, Victor Manuel Becerra, Chandrasekhar Kambhampati, and Kevin Warwick. *Strategies for feedback linearisation: a dynamic neural network approach*. Springer Science & Business Media, 2012.

[56] Gonzalo A Garcia, Shawn Kashmiri, and Daksh Shukla. Nonlinear control based on h-infinity theory for autonomous aerial vehicle. In *Unmanned Aircraft Systems (ICUAS), 2017 International Conference on*, pages 336–345. IEEE, 2017.

[57] C Lee Giles and Tom Maxwell. Learning, invariance, and generalization in high-order neural networks. *Applied optics*, 26(23):4972–4978, 1987.

[58] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[59] Ryan David Goodnight. *An Innovative Approach for Data Collection and Handling to Enable Advancements in Micro Air Vehicle Persistent Surveillance*. PhD thesis, Texas A&M University, 2009.

[60] CM Ha. Neural networks approach to aiaa aircraft control design challenge. *Journal of Guidance, Control, and Dynamics*, 18(4):731–739, 1995.

[61] JB Heaton, NG Polson, and Jan Hendrik Witte. Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry*, 33(1):3–12, 2017.

[62] Magnus Rudolph Hestenes. *Conjugate direction methods in optimization*, volume 12. Springer Science & Business Media, 2012.

[63] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14:8, 2012.

[64] Joseph Horn, Brian Geiger, and Eric Schmidt. Use of neural network approximation in multiple-unmanned aerial vehicle trajectory optimization. In *AIAA Guidance, Navigation, and Control Conference*, page 6103, 2009.

[65] Joseph F Horn, Eric M Schmidt, Brian R Geiger, and Mark P DeAngelo. Neural network-based trajectory optimization for unmanned aerial vehicles. *Journal of Guidance, Control, and Dynamics*, 35(2):548–562, 2012.

[66] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[67] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[68] Robert A Jacobs. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307, 1988.

[69] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

[70] Matt R. Jardin and Eric R. Mueller. Optimized measurements of unmanned-air-vehicle mass moment of inertia with a bifilar pendulum. *Journal of Aircraft*, 46, 2009.

[71] Fei Jiang, Yong Jiang, Hui Zhi, Yi Dong, Hao Li, Sufeng Ma, Yilong Wang, Qiang Dong, Haipeng Shen, and Yongjun Wang. Artificial intelligence in healthcare: past, present and future. *Stroke and vascular neurology*, 2(4):230–243, 2017.

[72] Erik M Johansson, Farid U Dowla, and Dennis M Goodman. Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method. *International Journal of Neural Systems*, 2(04):291–301, 1991.

[73] Charles C Jorgensen. Neural networks for flight control. 1996.

[74] Charles C Jorgensen. Direct adaptive aircraft control using dynamic cell structure neural networks. 1997.

[75] Kjell Kersandt. Deep reinforcement learning as control method for autonomous uavs. Master's thesis, Universitat Politècnica de Catalunya, 2018.

[76] A Ram Kim, Daksh Shukla, Aaron Blevins, Shawn Keshmiri, and Mark Ewing. Validation and verification of gnc for large fixed-wing unmanned aerial system in unstructured and noisy environment. In *Unmanned Aircraft Systems (ICUAS), 2018 International Conference on*. IEEE, 2018.

[77] A Ram Kim, Prasanth Vivekanandan, Patrick McNamee, Ian Sheppard, Aaron Blevins, and Alex Sizemore. Dynamic modeling and simulation of a quadcopter with motor dynamics. *AIAA Modeling and Simulation Technologies Conference*, 2017.

[78] Byoung S Kim and Anthony J Calise. Nonlinear flight control using neural networks. *Journal of Guidance, Control, and Dynamics*, 20(1):26–33, 1997.

[79] Dong Ki Kim and Tsuhan Chen. Deep neural network for real-time autonomous indoor navigation. *arXiv preprint arXiv:1511.04668*, 2015.

[80] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[81] Konstantina Kourou, Themis P Exarchos, Konstantinos P Exarchos, Michalis V Karamouzis, and Dimitrios I Fotiadis. Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal*, 13:8–17, 2015.

[82] Erwin Kreyszig. Advanced engineering mathematics, 10th eddition, 2009.

[83] Eugene Lavretsky and Kevin A Wise. *Robust and Adaptive Control*. Springer, 2013.

[84] D Jin Lee, Hyochoong Bang, and Kwangyul Baek. Autorotation of an unmanned helicopter by a reinforcement learning algorithm. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 7279, 2008.

[85] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[86] FL Lewis and Shuzhi Sam Ge. Neural networks in feedback control systems. *Mechanical Engineer's Handbook*, 2005.

[87] Konstantinos G Liakos, Patrizia Busato, Dimitrios Moshou, Simon Pearson, and Dionysis Bochtis. Machine learning in agriculture: A review. *Sensors*, 18(8):2674, 2018.

[88] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[89] Richard P Lippmann. Anintroduction to computing with neural nets. *IEEE Assp magazine*, 4(2):4–22, 1987.

[90] Peter Liu, Albert Y Chen, Yin-Nan Huang, Jen-Yu Han, Jihn-Sung Lai, Shih-Chung Kang, Tzong-Hann Wu, Ming-Chang Wen, Meng-Han Tsai, et al. A review of rotorcraft unmanned aerial vehicle (uav) developments and applications in civil engineering. *Smart Struct. Syst*, 13(6):1065–1094, 2014.

[91] Jerzy Manerowski and Dariusz Rykaczewski. Modelling of uav flight dynamics using perceptron artificial neural networks. *Journal of theoretical and applied mechanics*, 43, 2005.

[92] Gunasekaran Manogaran and Daphne Lopez. A survey of big data architectures and machine learning algorithms in healthcare. *International Journal of Biomedical Engineering and Technology*, 25(2-4):182–211, 2017.

[93] Andrés Marcos and Gary J Balas. Development of linear-parameter-varying models for aircraft. *Journal of Guidance, Control, and Dynamics*, 27(2):218–228, 2004.

[94] Matlab. *version 9.4 (R2018a)*. The MathWorks Inc., Natick, Massachusetts, 2018.

[95] Matlab deep neural network toolbox v13.0, 9.7.0.1190202 (R2019b). The MathWorks, Natick, MA, USA.

[96] Matlab reinforcement learning toolbox v1.1, 9.7.0.1190202 (R2019b). The MathWorks, Natick, MA, USA.

[97] D Mavris. Application of machine learning for aircraft design. In *CDT–Big Data Analytics and Machine Intelligence Team Hosting, Machine Learning Technologies and Their Applications to Scientific and Engineering Domains Workshop*, 2016.

[98] Robert J McQueen, Stephen R Garner, Craig G Nevill-Manning, and Ian H Witten. Applying machine learning to agricultural data. *Computers and electronics in agriculture*, 12(4):275–293, 1995.

[99] Alan Tan Wei Min, Ramon Sagarna, Abhishek Gupta, Yew-Soon Ong, and Chi Keong Goh. Knowledge transfer through machine learning in aircraft design. *IEEE Computational Intelligence Magazine*, 12(4):48–60, 2017.

[100] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[101] Martin F Møller. Learning by conjugate gradients. In *International Meeting of Young Computer Scientists*, pages 184–194. Springer, 1990.

[102] Martin Fodslette Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.

[103] Marcello R Napolitano, Charles Neppach, Van Casdorph, Steve Naylor, Mario Innocenti, and Giovanni Silvestri. Neural-network-based scheme for sensor failure detection, identification, and accommodation. *Journal of Guidance, Control, and Dynamics*, 18(6):1280–1286, 1995.

[104] Marsello R Napolitano and Michael Kincheloe. On-line learning neural-network controllers for autopilot systems. *Journal of Guidance, Control, and Dynamics*, 18(5):1008–1015, 1995.

[105] Fredrik Olsson. A literature survey of active machine learning in the context of natural language processing. 2009.

[106] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning in natural language processing. *arXiv preprint arXiv:1807.10854*, 2018.

[107] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile off-road autonomous driving using end-to-end deep imitation learning. *arXiv preprint arXiv:1709.07174*, 2017.

[108] Sanghyuk Park, John Deyst, and Jonathan How. A new nonlinear guidance logic for trajectory tracking. In *AIAA guidance, navigation, and control conference and exhibit*, page 4900, 2004.

[109] Urpo J Pesonen, James E Steck, Kamran Rokhsaz, Hugh Samuel Bruner, and Noel Duerksen. Adaptive neural network inverse controller for general aviation safety. *Journal of Guidance, Control, and Dynamics*, 27(3):434–443, 2004.

[110] Ali Punjani and Pieter Abbeel. Deep learning helicopter dynamics models. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3223–3230. IEEE, 2015.

[111] Gintaras V Puskorius and Lee A Feldkamp. Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks. *IEEE Transactions on neural networks*, 5(2):279–297, 1994.

[112] Python software foundation. python language reference, version 2.7, Release Date: Oct. 19, 2019. Available at http://www.python.org.

[113] Domenico Quagliarella and Antonio Della Cioppa. Genetic algorithms applied to the aerodynamic design of transonic airfoils. *Journal of Aircraft*, 32(4):889–891, 1995.

[114] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[115] G Ribière and E Polak. Note sur la convergence de directions conjugées. *Rev. Francaise Informat Recherche Opertionelle*, 16:35–43, 1969.

[116] Jan Roskam. *Airplane flight dynamics and automatic flight controls*. DARcorporation, 1998.

[117] Jan Roskam. Airplane flight dynamics and automatic flight controls. *DARcorporation*, 2001.

[118] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[119] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive uav con-

trol in cluttered natural environments. In *2013 IEEE international conference on robotics and automation*, pages 1765–1772. IEEE, 2013.

[120] Alireza Roudbari and Fariborz Saghafi. Intelligent modeling and identification of aircraft nonlinear flight. *Chinese Journal of Aeronautics*, 27(4):759–771, 2014.

[121] DE Rumellhart. Learning internal representations by error propagation. *Parallel distributed processing*, 1:318–362, 1986.

[122] Malcolm Ryan and Mark Reid. Learning to fly: An application of hierarchical reinforcement learning. In *In Proceedings of the 17th International Conference on Machine Learning*, pages 807–814. Morgan Kaufmann, 2000.

[123] Abd Manan Samad, Nazrin Kamarulzaman, Muhammad Asyraf Hamdani, Thuaibatul Aslamiah Mastor, and Khairil Afendy Hashim. The potential of unmanned aerial vehicle (uav) for civilian and mapping application. In *2013 IEEE 3rd International Conference on System Engineering and Technology*, pages 313–318. IEEE, 2013.

[124] Mahendra Kumar Samal, Sreenatha Anavatti, and Matthew Garratt. Neural network based system identification for autonomous flight of an eagle helicopter. *IFAC Proceedings Volumes*, 41(2):7421–7426, 2008.

[125] Carlos Sánchez-Sánchez and Dario Izzo. Real-time optimal control via deep neural networks: study on landing problems. *Journal of Guidance, Control, and Dynamics*, 41(5):1122–1135, 2018.

[126] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[127] Kumba Sennaar. Ai in agriculture–present applications and impact. *TechEmergence, October*, 16, 2017.

[128] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[129] Daksh Shukla, A R Kim, Aaron T Blevins, Shawn S Keshmiri, and Mark Ewing. Validation and verification flight testing of uas morphing potential field collision avoidance algorithms. In *2018 Flight Testing Conference*, page 4279, 2018.

[130] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395, 2014.

[131] Sahjendra N Singh, Woosoon Yirn, and William R Wells. Direct adaptive and neural control of wing-rock motion of slender delta wings. *Journal of Guidance, Control, and Dynamics*, 18(1):25–30, 1995.

[132] Eduardo D Sontag. Feedback stabilization using two-hidden-layer nets. In *1991 American Control Conference*, pages 815–820. IEEE, 1991.

[133] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.

[134] James Steck, Kamran Rokhsaz, James Steck, and Kamran Rokhsaz. Some applications of artificial neural networks in modeling of nonlinear aerodynamics and flight dynamics. In *35th Aerospace Sciences Meeting and Exhibit*, page 338, 1997.

[135] Robert F Stengel. *Flight dynamics*. Princeton University Press, 2015.

[136] Brian L Stevens, Frank L Lewis, and Eric N Johnson. *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons, 2015.

[137] Bjarne Stroustrup. *The C++ programming language*. Pearson Education India, 2000.

[138] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[139] Robert R Trippi, Preface By-Lee, and K Jae. *Artificial intelligence in finance and investing: state-of-the-art technologies for securities selection and portfolio management*. McGraw-Hill, Inc., 1995.

[140] Terry Troudet, Sanjay Garg, and Walter Merrill. Neural network application to aircraft control system design. In *Navigation and Control Conference*, page 2715, 1991.

[141] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

[142] DARPA Neural Network Study (US). *DARPA Neural Network Study: October 1987-February 1988*. AFCEA International Press, 1988.

[143] Deepak Vasisht, Zerina Kapetanovic, Jongho Won, Xinxin Jin, Ranveer Chandra, Sudipta Sinha, Ashish Kapoor, Madhusudhan Sudarshan, and Sean Stratman. Farmbeats: An iot platform for data-driven agriculture. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 515–529, 2017.

[144] Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *Advances in neural information processing systems*, pages 2773–2781, 2015.

[145] Chang Wang, Chao Yan, Xiaojia Xiang, and Han Zhou. A continuous actor-critic reinforcement learning approach to flocking with fixed-wing uavs. In *Asian Conference on Machine Learning*, pages 64–79, 2019.

[146] Darrell Whitley and Nachimuthu Karunanithi. Generalization in feed forward neural networks. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume 2, pages 77–82. IEEE, 1991.

[147] Jenna Wiens and Erica S Shenoy. Machine learning for healthcare: on the verge of a major shift in healthcare epidemiology. *Clinical Infectious Diseases*, 66(1):149–153, 2017.

[148] Timothy D Woodbury, Caroline Dunn, and John Valasek. Autonomous soaring using reinforcement learning for trajectory generation. In *52nd Aerospace Sciences Meeting*, page 0990, 2014.

[149] Jie Xu, Tao Du, Michael Foshey, Beichen Li, Bo Zhu, Adriana Schulz, and Wojciech Matusik. Learning to fly: computational controller design for hybrid uavs with reinforcement learning. *ACM Transactions on Graphics (TOG)*, 38(4):42, 2019.

[150] Zarita Zainuddin and Ong Pauline. Function approximation using artificial neural networks. *WSEAS Transactions on Mathematics*, 7(6):333–338, 2008.

[151] Baochang Zhang, Zhili Mao, Wanquan Liu, and Jianzhuang Liu. Geometric reinforcement learning for path planning of uavs. *Journal of Intelligent & Robotic Systems*, 77(2):391–409, 2015.

[152] Chunhua Zhang and John M Kovacs. The application of small unmanned aerial systems for precision agriculture: a review. *Precision agriculture*, 13(6):693–712, 2012.

[153] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 528–535. IEEE, 2016.