Fall 12-8-2022

Development and Deployment of a Dynamic Soaring Capable UAV using Reinforcement Learning

Jacob Adamski
Embry-Riddle Aeronautical University, adamskij@my.erau.edu

DEVELOPMENT AND DEPLOYMENT OF A DYNAMIC SOARING CAPABLE UAV USING REINFORCEMENT LEARNING

By

Jacob Adamski

A Thesis Submitted to the Faculty of Embry-Riddle Aeronautical University

In Partial Fulfillment of the Requirements for the Degree of

Master of Science in Aerospace Engineering

November 2022

Embry-Riddle Aeronautical University

Daytona Beach, Florida

DEVELOPMENT AND DEPLOYMENT OF A DYNAMIC SOARING CAPABLE UAV USING REINFORCEMENT LEARNING

By

Jacob Adamski

This Thesis was prepared under the direction of the candidate's Thesis Committee Chair, Dr. Vladimir Golubev, Department of Aerospace Engineering, and has been approved by the members of the Thesis Committee. It was submitted to the Office of the Senior Vice President for Academic Affairs and Provost, and was accepted in the partial fulfillment of the requirements for the Degree of Master of Science in Aerospace Engineering.

THESIS COMMITTEE

Vladimir Golubev
Digitally signed by Vladimir Golubev
DN: cn=Vladimir Golubev, o=ERAU, ou=Aerospace Engineering, email=golubd1b@erau.edu, c=US
Date: 2022.12.06 13:06:33 -05'00'

Chair, Dr. Vladimir Golubev

Snorri Gudmundsson
Digitally signed by Snorri Gudmundsson
Date: 2022.12.07 18:02:15 -05'00'

Member, Dr. Snorri Gudmundsson

Richard J. Prazenica
Digitally signed by Richard J. Prazenica
Date: 2022.12.07 18:56:05 -05'00'

Member, Dr. Richard Prazenica

William MacKunis
Digitally signed by William MacKunis
Date: 2022.12.08 06:28:27 -05'00'

Member, Dr. William MacKunis

Hever Moncayo
Digitally signed by Hever Moncayo
Date: 2022.12.08 11:30:03 -05'00'

Graduate Program Coordinator,
Dr. Hever Moncayo

12/08/2022

Date

Dean of the College of Engineering,
Dr. James W. Gregory

Date

Associate Provost of Academic Support,
Dr. Christopher Grant

Date

ACKNOWLEDGEMENTS

I would like to thank my mother, Tracy, father, Joseph, and sister, Alexandra for their continuous support through this difficult process.

I would like to thank my advisors Dr. Vladimir Golubev and Dr. Snorri Gudmundsson for their guidance on all aspects of the dynamic soaring project. I would also like to thank my other thesis committee members Dr. Richard Prazenica and Dr. William MacKunis for their guidance in the completion of my thesis.

I would like to thank Alex Carrion, my mentor during my internship at Pall Aerospace Research and Development. His guidance helped me to discover my potential as an engineer and ultimately convinced me to pursue a master's degree.

I would like to thank all undergraduate students who assisted with this project both virtually and physically. This project would not have been so ambitious without the support of other talented engineering students.

Lastly, I thank the Embry-Riddle Office of Undergraduate Research for their funding of the Dynamic Soaring IGNITE project. With these funds, the possibilities for dynamic soaring research at ERAU has been expanded. I hope to see this research continue to grow in the future.

ABSTRACT

Dynamic soaring (DS) is a bio-inspired flight maneuver in which energy can be gained by flying through regions of vertical wind gradient such as the wind shear layer. With reinforcement learning (RL), a fixed wing unmanned aerial vehicle (UAV) can be trained to perform DS maneuvers optimally for a variety of wind shear conditions. To accomplish this task, a 6-degrees-of-freedom (6DoF) flight simulation environment in MATLAB and Simulink has been developed which is based upon an off-the-shelf unmanned aerobatic glider. A combination of high-fidelity Reynolds-Averaged Navier-Stokes (RANS) computational fluid dynamics (CFD) in ANSYS Fluent and low-fidelity vortex lattice (VLM) method in Surfaces was employed to build a complete aerodynamic model of the UAV. Deep deterministic policy gradient (DDPG), an actor-critic RL algorithm, was used to train a closed-loop Path Following (PF) agent and an Unguided Energy-Seeking (UES) agent. Several generations of the PF agent were presented, with the final generation capable of controlling the climb and turn rate of the UAV to follow a closed-loop waypoint path with variable altitude. This must be paired with a waypoint optimizing agent to perform loitering DS. The UES agent was designed to perform traveling DS in a fixed wind shear condition. It was proven to extract energy from the wind shear to extend flight time during training but did not accomplish sustainable dynamic soaring. Further RL training is required for both agents. Recommendations on how to deploy an RL agent on a physical UAV are discussed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

Mother nature has evolved avian species of all kinds to fly optimally through a variety of clever maneuvers. Certain species may fly in formations to take advantage of the upwash generated by wingtip vortices. Birds in the rear of the formation can exploit the upwash generated upstream to fly with considerably less effort. Other species may take advantage of the ground effect to reduce drag over bodies of water. Alternatively, some species of birds may exploit thermal updrafts to gain free lift by flying in circular patterns in the heated air. Finally, large sea-fairing birds, such as the albatross, have evolved to take advantage of the wind shear layer above the ocean to fly for long distances with minimal energy expenditure. This is known as dynamic soaring and, compared to the other maneuvers, is not easily employed by man-made aerial vehicles. This research seeks to explore how to exploit this effect autonomously for fixed wing unmanned aerial vehicles.

1.1 Dynamic Soaring

Dynamic soaring is a bio-inspired maneuver in which energy is gained by flying through regions of vertical wind gradient. By exploiting wind gradients in nature, albatrosses can fly extremely long distances with little energy expenditure. To a similar effect, hobbyist model aviators utilize dynamic soaring to fly radio-controlled sailplanes in circular patterns over the edge of a cliff at extremely high airspeeds. Exploitable wind gradients can be found over a wide variety of geographical features such as flat land, bodies of water, or mountainous areas. The profile of the wind gradient changes in shape and magnitude depending on location. For instance, over the edge of a cliff, the velocity gradient is extremely sharp, with high velocity immediately over the edge and low velocity just below. Over the ocean the wind gradient resembles a viscous boundary layer with the gradient decreasing as height increases. In general, the larger the wind shear gradient, the better suited the conditions are for dynamic soaring.

There are four main phases to a dynamic soaring cycle:

1. Low Altitude Turn
2. Windward Climb
3. High Altitude Turn
4. Leeward Descent

In the low altitude turn, the UAV is at its maximum kinetic energy and minimum potential energy, or, in other words, maximum velocity and minimum altitude. This is immediately succeeded by the windward climb in which kinetic energy is exchanged with potential energy. At the top of the maneuver is the high altitude turn or when potential energy is maximum and kinetic energy is minimum. It is important that the UAV does not stall during this phase. Finally, the UAV executes the leeward descent to exchange the potential energy for kinetic energy. If dynamic soaring is successful, the total energy should be conserved at each point during the cycle. This can only be accomplished by harvesting energy from the vertical wind gradient. Otherwise, internal thrust from an engine would be required to maintain energy.

There are two primary variations of the dynamic soaring maneuver: traveling and loitering. A diagram of the traveling maneuver can be seen in Figure 1.1. The traveling maneuver is an open-loop cycle that minimizes the energy expenditure to travel over long distances. This maneuver is used by the albatross to travel long distances over the ocean. There exists a 180-degree field of potential flying directions relative to the wind where traveling dynamic soaring is possible. This is loosely analogous to a sailing ship. It cannot easily sail against the direction of the wind, rather the ship must travel within the natural constraints created by the wind. To maximize efficiency, a UAV should travel parallel to the wind direction. In contrast, to travel perpendicularly to the wind

direction extra energy must be expended to return to the starting position upwind. Thus, why parallel travel is most efficient and perpendicular is least efficient.

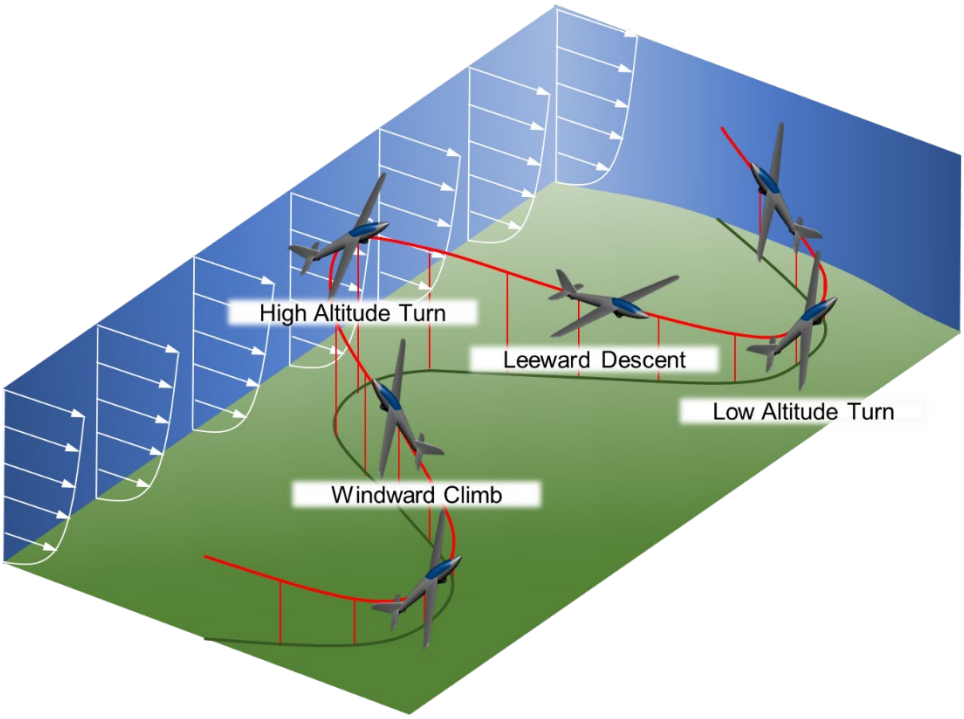

Figure 1.1 Diagram of the traveling dynamic soaring cycle.

A diagram of the loitering dynamic soaring cycle can be seen in Figure 1.2. This is mechanically similar to traveling dynamic soaring perpendicular to the wind direction. Rather, the UAV completes the leeward descent and windward climb in the same turning direction. Again, to reiterate the consequence, the UAV must expend additional energy in the real world to return to the starting position in the cycle. If too little enough energy is expended for this purpose the UAV will simply translate downwind the same as the traveling dynamic soaring maneuver. The loitering dynamic soaring maneuver is more practical to perform in regions of high wind gradient such as cliff soaring but may be impractical over the ocean where the gradient is smaller.

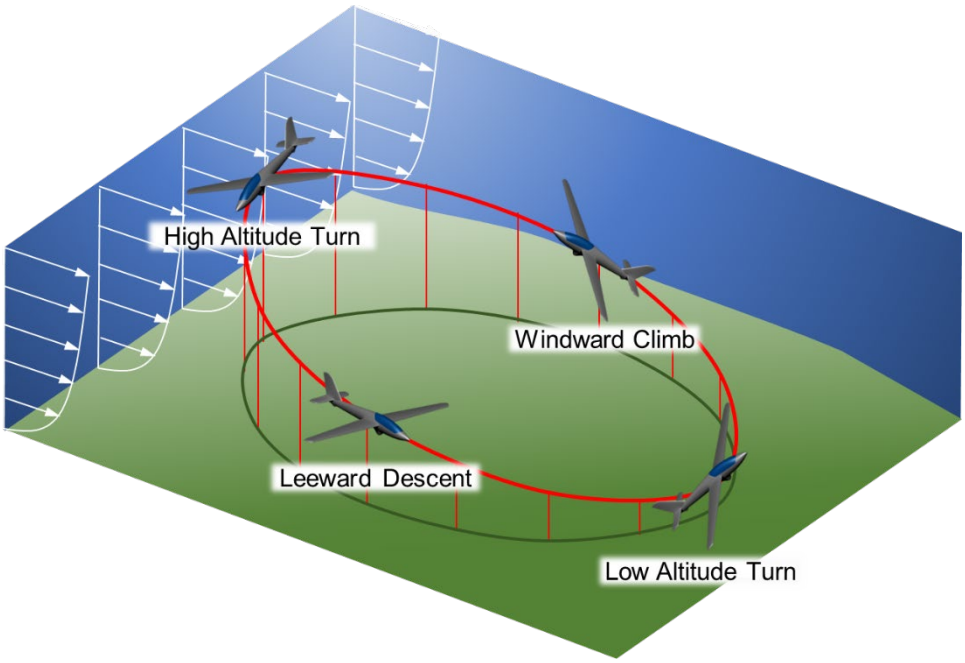

Figure 1.2 Diagram of the loitering dynamic soaring cycle.

1.2 Dynamic Soaring from Reinforcement Learning

Reinforcement learning (RL) will be utilized to develop both traveling and loitering dynamic soaring cycles. Reinforcement learning is a machine learning technique that develops an optimal policy function to solve a control problem by learning from experience. Unlike regular machine learning, no dataset is needed to train the genetic algorithm. A reinforcement learning agent can be trained with no knowledge of its environment. For training to be successful, an agent must balance exploration of its environment and exploitation of potential rewards. The results of several variations of reinforcement learning agents will be presented to evaluate the feasibility of using reinforcement learning to perform dynamic soaring.

There are two potential approaches to the application of reinforcement learning to dynamic soaring. The first, which will be referred to as the Unguided Energy-Seeking (UES) method, utilizes a single agent that controls all aspects of the maneuver. This agent controls the UAV's path and direction with the objective of gaining enough energy to maintain stable flight. A wind

shear is applied to the model, and the agent must learn without any prior knowledge how to extract energy from it. The second approach is a dual-agent method. In this case, the first agent, known as the path following (PF), controls the UAV to follow a predefined path and the second agent, referred to as the Path Optimizing (PO) agent, optimizes the path for dynamic soaring. This should be a more robust method compared to the single-agent method, allowing for precise control over the dynamic soaring cycle. However, this involves higher complexity compared to the UES method.

1.3 Objectives

There are two primary objectives for this research. The first is to use reinforcement learning to simulate dynamic soaring. The agent should be able to optimize the UAV's maneuver to maintain flight without engine thrust. Also, the reinforcement learning agent should be able to adapt to ever-changing conditions as experienced in the real-world. In the event that the wind shear is not strong enough to maintain dynamic soaring, then the UAV should use thrust to maintain flight.

The second goal is to develop the foundation for deployment of reinforcement learning based dynamic soaring in a physical UAV. Ultimately, a similar reinforcement learning algorithm will be installed on a physical UAV to test autonomous dynamic soaring maneuvers in the real world. This, however, is not attempted in this research due to time constraints. Nevertheless, the steps that should be taken to achieve real-world autonomous dynamic soaring will be addressed.

1.4 Importance of Research

This research attempts to provide answers to how dynamic soaring can be achieved and implemented in the real world. Substantial research exists that mathematically optimizes an ideal dynamic soaring path given a perfect, fixed wind shear condition. However, for dynamic soaring to work in practice, it needs to be adaptable to all atmospheric disturbances. Because it can generate an optimal policy for nearly any type of problem, reinforcement learning is chosen as the

optimization method. Ultimately, significant research is required to bridge the gap between the results and real-time implementation.

2 Review of the Relevant Literature

This thesis is a continuation of previous research on dynamic soaring by Gladston Joseph [1]. However, almost all methods from the previous research have been updated. A simple waypoint-based feedback-control autopilot was developed to perform traveling dynamic soaring maneuvers for a fixed wind shear condition. In addition to the autonomous dynamic soaring methods, human piloted dynamic soaring was performed for comparison. Dynamic soaring was achievable for both methods primarily as a result of an extremely high-lift-low-drag UAV model. Autonomous dynamic soaring performed consistent, yet inefficient maneuvers due to artificial limitations in the control laws. In contrast, manually piloted dynamic soaring was more efficient, but less consistent due to human control. Figure 2.1 shows a snapshot of the maneuvers on a single plot.

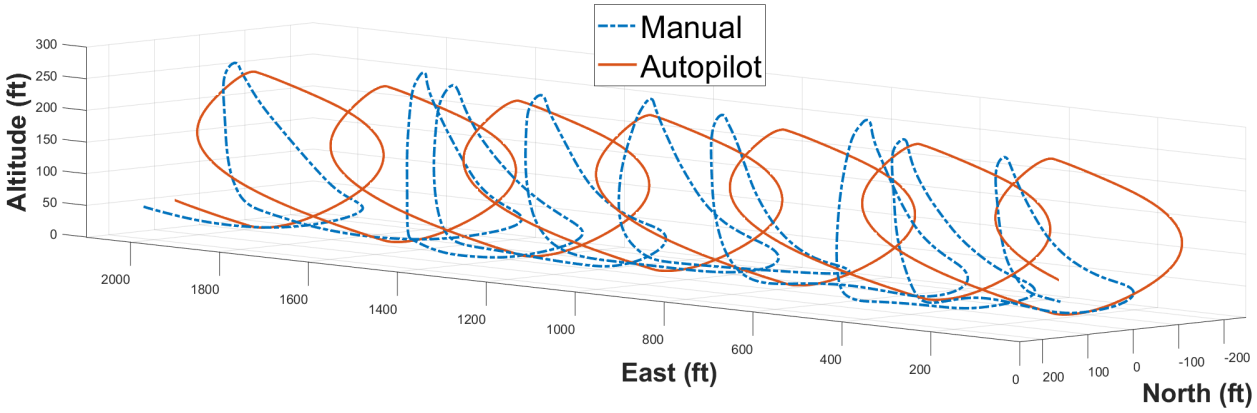

Figure 2.1 Results of the autonomous and manually piloted dynamics soaring cycles from Gladston Joseph's thesis [1].

Reinforcement learning algorithms have been used to optimize a variety of soaring problems. Chung, Lawrence, and Sukkarieh used the eGP-SARSA(λ) algorithm to teach an unpowered glider to perform energy gaining maneuvers in a thermal updraft [2]. While not directly related to DS, their paper demonstrated the ability of this algorithm to train a UAV to find energy-gaining

trajectories under resource-constrained conditions. Reddy et al. applied the Markov decision process to glider soaring in the field [3]. They developed an autonomous thermal control law using reinforcement learning in the real world. Their results demonstrate that reinforcement learning can also be applied to a real-world soaring application. Yet another work on reinforcement learning based soaring is by Woodbury, Dunn, and Valasek [4]. In this work, Q-Learning was applied to train a UAV to navigate to a known thermal updraft using bank-angle commands.

Regarding RL applied to DS maneuvers, Perez et al. applied the Neuro Evolution of Augmented Topologies (NEAT) algorithm to the dynamic soaring flight trajectory optimization problem [5]. The NEAT algorithm utilizes an evolutionary optimizer to build the simplest possible artificial neural network for dynamic soaring. Their work demonstrated the advantages of reinforcement learning algorithms over traditional optimization methods by using RL to further optimize a previously generated DS trajectory. Additionally, they were able to adapt their RL algorithm for DS in sinusoidally varying wind shear conditions. Barate, Doncieux, and Meyer hand-designed a DS trajectory using Takagi-Sugeno-Kang (TSK) fuzzy rules and optimized the trajectory with evolutionary algorithms [6]. Montella and Spletzer demonstrated that their RL trained DS controller could achieve better performance compared to a traditionally optimized teacher DS controller when deviations were made from the planned trajectory [7]. It is evident from the results of these works that RL presents an optimization paradigm which can achieve better performance and robustness compared to traditional control optimization methods.

There has been limited research in the area of real-world autonomous dynamic soaring. Until recently, UAV platforms were not quite capable of achieving the performance necessary for autonomous dynamic soaring in wind shears over flat land. Recent innovations in cost and flight-

controller performance, however, make these attempts at real-world dynamic soaring more practical, even for small research teams.

The first attempt of note was performed by Mark Boslough [8] at Sandia National Laboratories in 2001. The subject of his research also covered dynamic soaring simulations and optimization of the dynamic soaring maneuver using genetic algorithms. Boslough's flight test consisted of human-piloted ridge-soaring where the wind shear gradient is sharpest. Three separate maneuvers were recorded. First with the UAV held at a constant airspeed in a headwind over the ridge resulting in steady altitude gain, second with repeated climbs and dives into the headwind resulting in a net loss in total energy, and thirdly loitering dynamic soaring over the ridge for which a net gain in total energy was achieved. In 2015 Zhu et al. [9] attempted dynamic soaring experiments to verify their simulation data. Instead of repeated dynamic soaring cycles, they performed a single dynamic soaring cycle to compare the airspeed, flight path, and heading between the simulation and experiment. There was no net gain in energy during the flights. Another attempt at real-world dynamic soaring was carried out by Corey Montella as part of his dissertation [10]. While significant work was performed to optimize the dynamic soaring path in simulation, his flight tests did not achieve autonomous or human-piloted dynamic soaring due to the lack of wind shear during testing. Finally, Bronz et al. [11] attempted autonomous dynamic soaring experiments over the edge of a cliff showing that their glider could extract up to 60% of the required power from dynamic soaring despite lacking a well optimized path.

Real-world autonomous dynamic soaring simply has not been achieved yet. While there have been several attempts to quantify energy gain using autonomous control, none have truly achieved full repeatable dynamic soaring cycles.

3 Methodology

Reinforcement learning for the purpose of deployment on a physical UAV requires a high-fidelity flight simulator. To this end, the methods for obtaining a high-fidelity aerodynamic model are described in addition to the methods for building the flight simulation environment and reinforcement learning agents.

3.1 Dynamic Soaring UAV Platform

Dynamic soaring can, theoretically, be performed with any fixed-wing UAV given the right wind conditions and an adequately designed autonomous control law. However, reality dictates that even the most ideal UAV candidate cannot maintain DS forever. Eventually, the wind conditions will not be sufficient such that even the most aerodynamically efficient sailplane cannot gain enough energy from wind alone to maintain flight. In such an event, the UAV must utilize an onboard powerplant to prevent a fatal collision with the ground.

The ideal vehicle to perform dynamic soaring is a high-strength, lightweight glider with a high lift-drag ratio. This idealized UAV minimizes energy-loss through drag as a result of a high wing aspect-ratio and thin fuselage section. Alternatively, an argument can be made that such a UAV should also have a high mass similar to the wandering albatross to maximize the contribution to kinetic energy from gravity as well as from the wind during the leeward descent phase. Of course, this is a tradeoff as more lift would be required compared to a lighter UAV of the same design, necessitating higher airspeed throughout the maneuver.

To exploit the vertical wind gradient near the ground for long-range circumnavigational dynamic soaring, a UAV needs to be autonomous and intelligent. The wind gradients experienced near the ground are not large enough to be consistently utilized by a human pilot. Furthermore, to be intelligent requires the UAV to have onboard propulsion to recover airspeed and maintain altitude in the absence of wind.

Considering these requirements, the FMS Fox Aerobatic Glider was selected as the UAV platform for dynamic soaring. The most basic description of the Fox is that it is a glider with a motor. For a more detailed description, the geometric and mass properties of the Fox can be found in Table 3.1. The Fox has a 3-meter wingspan and wing aspect ratio of 12. Its powerplant is an electrically powered motor with a folding propeller mounted on the nose. When not in use, the propellors fold back and are pushed against the fuselage by the wind. Despite its glider classification, the Fox has a spacious fuselage which allows for great compatibility with many different flight controllers and avionics equipment for autonomous control. This large fuselage unfortunately contributes a large profile drag, which is a necessary tradeoff. With these factors in consideration, the Fox is a great candidate UAV for the development of autonomous DS.

Table 3.1 Geometric properties of the Fox UAV.

Variable	**WING**	**HT**	**VT**	**UNIT**
Airfoil	Clark Y	NACA 0015	NACA 0010	-
C_R	350	200	360	mm
C_T	164	160	190	mm
MGC	257	180	275	mm
b	3000	750	385	mm
Λ_{LE}	1	5	43	deg
Λ_{TE}	-6	-1.1	21.2	deg
S	746500	65961	97548	mm^2
I	2	0	0	deg
AR	12.056	3.44	1.33	-
TR	0.469	0.8	0.523	-

Before real-world testing, autonomous DS must be proven in a simulation using the Fox. This requires the creation of a complete aerodynamic model. The first step is to design a high-fidelity CAD model, as can be seen in Figure 3.1. The CAD model was created using the parametric computer-aided-design software CATIA. All dimensions were obtained by physically measuring the real Fox UAV. Unfortunately, the true airfoil of the Fox UAV is unknown, so a “close-enough” low-speed airfoil was chosen. It is unlikely that this discrepancy would have a significant effect

on the performance of the simulated Fox compared to the real Fox. Otherwise, the only major difference is that the landing gear are modeled as smooth surfaces to simplify the model slightly for CFD. Again, this simplification will not have a great effect on the accuracy final aerodynamic model.

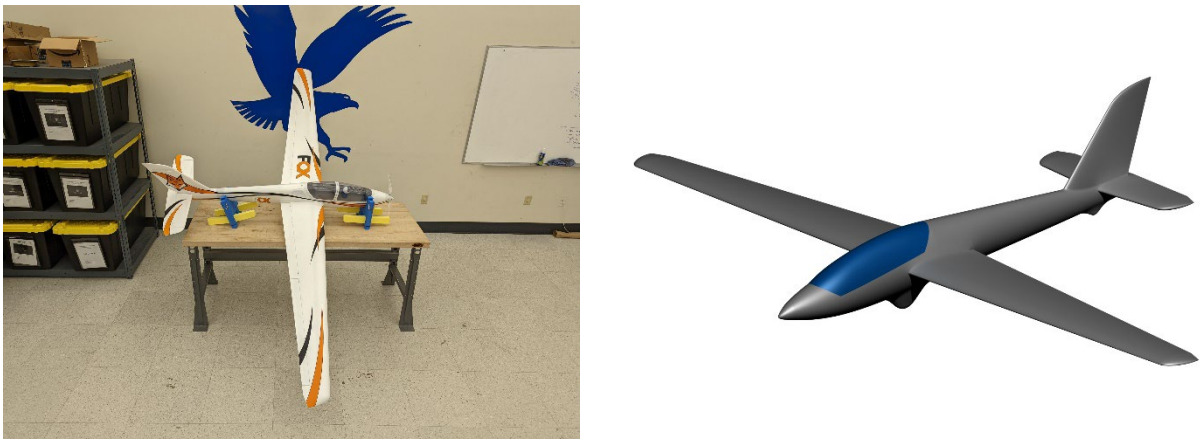

Figure 3.1 Comparison of the physical FMS Fox UAV (left) and Fox CAD model.

3.1.1 Fox Avionics

In simulation, any property can be obtained to describe the state of an aircraft. However, in the real world these properties must be obtained using sensors inside the aircraft. Not all variables can be easily measured. For example, angle of attack is the most direct method to evaluate the stall characteristics of an aircraft. It is simple to quantify in a point-mass based simulation. In practice, measurement of angle of attack is not feasible for small scale UAVs due to the lack of low-cost angle of angle of attack sensors. Because of these physical limitations, if a reinforcement learning agent is to be trained offline in a simulator and then deployed on a real UAV, it must only utilize aircraft data that can be obtained practically with existing sensors.

Internally, the Fox UAV contains a full suite of avionics to allow for the measurement of position, groundspeed, airspeed, attitude, and to a limited extent, angle of attack. To control the aircraft autonomously, an off-the-shelf flight controller called the Pixhawk 4 is used. It also

contains an integral accelerometer, gyroscope, magnetometer, and barometer. Paired alongside the Pixhawk is a power-management board to allow for precise measurements of power in flight. An external real-time kinematics (RTK) GNSS is used to measure position. The GNSS can communicate with a RTK base-station element to obtain high-accuracy positional data compared to satellite data alone. An airspeed sensor is mounted on the right-wing near the quarter-span position. Just as with the Pixhawk flight controller, all components are off-the-shelf. A wiring diagram of the full Fox UAV platform can be found in Figure 3.2.

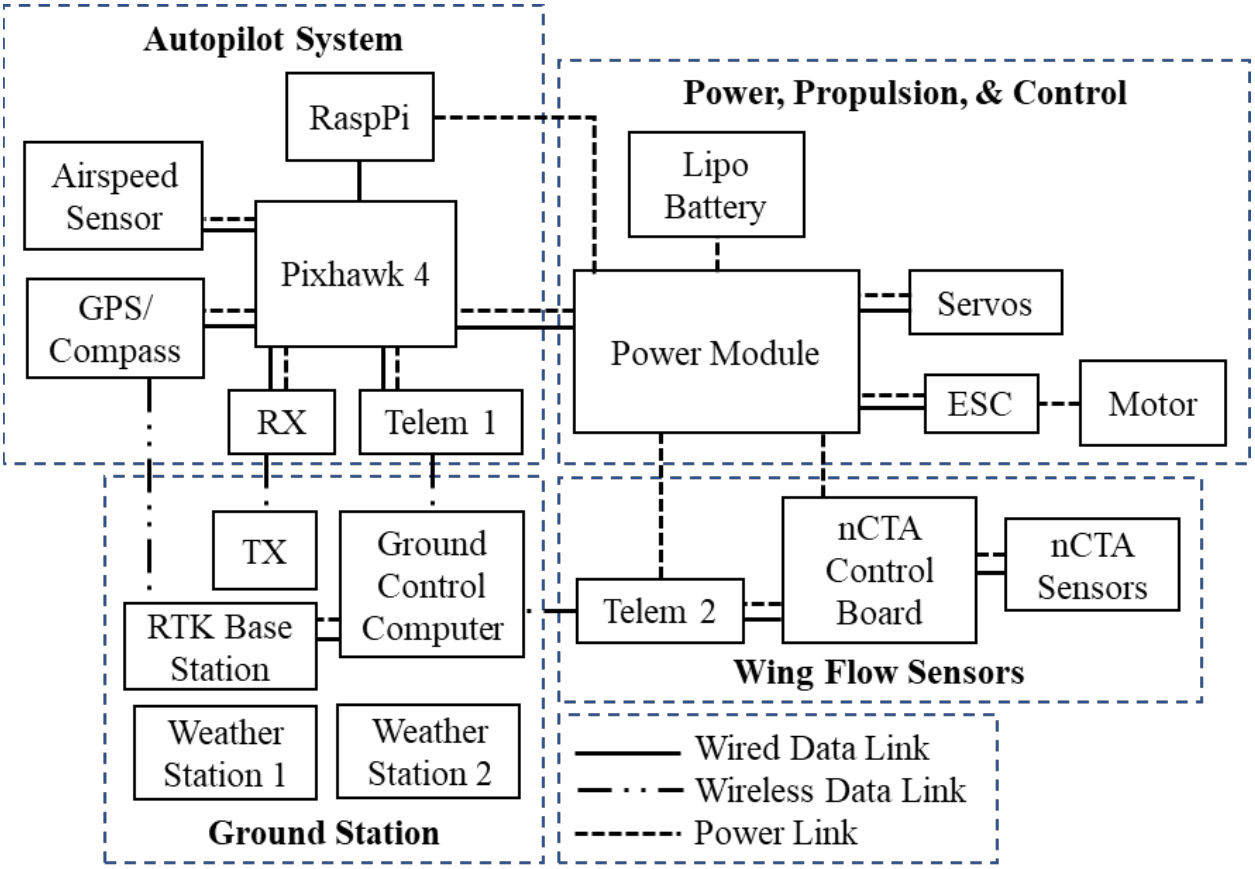

Figure 3.2 Wiring diagram of all components in the Fox UAV platform.

Finally, despite the lack of affordable angle of attack sensors, an experimental thermal anemometry sensor array was used to estimate angle of attack. These sensors must be calibrated

in-situ using wind tunnel data to correlate angle of attack to the sensor data. Development of these sensors to measure angle of attack is ongoing.

The mass properties of all components can be found in Table 3.2. All electronic components are grouped together in "Avionics" with the exception of the motor and battery, which are the heaviest non-airframe components.

Table 3.2 Mass properties of the Fox UAV platform.

Component	**Weight [lb]**	**Weight [kg]**
Motor	0.60	0.272
Battery	2.00	0.907
Spar	0.29	0.132
Left Wing	1.85	0.839
Right Wing	1.85	0.839
HT	0.44	0.200
VT	0.64	0.290
Fuselage	2.86	1.297
Avionics	0.86	0.390
Total	11.39	5.166

3.2 CFD Analysis

Computational fluid dynamics (CFD) is a powerful tool for engineers to evaluate the performance of an aircraft design quickly and conveniently without expensive prototyping and wind tunnel tests. However, for a CFD solution to be trusted as accurate, significant validation must be performed to prove the methods. Validation of all results is not practical for this project. The Fox UAV, with a 3-meter wingspan, is too large to fit inside the Embry-Riddle wind tunnel. To make up for this problem, validation was performed against pre-existing wind tunnel data for the Fox's airfoil.

The CFD method employed in this analysis is known as Reynold's averaged Navier-Stokes (RANS). RANS is a popular method for modeling turbulent flows which decomposes the flow properties into both mean and time-averaged components. A byproduct of this methodology is the

introduction of the Reynold's stress term which cannot be easily computed. To complete the RANS equations, engineers and physicists have derived several turbulence "closure" models that capture the effects of turbulence to varying degrees of accuracy. The use of a particular turbulence model can have a great effect on the accuracy of a solution. A simpler model requires less computational power to implement but may not accurately represent the turbulence in all applications. Particularly for the application of reinforcement learning to dynamic soaring, it is crucial that the CFD solution accurately predicts the aerodynamic coefficients at various flight conditions.

3.2.1 2D CFD Setup

Before conducting a CFD analysis on the Fox UAV, a 2-dimensional CFD analysis of the Fox's airfoil was performed for validation purposes. A Clark-Y is used as the main airfoil for CFD purposes (Figure 3.3). Again, the Clark-Y is used as an approximation of the real Fox's airfoil. Because it is an extremely popular airfoil for low-Reynold's number aircraft, a significant pool of experimental aerodynamic data exists for comparison to CFD predictions. The goal of this analysis is to determine the best turbulence model to apply to the 3-dimensional CFD analysis on the Fox UAV.

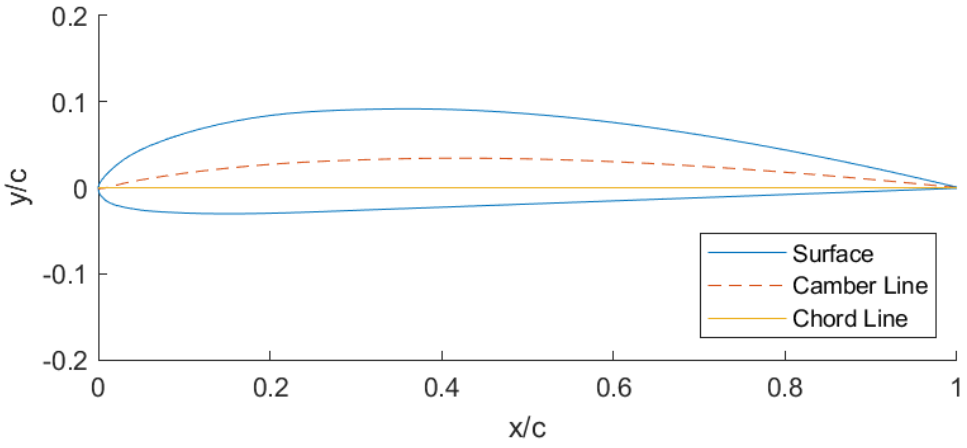

Figure 3.3 Clark Y airfoil.

The experimental data from the University of Illinois at Urbana-Champaign (UIUC) Low Speed Airfoil Tests (LSAT) [12] contains lift, drag, and pitching moment coefficients for several Reynolds flows under 500,000, which is representative of the flight conditions experienced by the Fox UAV. This data set can be seen in Figure 3.4.

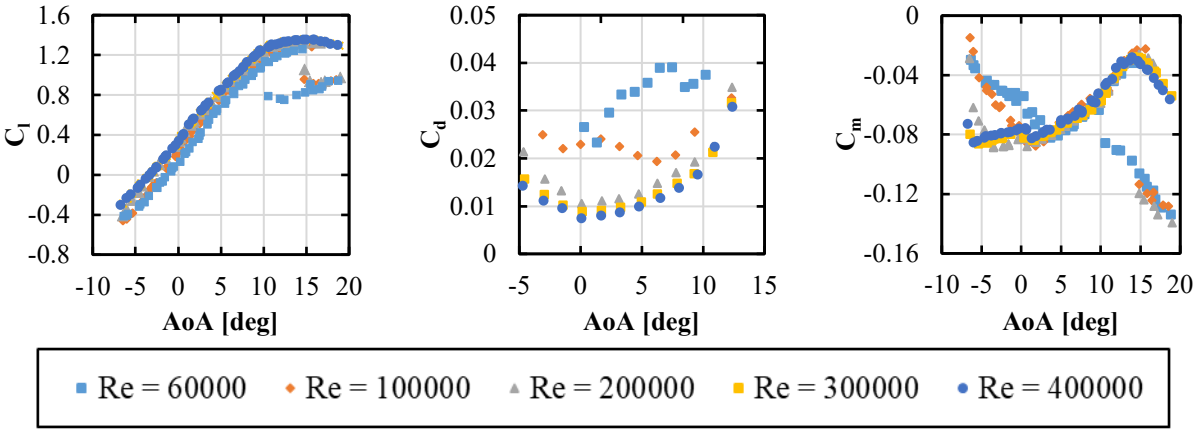

Figure 3.4 UIUC LSAT wind tunnel data for the Clark Y airfoil at various Reynolds numbers [12].

Examining the lift curves, the airfoil has poor performance at Reynolds number of 60,000, 100,000, and 200,000. As the angle of attack is increased, the flow over the airfoil will eventually separate from the surface. As the angle of attack decreases, the flow should eventually re-attach to the surface. However, at low speeds the flow does not immediately re-attach causing a severe drop in lift performance as can be seen in the lift curve in Figure 3.4. Additionally, the drag coefficient is significantly higher in these cases, perhaps because of a larger laminar boundary layer comparatively. Ultimately, the data at a Reynolds number of 300,000 was chosen for the CFD comparison because it was the lowest Reynolds number where the lift, drag, and pitching moment coefficients are consistent, even at high angles of attack. This Reynolds number was used to determine the flow properties for all subsequent CFD analyses.

With the experimental data for verification and corresponding Reynolds number chosen, the atmospheric properties required for CFD can be calculated. These are summarized in Table 3.3,

which also includes the reference chord length (MGC) and wing area of the Fox. These numbers were used in both 2D and 3D CFD analyses. Additionally, the atmospheric properties for a typical day at sea level were used to calculate the flow properties.

Table 3.3 Atmospheric, geometric, and flow properties used in both 2D and 3D CFD analyses.

VARIABLE	SYMBOL	METRIC		US	
Reynolds Number	Re_x	300000	-	300000	-
Wing Area	S	0.7465	m^2	8.0353	ft^2
Chord Length	c	0.257	m	0.8432	ft
Altitude	h	0	m	0	ft
Temperature	T	300	K	80.33	°F
Pressure	P	101325	Pa	2116.2	lb/ft^2
Speed of Sound	a	347	m/s	1139.2	ft/s
Density	ρ	1.1768	kg/m^3	2.28E-03	$slugs/ft^3$
Dynamic Viscosity	μ	1.85E-05	kg/m*s	3.86E-07	slugs/ft*s
Velocity	U	18.313	m/s	60.08	ft/s
Mach Number	M	0.05274	-	0.05274	-

Lastly, a 2D mesh must be created to perform the CFD analysis upon. The meshing software Pointwise was used to generate a mesh for the Clark Y. Pointwise allows users to combine the benefits of structured and unstructured cell elements using a proprietary meshing method called "T-Rex" mesh. In "T-Rex" mesh, rectangular structured grid elements are placed along the surface of the target object. These structured rectangular cell elements are designed to accurately model the viscous boundary layer along an object's surface. The cell spacing normal to the object's surface is very small to capture the boundary layer and grows further away from the surface to optimize the total number of mesh elements. At a certain point, the mesh transforms into unstructured mesh elements to fill in the rest of the mesh that does not require such extreme detail. Again, optimizing the total number of elements.

The mesh used in this analysis can be seen in Figure 3.5. With a mesh generated and freestream flow properties derived for a Reynold's number of 300,000, the CFD analysis can be performed.

Again, the objective of the CFD analysis on the airfoil is to determine the best turbulence model to apply to the CFD on the full Fox model. The best turbulence model is one which matches the experimental data the closest. The full settings of the mesh and CFD for the 2D case can be found in Table 7.1 in the Appendix.

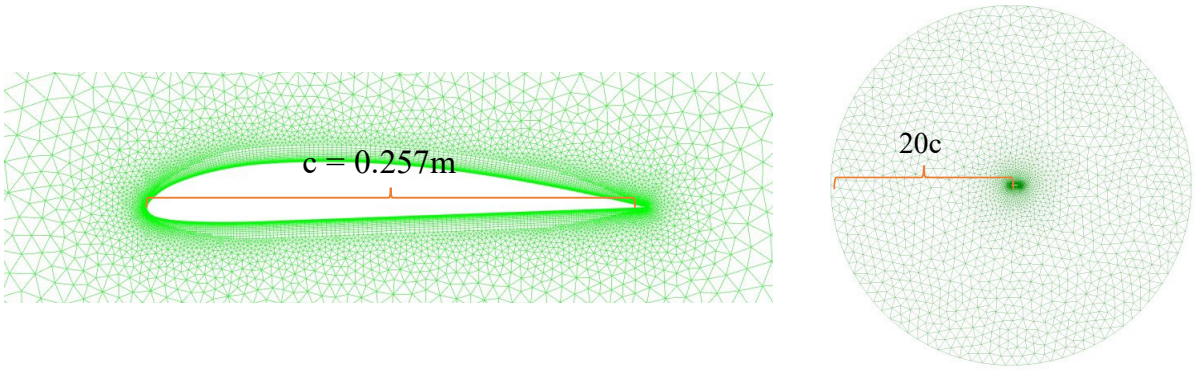

Figure 3.5 Nearfield (left) and farfield (right) grid for Clark Y airfoil generated in Pointwise.

3.2.2 2D CFD Analysis

Before obtaining the results, the resolution of the mesh must be analyzed. Too coarse of a mesh, particularly in regions of turbulent flow such as the boundary layer and separated flow, will hurt the accuracy of the solution. Ideally, a CFD solution should be mesh independent, meaning that the mesh is fine enough that increasing the number of mesh elements will not change the solution.

Such a mesh independence test can be performed in several ways. Typically, the number of mesh elements are doubled twice, and the solutions are compared. Of course, not all elements hold equal importance to the solution. It is critical that the detail of the mesh is improved around particularly turbulent flow features. Doubling the resolution of the far field elements, for example, would be a poor application of this method because most of the flow in this region should be ideal.

While Pointwise was used to discretize the mesh, ANSYS Fluent was used to compute the CFD solution. Fluent has a tool called mesh adaptation which enables the refinement, or

coarsening, of mesh elements as a function of the flow properties. This allows for high precision mesh adjustments. Thus, the mesh refinement tool was applied around the regions with the highest gradient of total pressure. This results in refined mesh elements around the boundary layer and wake only. The mesh elements containing more than 0.5% of the gradient of total pressure were selected and can be seen in Figure 3.6. Moreover, Figure 3.7 is a comparison of the adapted mesh elements at the trailing edge for both adaptation cases. The total number of mesh elements was doubled twice to prove mesh independence. Table 3.4 includes the results of this test. All three cases were evaluated using the flow properties in Table 3.3 at 5 degrees angle of attack. There is no significant difference in solutions between the original or either of the refined meshes. The solution is clearly mesh independent.

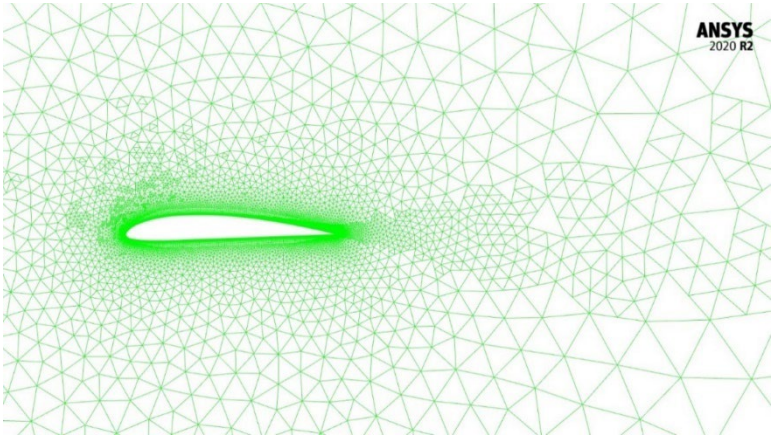

Figure 3.6 Clark Y mesh after applying the Fluent mesh adaptation tool.

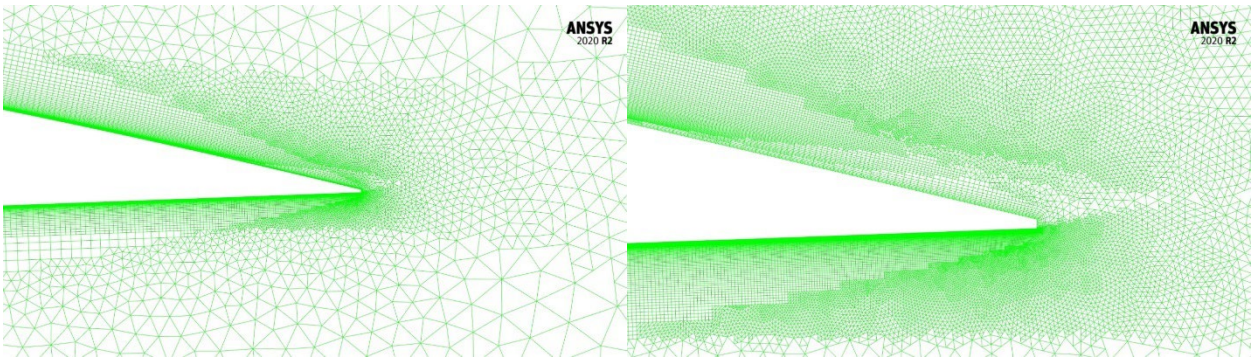

Figure 3.7 Clark Y trailing edge mesh for the first (left) and second (right) adaptation cases.

Table 3.4 Results of the 2D mesh adaptation analysis.

AoA [deg]	Mesh #	C_d	C_l	C_m
5	22884	0.01475	0.84692	-0.06678
	49798	0.01483	0.84668	-0.06678
	155977	0.01485	0.84490	-0.06640

Figure 3.8 shows the results of the Clark-Y CFD analysis. Several turbulence models were tested and compared to the UIUC data at a Reynold's number of 300,000. The effects on the CFD prediction of employing K-Omega, K-Epsilon, and Spalart-Allmaras as the turbulence model are evaluated. Spalart-Allmaras is a one-equation turbulence model which solves for a single transport equation for a viscosity-like variable. K-Omega and K-Epsilon are both two-equation turbulence models in which the scale and energy of the turbulence are defined. In both models, the first equation is used to solve for the turbulent kinetic energy. Regarding the second equation, K-Epsilon solves for the turbulent dissipation and K-Omega solves for the specific turbulent dissipation rate which defines the turbulence scale. Two-equation turbulence models are generally considered to provide better CFD predictions while one-equation turbulence models are less expensive computationally. Additionally, variations of both two-equation models are also compared.

It is apparent, examining the results in Figure 3.8, that both K-Epsilon and K-Epsilon Realizable significantly overestimate the drag compared to the wind tunnel data. Additionally, the K-Epsilon models appear to underestimate lift at high angles of attack. In contrast, the K-Omega variations are a far better prediction of lift and drag. Between the two variations, standard K-Omega is almost a perfect prediction of lift to an angle of attack of 15 degrees, while the lift curve of the low Reynolds number variation drops off sharply after stall. In the case of Spalart-Allmaras, the lift and drag curves are reasonably close to the experimental data. However, the maximum

magnitude of the lift coefficient is overestimated. In all cases, the moment prediction appears to be poor. It is likely, however, that there is a slight deviation between the chord location from which the moment is calculated about between the wind tunnel model and CFD, causing the difference in slope. Ultimately, K-Omega SST was chosen as the best turbulence model for the 3D case since the lift prediction is the closest to the experimental data beyond the point of stall.

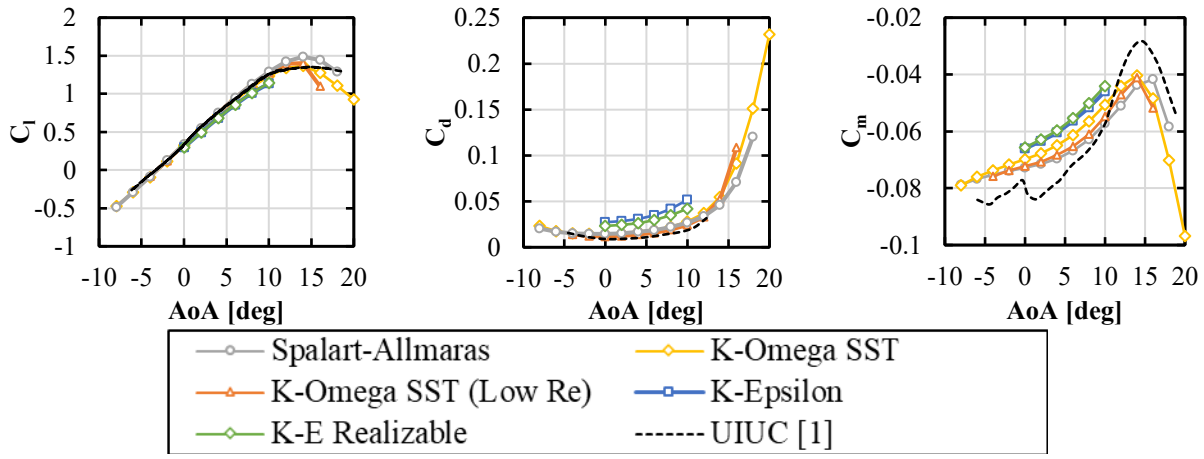

Figure 3.8 Results of the 2D CFD analysis comparing several turbulence models to wind tunnel data.

3.2.3 3D CFD Setup

In contrast to the Clark-Y airfoil CFD analysis, there is no experimental data to compare against the Fox UAV CFD. This makes validation of the subsequent aerodynamic model difficult, but the quality of the solution can still be evaluated using similar methods as the 2D case. Nevertheless, there is one real world validation that is possible. Rather than performing a traditional wind tunnel test, the resulting lift-to-drag ratio from CFD can be compared to the glide slope of the real Fox UAV from flight testing. The maximum glide slope is directly proportional to the maximum lift-to-drag ratio. Unfortunately, this type of analysis provides limited feedback. The rest of the aerodynamic model would still need to be validated against to wind tunnel experiments. Due to time constraints, neither validation test was performed. Instead, the quality of the CFD solution will be evaluated through comparison to other theoretical methods.

Using the CAD model from CATIA, a mesh was generated for the Fox UAV in Pointwise using the same methods as the 2D mesh. In this case, T-Rex mesh was applied to the surface of the CAD model along the leading edges of the wing, horizontal tail, and vertical tail to obtain a geometrically smooth leading edge while minimizing the amount of mesh used. This is analogous to the 2D case where the spacing between points on the airfoil was smaller near the leading edge and larger near the mid-chord. A wing with a fully unstructured mesh would require significantly more mesh elements around the leading edge to obtain the same level of detail. The Fox's surface mesh can be seen in Figure 3.9. The remaining of the surface mesh is uniformly unstructured with fine unstructured mesh around the root and tip of the wing, horizontal, and vertical tail, and the tip of the nosecone.

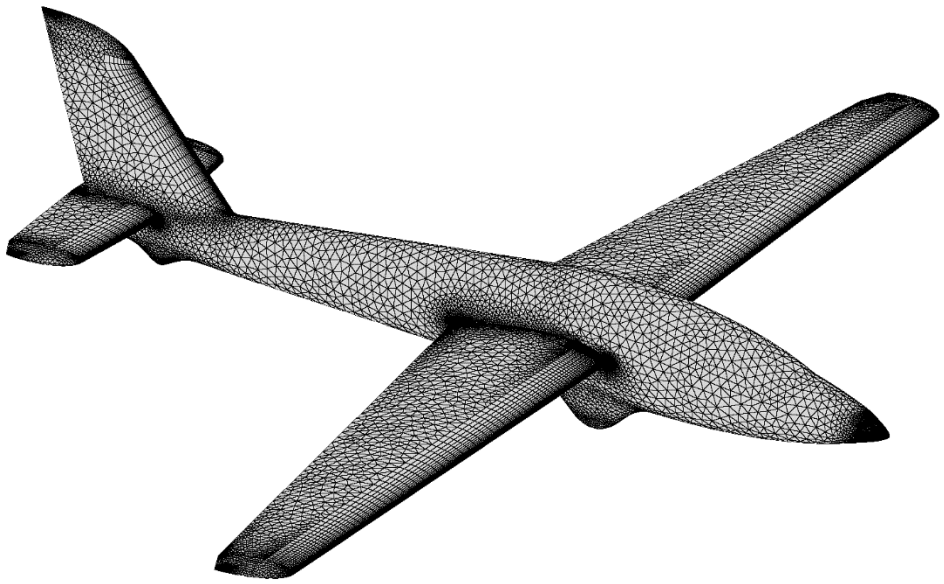

Figure 3.9 Fox UAV surface mesh.

Just as in the 2D case, T-Rex is applied to the mesh elements growing off the surface to capture the boundary layer on the surface of the Fox. The 3D mesh is divided into two components: near-field and far-field to allow for high-resolution around the Fox for solution accuracy and low-

resolution elsewhere to optimize mesh size. The segregation of these elements prevents the T-Rex mesh from growing too far away from the Fox as it cannot penetrate the far-field box. Also, the near-field boundary can be rotated to change angle of attack, sideslip, and roll. Figure 3.10 shows a sectional cut of the final T-Rex mesh around the fuselage and wing. The coloring is used to differentiate different types of mesh elements.

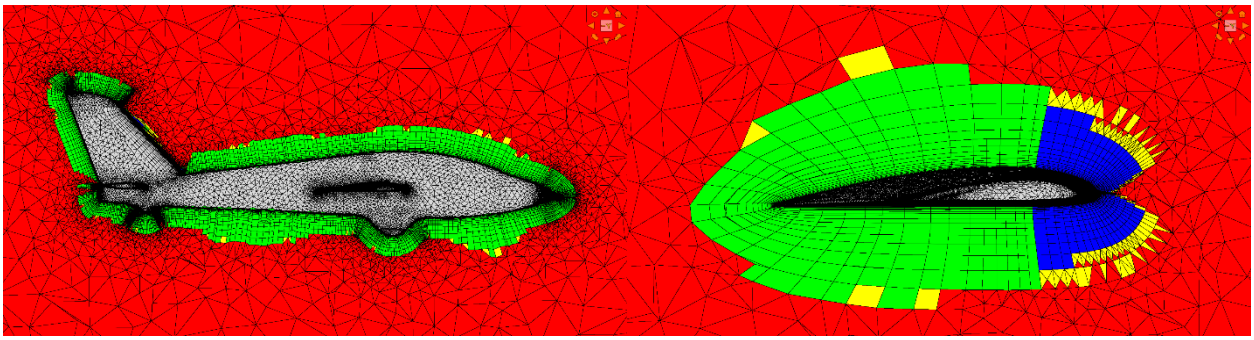

Figure 3.10 Section view of the T-Rex mesh elements surrounding the fuselage and right wing.

Lastly, boundary conditions must be applied to the mesh before being imported into Fluent for analysis. All surfaces of the far-field box require boundary conditions. “Velocity Inlet” is applied to the wall upstream of the Fox. “Pressure Outlet” is applied to the wall downstream of the Fox. “Symmetry” was applied to the remaining four walls on the far-field boundary. Finally, “wall” is applied to the surface of the Fox UAV. The resulting mesh and locations of the boundary conditions are shown in Figure 3.11.

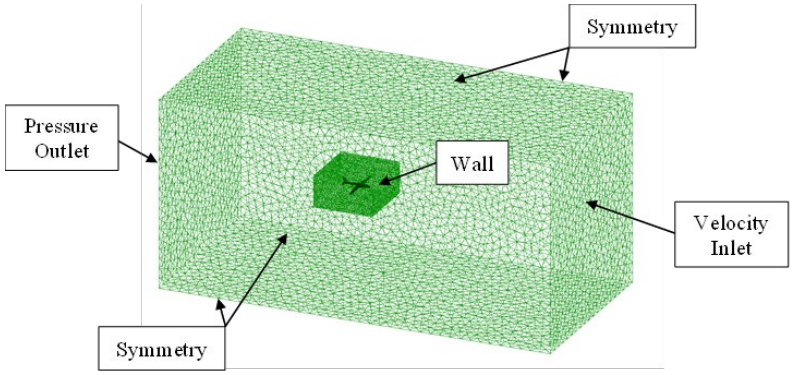

Figure 3.11 Far-field and near-field boundaries surrounding the Fox UAV and boundary conditions.

3.2.4 3D CFD Analysis

A mesh independence analysis was performed to evaluate the quality of the 3D mesh at multiple angles of attack. This was again achieved by applying the Fluent mesh adaptation tool after an initial solution was computed. This was performed at multiple angles of attack to determine if the mesh adequate to capture turbulence and separated flow in three-dimensions. Unlike the 2D case, the mesh size was doubled only once as a consequence of the significant increase in computational cost for the adapted cases. The results can be seen in Table 3.5. Oddly, there is a 10.6% change in the solution of the pitching moment coefficient at 0 degrees angle of attack. This is likely because the initial pitching moment solution was not fully converged since the 4-degree case does not have nearly as much of a difference. The other coefficients have minute differences between mesh levels. Thus, the resulting solution for all three-dimensional cases can be considered mesh independent.

Table 3.5 Results of the 3D grid adaptation test at multiple angles of attack.

AOA [deg]	**Grid #**	**C_D**	**C_L**	**C_M**	**L/D**
0	1934318	0.03082	0.41456	-0.01982	13.45078
	3707676	0.03038	0.41869	-0.02218	13.78154
% Change		1.45	0.99	10.6	2.40
4	1934550	0.04687	0.77556	-0.07291	16.54824
	3548222	0.04647	0.77691	-0.07133	16.71910
% Change		0.86	0.17	2.21	1.02

Compared to the 2D analysis, the computational cost to obtain a converged solution in three-dimensions is extremely high. This is primarily a function of the mesh size (2,000,000 >> 20,000) but is also affected by the equations required to calculate the solution. Rather than reducing mesh size, simplifications can be applied to the governing equations to reduce computational cost with little effect on the accuracy of the solution.

The Navier-Stokes equations are comprised of the continuity, momentum, and energy equations. The continuity and momentum equations must always be solved. However, the energy equation is optional under specific circumstances. In this case, the flow can be assumed to be incompressible such that the density is a constant. Also, if viscosity is assumed to be a constant as well, the energy equation can be decoupled from the continuity and momentum equations. This allows for the equation to be ignored during computation of the solution, dramatically reducing the computational cost. Despite this, the effect on the solution may not be negligible in all cases. Thus, a comparative analysis between solutions with and without the energy equations must be performed at multiple flow conditions. The results of this analysis can be seen in Table 3.6. There is no significant difference with or without the energy equation. Therefore, the energy equation can be dropped from the RANS equations.

Table 3.6 Comparison of compressible and incompressible K-Omega results.

AoA [deg]	Energy Equation	C_D	C_L	C_M	L/D
0	No	0.0307	0.4155	-0.0164	13.5187
	Yes	0.0310	0.4145	-0.0166	13.3876
% Change		0.73	0.24	1.56	0.98
10	No	0.1117	1.1405	-0.3058	10.2140
	Yes	0.1118	1.1332	-0.3038	10.1332
% Change		0.15	0.64	0.67	0.80

The results of the CFD analysis on the Fox UAV can be found in Figure 3.12. The solution was evaluated from -40 to 40 degrees angle of attack, -40 to 40 degrees angle of sideslip, and -90 to 90 degrees of roll. Angle of attack sweep only has an effect on the longitudinal force and moment coefficients C_L, C_D, and C_M. In the cases where sideslip and roll are 0 degrees, the lateral coefficients C_Y, C_l, and C_N are effectively zero due to the Fox being symmetrical about the longitudinal plane. In contrast, sweeping sideslip and roll affects both the longitudinal and lateral coefficients.

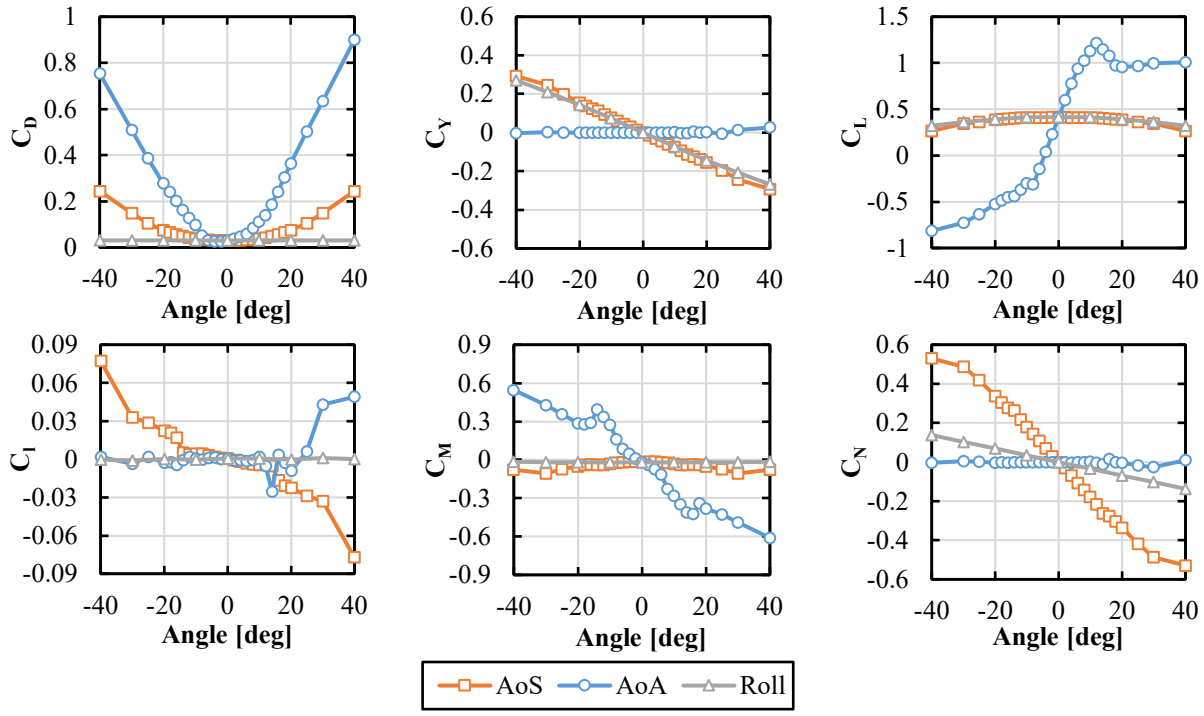

Figure 3.12 Full results of the Fox UAV CFD analysis.

The velocity streamlines and surface pressure contours as a result of sweeping angle of attack and sideslip can be seen in Figures 3.13 and 3.14 respectively. Critical angles were chosen to show the flow at different stages of separation. Additionally, the total pressure contour over the Fox's surface is included. Figure 3.13 shows the progression of flow with increasing angle of attack at 0 AoA, 12 AoA (maximum CL), and 30 AoA (post stall). Figure 3.14 shows the progression of the flow over the Fox as sideslip is increased from 0 to 30 degrees.

To integrate the resulting static force and moment coefficients with the aircraft simulation, the results for each swept angle must be combined into a single coefficient. For the lift coefficient as an example, a different value exists for angle of attack, sideslip, and roll at any point in time. These must be combined such that there is a single lift coefficient as a function of the three angles. To accomplish this, equation 1 was developed.

$$C_{L_{static}} = C_L(\alpha) + C_L(\phi) + C_L(\beta) - 2C_L(0) \quad (1)$$

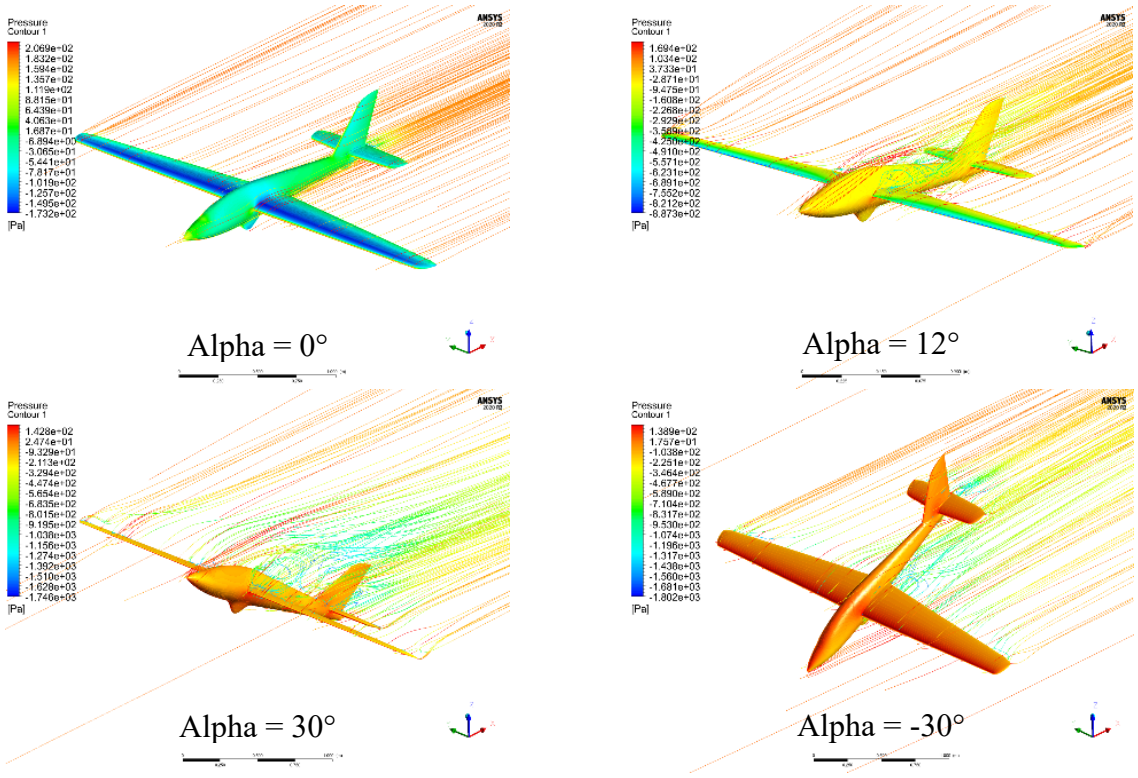

Figure 3.13 Streamlines and surface pressure contours for AoA sweep.

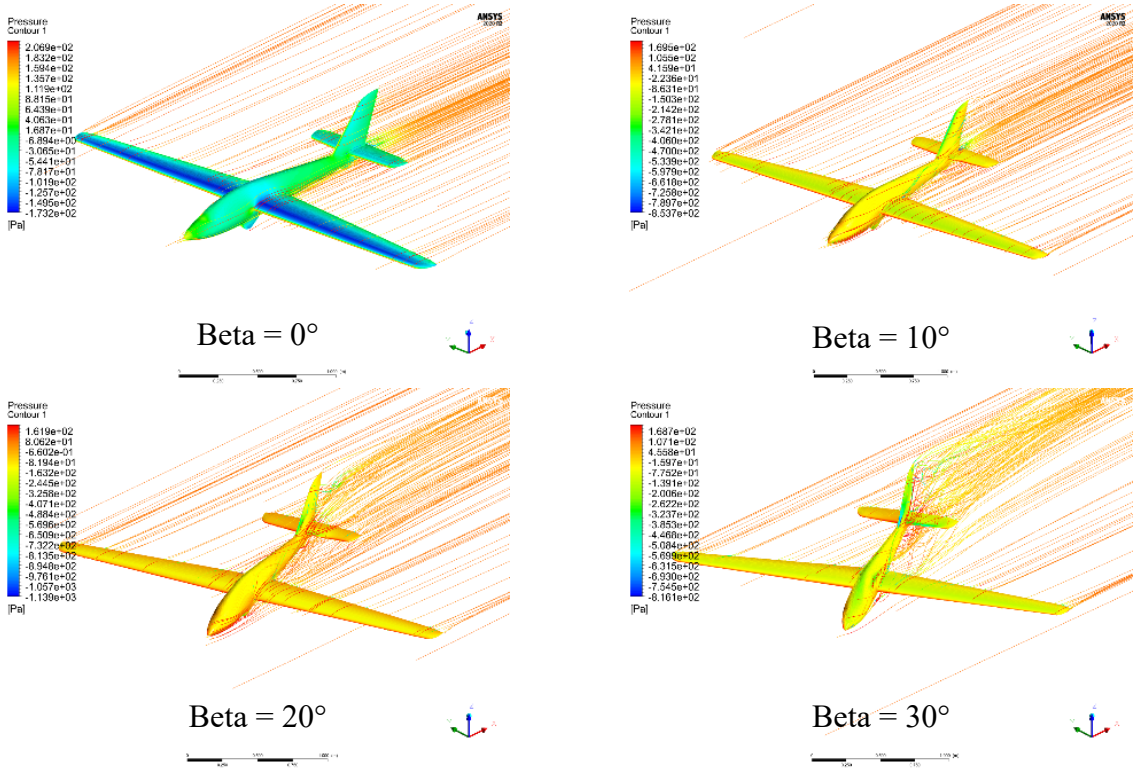

Figure 3.14 Streamlines and surface pressure contours for sideslip sweep.

Equation 1 combines the contributions of all three coefficients through summation and then subtracts away two times the coefficient at steady-level flight (0 degrees angle of attack, sideslip, and roll). This prevents the coefficient from being too large when multiple angles are non-zero. For example, if both angle of attack and sideslip are nonzero, then the lift coefficient should be smaller than if the sideslip was zero. This equation is implemented in the aircraft simulation to convert the three sets of 1-dimensional coefficient lookup tables into one set of 3-dimensional lookup tables.

A simple comparison can be performed to evaluate the usefulness of this method. Three test cases were chosen with varying angle of attack, sideslip, and roll. A unique CFD solution was computed for each case to compare to the output of equation 1. The cases chosen were based upon the results from Gladston Joseph [1]. The human piloted dynamic soaring case was used to select the cases for the analysis because the attitude was not limited in any way unlike the autonomous case. The governing hypothesis is that a human-piloted maneuver more closely reflects dynamic soaring generated by reinforcement learning. Figure 3.15 shows the angle of attack, sideslip, and roll for the manually piloted dynamic soaring case. From this figure, the angle of attack reaches a maximum of about 10 degrees for a normal DS cycle, the sideslip varies between 15 and -20 degrees, and the roll varies from 0 to 90 degrees. Given this, special cases can be chosen to perform a CFD analysis to compare the static coefficients to the lookup table output.

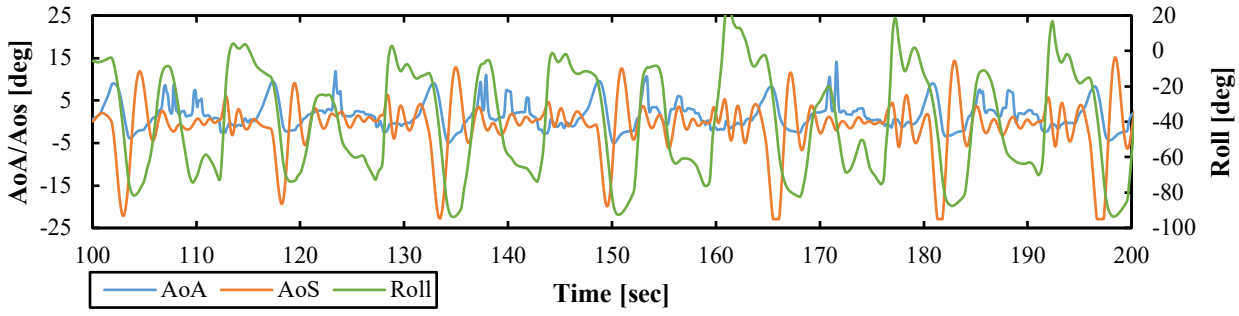

Figure 3.15 Human-piloted dynamic soaring angle of attack, sideslip, and roll data [1].

The subsequent dynamic soaring cases to evaluate the lookup tables are shown in Table 3.7. The lookup tables should perform best for Case 1 because it has a high angle of attack, low sideslip, and zero roll angle. It should have only slightly more error than if sideslip was zero. Case 2 has a high angle of attack and sideslip which will have larger error. Case 3 is the worst-case scenario for equation 1 with three large angles.

Table 3.7 Definition of the special dynamic soaring cases.

Case	AOA [deg]	Beta [deg]	Roll [deg]
1	10	3	0
2	10	15	0
3	10	15	30

Tables 3.8 and 3.9 show the results of the static aerodynamic lookup table analysis. The longitudinal force and moment coefficients (lift, drag, and pitching moment) exhibit the expected behavior where the error is larger as the angles increase. However, the lateral force and moment coefficients (side force, rolling moment, yawing moment) have large errors in all cases. This analysis reveals that equation 1 is not a good estimation of the true aerodynamic model. To prevent this error in the future, a fully defined two-dimensional lookup table based upon angle of attack and sideslip should be developed.

Table 3.8 Comparison of CFD and LUT force and moment results for each DS case.

Case		1	2	3
C_D	**CFD**	0.10670	0.12011	0.24711
	LUT	0.11123	0.13313	0.13268
	% Diff	4.24	10.84	46.31
C_Y	**CFD**	-0.00869	-0.09524	-0.57025
	LUT	-0.02185	-0.11870	-0.32700
	% Diff	151.42	24.62	42.66
C_L	**CFD**	1.15092	1.15552	0.96511
	LUT	1.12785	1.11495	1.06095
	% Diff	2.00	3.51	9.93

Table 3.9 Comparison of CFD and LUT force and moment results for each DS case.

Case		1	2	3
C_l	**CFD**	-0.00886	-0.02105	0.30850
	LUT	-0.00066	-0.01027	-0.01009
	% Diff	92.51	51.22	103.27
C_M	**CFD**	-0.25769	-0.23303	-0.28701
	LUT	-0.27792	-0.29947	-0.29697
	% Diff	7.85	28.51	3.47
C_N	**CFD**	-0.01929	-0.27966	-0.25452
	LUT	-0.04998	-0.27080	-0.28099
	% Diff	159.09	3.17	10.40

Not only was equation 1 determined to be a poor representation of the aerodynamic model, but the roll angle aerodynamic coefficient lookup table is completely incorrect. Through simulation, it was revealed that rolling the UAV in any direction would create an unreasonably large yawing force. This unusual force would cause the UAV to gain altitude as a result which does not obey conventional aeronautical logic. The roll component of the static aerodynamic lookup tables was ultimately found to be the culprit. Further review of the CFD methods for roll reveal that the forces and moments were not computed around the correct axes relative to the flow. For the wind reference frame, which the aerodynamic forces and moments are related to, when the aircraft rolls, so does the axes of the aerodynamic forces and moments. The lift vector should be orientated perpendicular to the flow as well as the wings. In the CFD solution, this vector was not perpendicular to the wings.

To prove this point, analyzing the side force plot in Figure 3.12, the side force at 90 degrees roll is about 0.4. Considering the axis this was measured about did not roll with the aircraft, the side force at 90 degrees of roll is the same as the lift force at zero degrees of roll. The lift coefficient at zero degrees of roll is also equal to 0.4. So as the UAV rolls in the simulation, lift from the wings transforms into side force. If the axes were correct in the roll CFD solution, there would be no variation of any of the forces or moments. Consequently, in the final versions of the simulation,

the static aerodynamic lookup tables are reduced such that the longitudinal forces and moments are only a function of angle of attack and the lateral forces and moments are a function of only sideslip. This eliminates the need for any special equation, such as equation 1, for the static aerodynamic lookup tables.

3.3 Surfaces VLM

Using computational fluid dynamics, the static aerodynamic model of the Fox UAV was obtained. The static aerodynamic coefficients can define the static forces and moments of an aircraft at an instance in time. However, for an aircraft simulation the dynamic behavior also needs to be defined. Even if the static aerodynamic data indicates that the aircraft is statically stable, dynamic instabilities may exist. An aircraft can be both stable statically and unstable dynamically, thus, establishing the need for a dynamic stability analysis. Also, the true aerodynamic forces and moments acting upon the aircraft are a function of both static and dynamic contributions. Computational methods can be used to obtain these dynamic stability derivatives as well as the control surface coefficients.

While computational fluid dynamics can be used to directly calculate the static stability coefficients in high fidelity, it cannot be used to directly obtain the dynamic stability derivatives and control surface coefficients. These are more easily obtained using a method known as vortex lattice method (VLM). A software known a Surfaces, a VLM solver, will be used to obtain the dynamic aerodynamic model.

VLM is considered “low-fidelity” in comparison to a “high-fidelity” method such as RANS because it neglects both viscosity and geometrical thickness effects. As a result, VLM solvers do not model flow separation and are not accurate near or after the point of stall. However, this does not mean that results obtained using VLM are unreliable. In fact, they can obtain accurate results pre-stall so long as an accurate panel-based representation of an aircraft can be modeled. Figure

3.16 shows the Surfaces model of the Fox UAV. VLM works by generating a sheet of vortices to model lift and induced drag. Pressure on a panel can be derived from the vortices, which can then be used to calculate aerodynamic forces and moments. In Surfaces VLM, both the lifting surfaces, such as the wings and empennage, and the fuselage must be modeled to obtain a reasonable prediction for the dynamic aerodynamic coefficients.

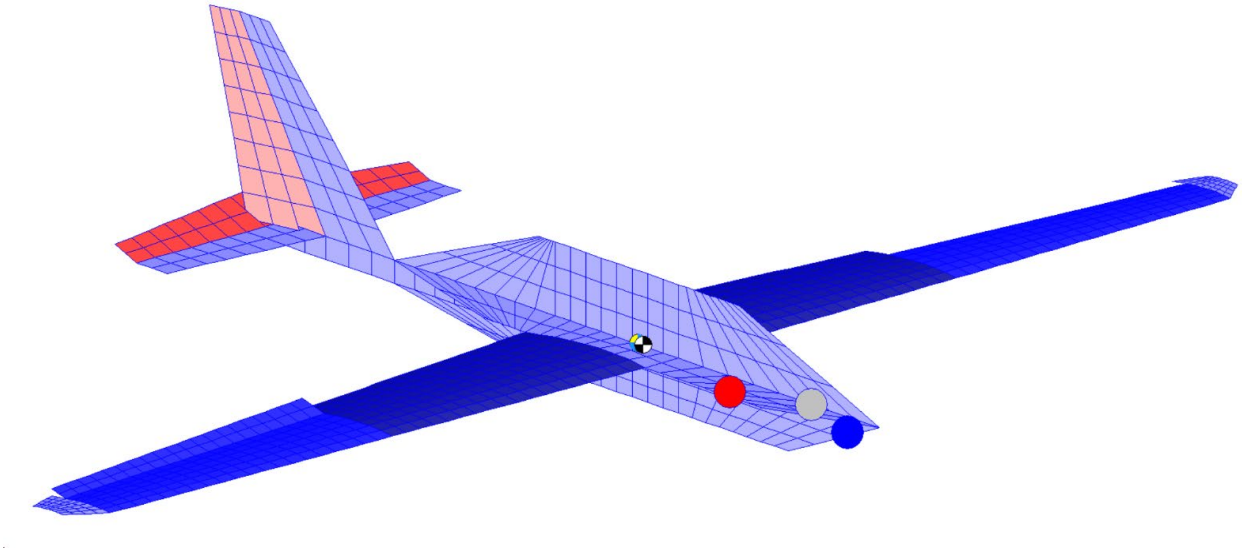

Figure 3.16 Surfaces model of the Fox UAV.

While the VLM model of the Fox looks primitive in comparison to the CAD model, it is debatably a more accurate model of the physical Fox UAV. This is because not only is the geometry of the airframe modeled, the VLM Fox includes control surfaces and an accurate mass model. The control surfaces including the ailerons, flaps, elevator, and rudder are modeled to obtain the control surface coefficients for control in the simulation. Another important component of the VLM model is the mass. The Fox is split into several components which all have individual masses. Both the left and right wings, fuselage, horizontal stabilizer, and vertical stabilizer have mass which was measured from the real Fox UAV. Additionally, the non-airframe components are also modeled to ensure the VLM Fox model's weight distribution is as accurate as possible. This

includes the motor, battery, and all avionics as point-masses. These masses were arranged in such a way as to place the center of gravity at 37.5% of the mean geometric chord. The masses of all components can be seen in Table 3.2.

Similar to the high-fidelity CFD, verification needs to be performed to validate the resulting VLM solution. There are two analyses that can be used to accomplish this. The first, is to directly compare the static coefficients between the CFD and VLM methods. While VLM cannot predict an accurate solution post-stall, the pre-stall solution should be reasonably close to the CFD solution. This analysis is shown in Figure 3.17 where lift, drag, and pitching moment coefficients are compared.

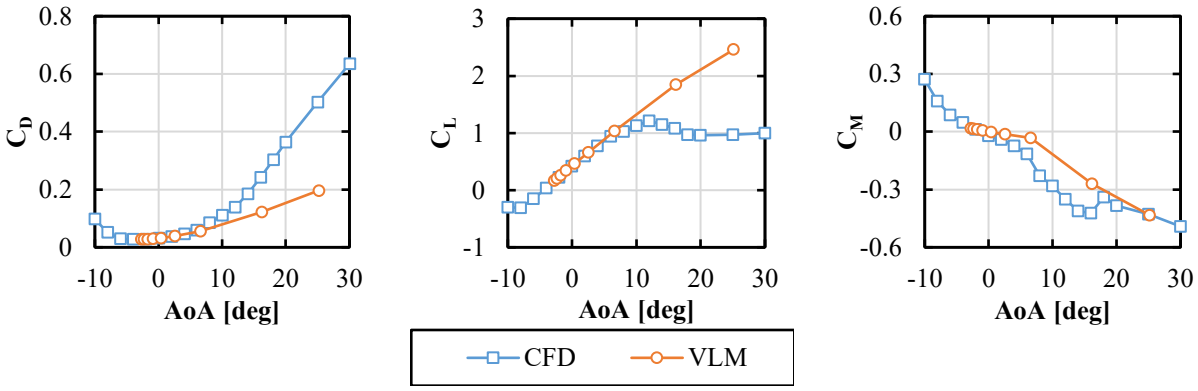

Figure 3.17 Comparison of VLM and CFD longitudinal solutions.

Evaluating these plots shows that the VLM model agrees strongly with the CFD solution up to roughly 8 degrees angle of attack. This is exactly as predicted. It should be noted that for the drag, while VLM can only directly calculate the vortex induced drag, Surfaces allows for the drag to be supplemented by external data. In this case, the drag coefficient at 0 angle of attack and skin-friction drag from other sources can be combined with the induced drag. Additionally, Surfaces uses a method to calculate induced drag which is a function of the lift coefficient at minimum drag. Both the standard drag coefficient and lift coefficient at minimum drag are obtained from the CFD

solution. Since the solutions agree, the dynamic aerodynamic model can be evaluated with high confidence.

A second analysis can be performed to compare the solution to the physical Fox UAV. The manufacturer of the Fox states that the ideal center of gravity (CG) margin should be between 90 to 95 millimeters behind the leading edge of the wing. In addition, model aviation hobbyists who have flown the Fox UAV claim that this range can be expanded up to 110 millimeters for improved pitch performance. Thus, if the VLM model is true to the real Fox, then it should be dynamically stable with the CG located within that margin. Also, it can be inferred that the neutral point (AC) should be located just behind that margin, since the CG needs to be in front of the AC for stability. The internal components in the Fox were arranged such that the CG is 100 millimeters behind the leading edge or at 33% of the mean geometric chord. The neutral point was determined to be located 113 millimeters behind the leading edge of the wing, which falls just outside of the alleged CG margin as expected. To evaluate the stability of the Fox with this CG and AC configuration, a dynamic stability analysis must be performed.

Surfaces has a built-in tool that automatically evaluates the dynamic stability both longitudinally and laterally. The stability results can be seen in Figures 3.18 and 3.19.

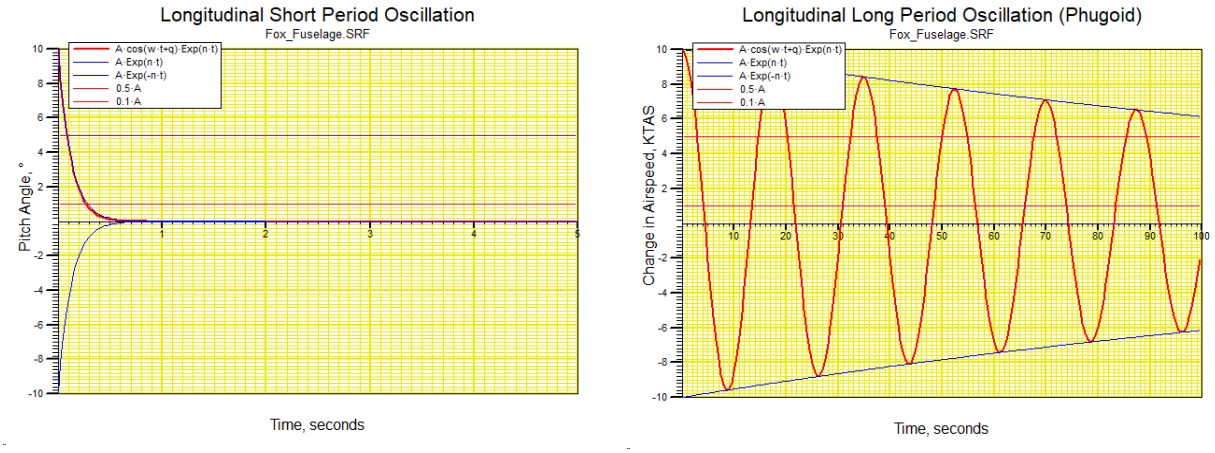

Figure 3.18 Longitudinal dynamic stability plots.

Figure 3.18 shows the longitudinal short period and long period stability plots. Both periods are decreasing over time and are dynamically stable. The short period oscillation evaluates the pitch angle over time. It quickly damps out to zero within 1 second from its initial perturbation. On the other hand, the long period, which is the change in airspeed as a result of varying the pitch angle, takes significantly longer to damp out. This low phugoid damping is typical of glider-like aircraft such as the Fox.

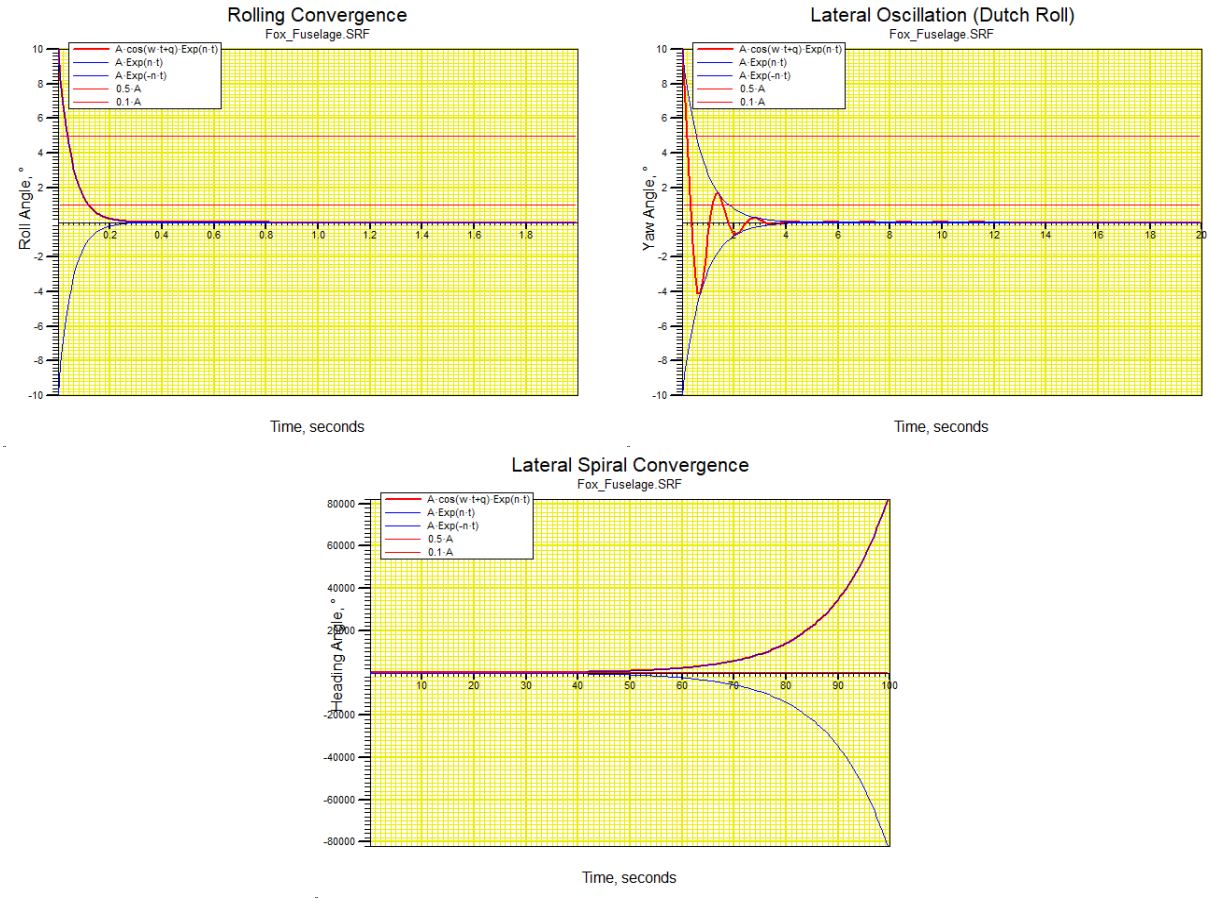

Figure 3.19 Lateral dynamic stability plots.

Figure 3.19 shows the lateral dynamic stability plots including the rolling convergence, lateral oscillation (Dutch Roll), and lateral spiral convergence. To top left graph in Figure 3.19 shows the rolling convergence which, like the longitudinal short period, is the damping of the roll angle after a perturbation. It is heavily damped and completely dissipates within a quarter of a second. The

top right graph in Figure 3.19 is the lateral oscillation, or "Dutch Roll". This value is a measure of the coupling between sideslip, yaw, and roll. This is heavily damped for the Fox and the oscillations are effectively zero within five seconds. Finally, the bottom graph in Figure 3.19 is the lateral spiral convergence. If the pilot does not give any input in a banked turn, a stable spiral mode will return the wings to level over time and an unstable spiral mode will increase the bank angle. For the Fox this mode is unstable and grows over a significant amount of time. This is not a substantial issue since a pilot can easily counter this negative effect by simply taking control. Likewise, in autonomous flight, this instability is a non-issue. From the data in both Figures 3.18 and 3.19, the Fox is dynamically stable.

With the Fox proven to be dynamically stable when the CG and AC are in the same area as the physical Fox and the static VLM solution proven to agree with the CFD solution, the dynamic stability coefficients can be obtained. Figure 3.20 shows the dynamic damping derivatives varied by angle of attack. These plots are grouped by their respective angular rates p, q, and r. The angles of attack that the stability derivatives were computed at were determined by varying the airspeed. The geometry of an aircraft changes based on the flight conditions. At low-speed and high-angle of attack, the elevator must be deflected to a large angle to trim the aircraft for steady-level-flight. At high-speed and low angle of attack, the elevator is deflected to a lesser angle. Because only angle of attack was swept in the solution, the derivatives for roll and yaw are comparatively small to the pitch derivatives. However, only the roll damping with respect to roll and yaw rate changes significantly with angle of attack. All other derivatives are nearly constant. The final derivatives and control surface coefficients used for the Fox UAV can be seen in the Appendix.

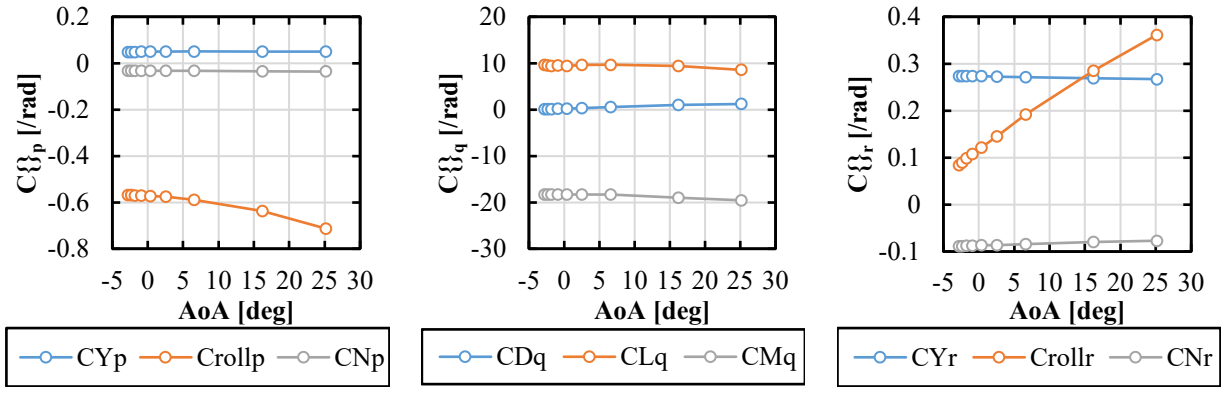

Figure 3.20 Dynamic stability derivatives as a function of angle of attack.

3.4 Flight Simulation

To simulate the Fox UAV, a 6-degree-of-freedom (6 DoF) aircraft simulation was developed using MATLAB and Simulink. MATLAB is a high-level programming language with a set of toolboxes useful for reinforcement learning. Simulink is a block diagram programming environment which can be used to create time-based multi-rate models. This optimizes the programming of the aircraft dynamics model compared to simple lines of code because Simulink automatically handles all time-based operation such as derivatives or integrals.

The Fox UAV simulation was created combining the advantages of both MATLAB and Simulink. In MATLAB, the aerodynamic lookup tables from CFD and VLM were programmed into a script. This script initializes the aerodynamic model, geometric and inertial reference values, and aircraft states. This information is required to define the initial state of the simulation. The Simulink model can be seen in Figure 3.21. It is comprised of three primary components. The first is the Aircraft Dynamics block which contains the aerodynamic lookup tables, 6-degrees-of-freedom (6 DoF) aircraft equations of motion, as well as the wind shear model needed for dynamic soaring. Additionally, it computes useful properties from the current state of the UAV. The second is the RL Autopilot which is the block that performs reinforcement learning. It directly interfaces with an external MATLAB script to perform the training and store experience. The third is the

Pilot and AP (autopilot) Inputs block. This block allows for a human pilot to control the UAV using an external joystick. This is advantageous for debugging the simulation so that the aircraft can be tested at different states without the need for autonomous control. Finally, the Cockpit, Trim Checker, and 3D Animation blocks are to allow real time visualization of the Fox's states. The three main blocks form a closed-loop system allowing for autonomous dynamic soaring simulations.

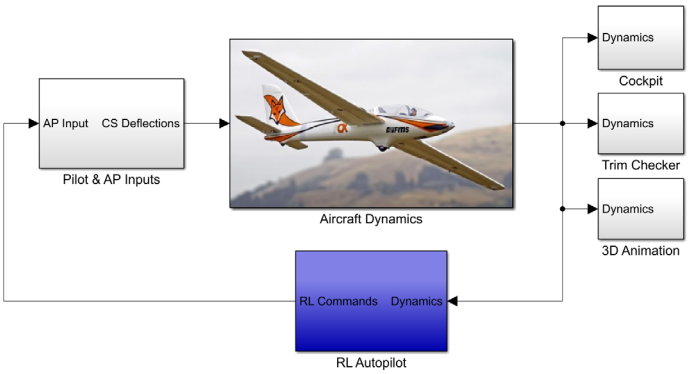

Figure 3.21 Overview of the main components of the Fox UAV simulation in Simulink.

3.4.1 Aircraft Dynamics Block

The aircraft dynamics block contains eight sub-blocks which form the aerodynamic lookup tables and 6-DoF aircraft equations of motion to define the state of the UAV at a point in time. The contents of each block will be covered in detail.

3.4.2 Fox Lookup Tables Block

The purpose of the Fox Lookup Tables block is to define the aerodynamic force and moment coefficients at an instance in time as a function of the static, dynamic, and control surface coefficients. It requires the angle of attack, sideslip, and roll for inputs and returns the corresponding force and moment coefficients.

Inside of this block are three sub-blocks for the static coefficients, dynamic stability derivatives, and the control surface coefficeints respectively. Since equation 1 was proven to be

incapable of replicating 3D lookup tables as a function of all three angles, the static lookup tables were simplified so that lift, drag, and pitching moment are only a function of angle of attack. Side force, rolling moment, and yawing moment are, consequently, only a function of side slip. Thus, this block simply passes through the coefficents from the state initialization script in MATLAB.

The dynamic stability derivative lookup table block performs a similar role. The derivatives are only a function of angle of attack. Again, the values are simply imported from the MATLAB script. The control surface coefficent block also passes through the values to the subsequent block in the Simulink model.

3.4.3 Aerodynamic Block

The aerodynamics block receives the force and moment coefficients from the lookup tables and outputs the resulting aerodynamic forces and moments as a function of the aircraft's current state. This block can be seen in Figure 3.22. The other inputs from the rest of the model are the control surface deflection angles, airspeed, angle of attack, angle of sideslip, and angular rates. Only the force and moment outputs are used for calculating the aircraft's state. The other outputs are for data analysis purposes only.

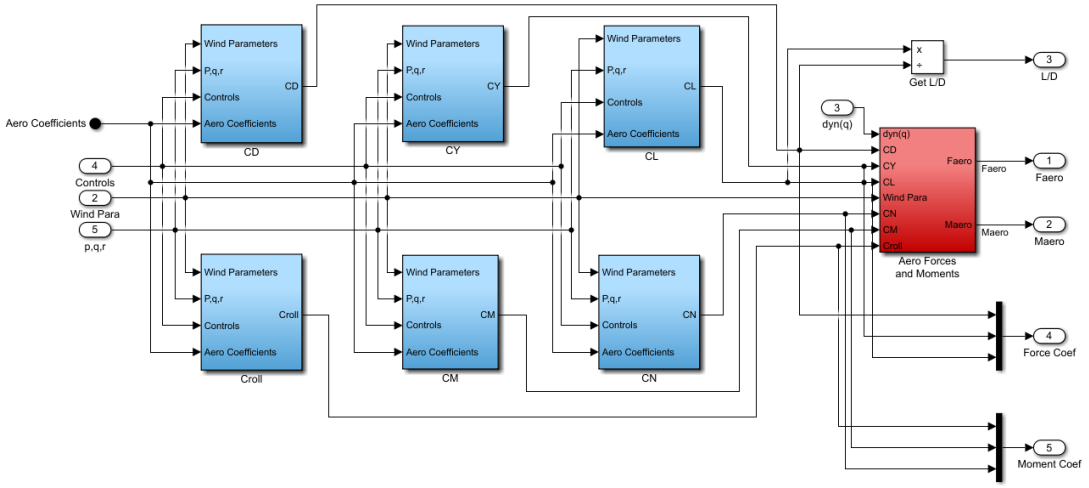

Figure 3.22 Inside the Aerodynamics block.

3.4.4 Aerodynamic Coefficient Formulation

The six blue blocks on the left side of Figure 3.22 are used to calculate the final force and moment coefficients to determine the resulting aerodynamic forces and moments as a function of the static and dynamic contributions. Equations 2 through 7 are the full set of aerodynamic equations. These equations are developed from a larger set of aerodynamic coefficient equations from Aircraft Control and Simulation [13] and have been reduced to fit the Fox's aerodynamic model.

$$C_D = C_{D_{Static}} + C_{D_q}\frac{q\bar{c}}{V} + C_{D_{\delta_e}}\delta_e + C_{D_{\delta_f}}\delta_f \tag{2}$$

$$C_Y = C_{Y_{Static}} + C_{Y_p}\frac{pb}{2V} + C_{Y_r}\frac{rb}{2V} + C_{Y_{\delta_a}}\delta_a + C_{Y_{\delta_r}}\delta_r \tag{3}$$

$$C_L = C_{L_{Static}} + C_{L_q}\frac{q\bar{c}}{V} + C_{L_{\delta_e}}\delta_e + C_{L_{\delta_f}}\delta_f \tag{4}$$

$$C_l = C_{l_{Static}} + C_{l_p}\frac{pb}{2V} + C_{l_r}\frac{rb}{2V} + C_{l_{\delta_a}}\delta_a \tag{5}$$

$$C_M = C_{M_{Static}} + C_{M_q}\frac{q\bar{c}}{V} + C_{M_{\delta_e}}\delta_e \tag{6}$$

$$C_N = C_{N_{Static}} + C_{N_p}\frac{pb}{2V} + C_{N_r}\frac{rb}{2V} + C_{N_{\delta_a}}\delta_a + C_{N_{\delta_r}}\delta_r \tag{7}$$

For equations 2 through 7, the terms with the static subscript are the CFD coefficients, the terms with p, q, or r as subscript are the dynamic damping derivatives, and the terms with a δ subscript are the control surface derivatives. The dimensional terms include the angular rates p, q, and r, the true airspeed V, mean geometric chord $\bar{c}$, the wingspan b, and the control surface angles δ.

3.4.5 Aerodynamic Forces and Moments

The red block in Figure 3.22 calculates the resulting aerodynamic forces and moments. The forces are first calculated in the wind reference frame using equations 8 through 10.

$$D = \bar{q}SC_D \tag{8}$$

$$Y = \bar{q}SC_Y \tag{9}$$

$$L = qSC_L \tag{10}$$

Then, the forces are converted to the body reference frame using equations 11 and 12. Equation 11 is the transformation matrix to convert from wind to body reference frame. This is multiplied with the forces in equation 12 to obtain the resulting forces in the correct reference frame. The forces have to be converted to the body reference to be used in the equations of motion since they will be combined with non-aerodynamic forces.

$$S = S_\alpha S_\beta = \begin{bmatrix} \cos\alpha\cos\beta & \sin\beta & \sin\alpha\cos\beta \\ -\cos\alpha\sin\beta & \cos\beta & -\sin\alpha\sin\beta \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} \tag{11}$$

$$F_{B_{Aero}} = S^T * \begin{bmatrix} -D \\ Y \\ -L \end{bmatrix} \tag{12}$$

The moments do not require such a transformation and can be directly applied in either reference frame. Equations 13 through 16 are used to calculate the moments from the moment coefficients in this block.

$$l = \bar{q}SbC_l \tag{13}$$

$$M = \bar{q}SbC_M \tag{14}$$

$$N = \bar{q}SbC_N \tag{15}$$

$$M_{B_{Aero}} = \begin{bmatrix} l \\ M \\ N \end{bmatrix} \tag{16}$$

3.4.6 Linear Acceleration and Moments

The Linear Acceleration and Moments block calculates the total forces and moments as a result of non-aerodynamic contributions. For the total body force, the thrust force produced by the

engines must be added to the aerodynamic forces as in equation 17. This force is then converted into linear acceleration by simply dividing by the mass of the UAV in equation 18.

$$F_B = F_{B_A} + F_{B_T} \tag{17}$$

$$a = \frac{F_B}{m} \tag{18}$$

Similarly, the total moment on the UAV is a result of more than just aerodynamic moments. There is also a moment generated by the offset of the aerodynamic center to the center of gravity which is shown in equation 19. Also, a moment is generated by the engine if it is offset to the vertical position of the center of gravity as in equation 20. All three moments are summed together in equation 21.

$$M_{B_{AC}} = [AC - CG] \times F_{B_A} \tag{19}$$

$$M_{B_{Eng}} = [Eng - CG] \times F_{B_T} \tag{20}$$

$$M_B = M_{B_{Aero}} + M_{B_{AC}} + M_{B_{Eng}} \tag{21}$$

3.4.7 Equations of Motion and Numerical Integration

With the total forces and moments acting on the UAV calculated, they can be used to solve the 6-DoF aircraft equations of motion to derive the state at an instance in time. This will provide important quantities such as attitude, angular rates, angular accelerations, airspeed, and groundspeed. An overview of this block's contents can be seen in Figure 3.23. All equations contained within that are related to the aircraft equations of motion are taken from Aircraft Control and Simulation [13] unless specified otherwise.

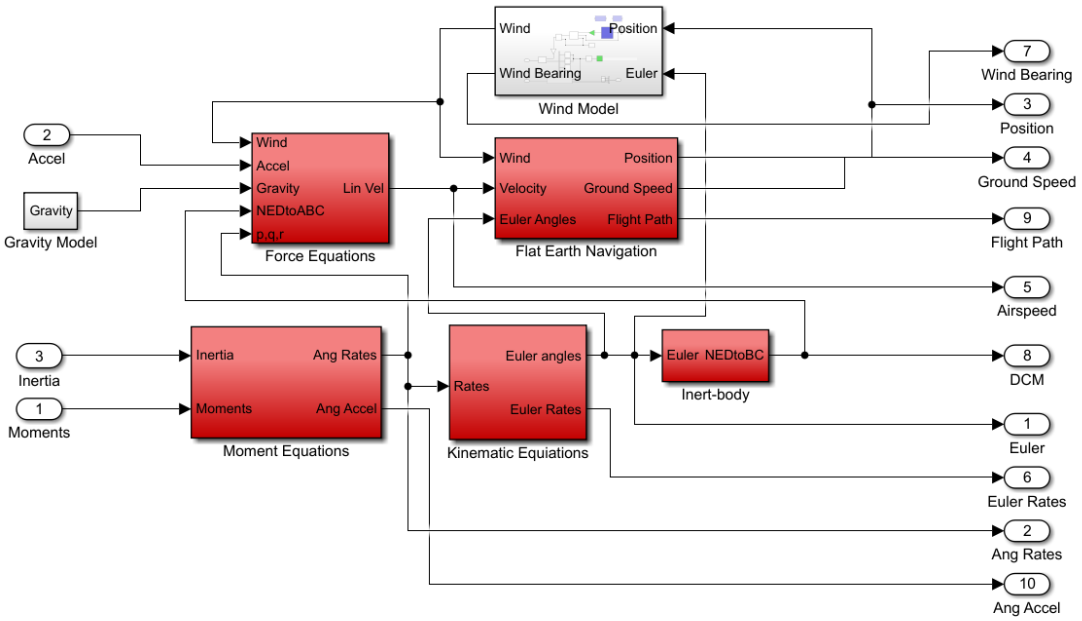

Figure 3.23 Inside the Equations of Motion and Numerical Integration block.

3.4.8 Force Equations

Equations 22 through 24 are used to calculate the UAV's airspeed. These equations combine the contributions of angular acceleration, gravitational acceleration, and linear acceleration and integrate to obtain the resulting airspeed. Additionally, to integrate the contribution of the wind shear to the UAV's energy, the derivative of the wind velocity is converted to the body reference frame and included in the integral. The integration of the windshear term in the force equation is based upon the work by Frost and Bowles [14].

$$U = \int RV - QW - g\sin\theta + \frac{F_{B_x}}{m} - \dot{W}_x B_B \, dt \tag{22}$$

$$V = \int -RU + PW + g\sin\phi\cos\theta + \frac{F_{B_y}}{m} - \dot{W}_y B_B \, dt \tag{23}$$

$$W = \int QU - PV + g\cos\phi\cos\theta + \frac{F_{B_z}}{m} + \dot{W}_z B_B \, dt \tag{24}$$

3.4.9 Flat Earth Navigation

The Flat Earth Navigation block converts the airspeed to the earth reference frame and calculates the position, groundspeed, and flight path of the UAV relative to the earth. Equations 25 through 27 are used to calculate the groundspeed. Similar to the force equations, the wind shear term, W, is included in each equation as suggested by Frost and Bowles [14].

$$p_{North} = \int U\cos\theta\cos\psi + V(-\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi) + W(\sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi) + W_x\,dt \tag{25}$$

$$p_{East} = \int U\cos\theta\sin\psi + V(\cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi) + W(-\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi) + W_y\,dt \tag{26}$$

$$\dot{h} = \int U\sin\theta - V\sin\phi\cos\theta - W\cos\phi\cos\theta + W_z\,dt \tag{27}$$

The UAV flight path is a quantity unique from its attitude. It is defined as the direction that the UAV is actually moving in, rather than where the nose is pointed. This can be derived from the current position and the position during the previous time step. Equations 28 through 30 are to calculate the difference of position between time levels, and equations 31 and 32 are used to calculate the flight path.

$$\Delta N = N^t - N^{t-\Delta t} \tag{28}$$

$$\Delta E = E^t - E^{t-\Delta t} \tag{29}$$

$$\Delta h = h^t - h^{t-\Delta t} \tag{30}$$

$$FP_{long} = \text{atan2}\left(\Delta h, \sqrt{\Delta N^2 + \Delta E^2}\right) \tag{31}$$

$$FP_{lat} = \text{atan2}(\Delta E, \Delta N) \tag{32}$$

3.4.10 Moment Equations

The moment equations are used to calculate the angular rates and accelerations. Equations 33 through 35 are integrated to obtain the angular rates. Additionally, these equations require the addition of a set of 9 coefficients. These can be obtained using equations 36 through 45.

$$P = \int (c_1 R + c_2 P)Q + c_3 l + c_4 N \, dt \tag{33}$$

$$Q = \int c_5 PR - c_6 (P^2 - R^2) + c_7 M \, dt \tag{34}$$

$$R = \int (c_8 P - c_2 R)Q + c_4 l + c_9 N \, dt \tag{35}$$

$$\Gamma c_1 = (J_y - J_z)J_z - J_{xz}^2 \tag{36}$$

$$\Gamma c_2 = (J_x - J_y + J_z)J_{xz} \tag{37}$$

$$\Gamma c_3 = J_z \tag{38}$$

$$\Gamma c_4 = J_{xz} \tag{39}$$

$$c_5 = \frac{J_z - J_x}{J_y} \tag{40}$$

$$c_6 = \frac{J_{xz}}{J_y} \tag{41}$$

$$c_7 = \frac{1}{J_y} \tag{42}$$

$$\Gamma c_8 = J_x(J_x - J_y) + J_{xz}^2 \tag{43}$$

$$\Gamma c_9 = J_x \tag{44}$$

$$\Gamma = J_x J_z - J_{xz}^2 \tag{45}$$

3.4.11 Kinematic Equations

Equations 46 through 48 are used to calculate the attitude of the aircraft from the angular rates p, q, and r. The attitude is also referred to as the Euler angles where yaw is defined in the North, East, and Down frame of reference, while pitch and roll are defined in an intermediate reference frame.

$$\phi = \int P + \tan\theta \, (Q \sin\phi + R \cos\phi) \, dt \tag{46}$$

$$\theta = \int Q \cos\phi - R \sin\phi \, dt \tag{47}$$

$$\psi = \int \frac{Q \sin\phi + R \cos\phi}{\cos\theta} dt \tag{48}$$

3.4.12 Inert Body

The Inert Body block is used simply to calculate the transformation matrix from the North, East, and Down reference frame to Body reference frame. Equation 49 is also known as the Direction Cosine Matrix or DCM.

$$NEDtoABC = B_B = \begin{bmatrix} \mathrm{c}\,\theta\,\mathrm{c}\,\psi & \mathrm{c}\,\theta\,\mathrm{s}\,\psi & -\mathrm{s}\,\theta \\ -\mathrm{c}\,\phi\,\mathrm{s}\,\psi + \mathrm{s}\,\phi\,\mathrm{s}\,\theta\,\mathrm{c}\,\psi & \mathrm{c}\,\phi\,\mathrm{c}\,\psi + \mathrm{s}\,\phi\,\mathrm{s}\,\theta\,\mathrm{s}\,\psi & \mathrm{s}\,\phi\,\mathrm{c}\,\theta \\ \mathrm{s}\,\phi\,\mathrm{s}\,\psi + \mathrm{c}\,\phi\,\mathrm{s}\,\theta\,\mathrm{c}\,\psi & -\mathrm{s}\,\phi\,\mathrm{c}\,\psi + \mathrm{c}\,\phi\,\mathrm{s}\,\theta\,\mathrm{s}\,\psi & \mathrm{c}\,\phi\,\mathrm{c}\,\theta \end{bmatrix} \tag{49}$$

3.4.13 Wind Model

The wind shear model block allows for the creation of a wind shear of any shape. The direction and maximum magnitude can also be changed, allowing for dynamically varying wind shear conditions during simulations. This can be seen in Equations 50 through 52 which were developed by Military Specification MIL-F-8785C [15]. For this case, a logarithmic wind shear function is given as an example. However, the logarithmic terms can be substituted with any type of gradient normalized from 0 to 1 to form a wind shear.

$$W_x = -W_{20}\frac{\ln\frac{h}{0.15}}{\ln\frac{20}{0.15}}\cos\psi_{wind} \tag{50}$$

$$W_y = -W_{20}\frac{\ln\frac{h}{0.15}}{\ln\frac{20}{0.15}}\sin\psi_{wind} \tag{51}$$

$$W_z = 0 \tag{52}$$

3.4.14 Flight Parameters

The Flight Parameter block calculates the angle of attack, sideslip, magnitude of the airspeed, and dynamic pressure. These are obtained using equations 53 through 56.

$$\alpha = \tan^{-1}\frac{W}{U} \tag{53}$$

$$\beta = \sin^{-1}\frac{V}{V_{TAS}} \tag{54}$$

$$V_{TAS} = \sqrt{U^2 + V^2 + W^2} \tag{55}$$

$$\bar{q} = \frac{1}{2}\rho {V_{TAS}}^2 \tag{56}$$

3.4.15 Engines

The thrust of the engine is a function of the airspeed and propellor angular rate. Propellor angular rate is calculated using equation 57 and thrust is calculated using equation 58. The thrust equation is derived based upon certain geometrical properties of the propellor such as diameter and pitch. The source for these equations is regrettably lost. These must be replaced in the future for better documentation. However, the maximum and minimum airspeeds of the Fox simulation are within a reasonable range of the true maximum and minimum airspeeds of the Fox.

$$\omega = 10120\delta_{thr} \tag{57}$$

$$T = 0.2248 * 1.225\pi * \frac{(0.0254 * 15)^2}{4}\left(\frac{7.5 * 0.0254\omega}{60}\right)^2 - \left(\frac{7.5 * 0.0254\omega}{60}\right)(0.3048V)\left(\frac{15}{3.29546 * 7.5}\right)^{3/2} \quad (58)$$

3.5 Reinforcement Learning

Reinforcement learning (RL) is a machine-learning method that automates the process of decision-making through goal-oriented modeling. An agent can learn from its environment without the need for large, complex data sets for training. Instead, the agent seeks to perform actions which result in rewards of variable weight until it reaches its final goal. In other words, a reinforcement learning agent searches for an optimal policy that maximizes the expected cumulative long-term reward.

The key elements of the reinforcement learning method are the agent and environment. The environment is the world around the agent. It can include a physical environment such as the earth, and it can also contain a vehicle with states such as a UAV. The agent is the "brain" of the system. It receives observations and rewards from the environment and chooses actions to perform through the creation of a policy.

The policy maps states to actions and can be developed through many different methods. Policies can be either deterministic, stochastic, or parametric. A deterministic policy produces a consistent result given a set of inputs. If "A" is true, then do "B". An analogy to this could be a lookup table such as the aerodynamic lookup tables for the Fox. A stochastic policy has an element of randomness. It can handle less predictable inputs that a deterministic policy cannot. Finally, there are parametric policies. These are policies determined using complex deep neural networks.

In addition to the policy, there is also a learning algorithm contained within the agent's system. The learning algorithm is what generates the policy. It directly observes the environment and receives rewards for the actions determined by the policy. There are many types of learning

algorithms with different strengths and weaknesses, but one must be chosen based upon the requirements of the application. Figure 3.24 shows a diagram of the reinforcement learning loop. Again, the environment in the case of dynamic soaring is the UAV.

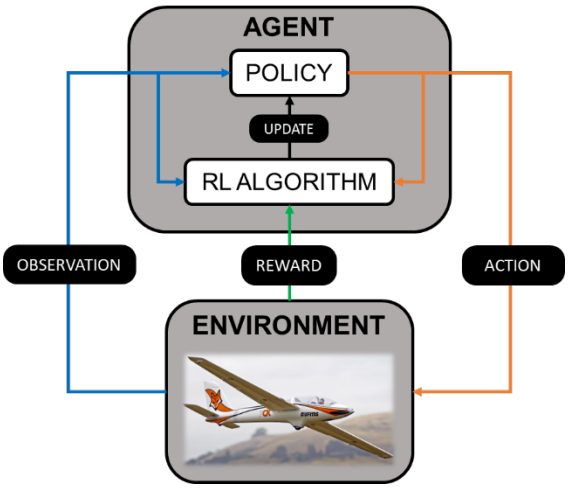

Figure 3.24 Diagram of the reinforcement learning process.

Other aspects of reinforcement learning are the reward, actions, and observations. The actions and observations can be either discrete or continuous. A discrete action/observation example is a game of checkers. There are a finite number of spaces on the board and a finite number of actions that can be taken at a given state. A continuous action is an action in which there are theoretically unlimited states. An example of this is the elevator on an aircraft. To maintain level flight, there is no easily definable position to hold the elevator to. Both large and fine adjustments are needed to adapt to the state of the aircraft due to atmospheric perturbations. The observations follow the same concept. Ideally, observations need to be states related to the actions in some logical way. Thus, a reinforcement learning policy can be derived. If illogical observations are chosen, there is no way for an optimal policy to be learned. The final component is the reward function. Rewards are used to train an agent to perform specific tasks. Reward functions must be designed such that the agent can learn optimal policies through continuous training.

In the case of dynamic soaring, both the actions and observations are continuous. This significantly narrows the field of candidate learning algorithms. In MATLAB, there are several built-in learning algorithms in the Reinforcement Learning toolbox. The most basic algorithm in MATLAB that uses both continuous actions and observation states is Deep Deterministic Policy Gradient.

3.5.1 Neural Network

A neural network is a universal function approximator comprised of a group of interconnected nodes called "neurons". Each neuron has a value which is affected by the weights and biases of nodes in lower levels. Neural networks are analogous to a complex lookup table that can model any input vs. output relationship given the right combination of nodes. All networks are comprised of input nodes, output nodes, and hidden layers between. A diagram of a neural network can be seen in Figure 3.25.

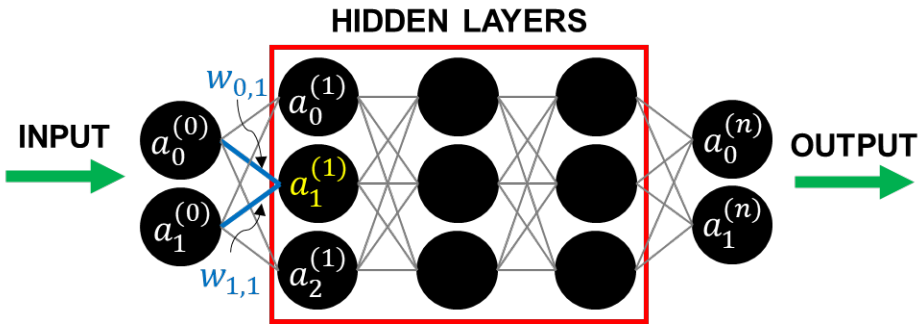

Figure 3.25 Diagram of a neural networks with fully connected hidden layers.

All nodes are fully connected to the nodes in the previous layer. Each node has value and bias, and each connection has a weight. These values are combined to form the weight of the new node as in equation 59. A nonlinear activation function can be applied to equation 59 to allow for modeling of any function using a neural network. This is shown in equation 60.

$$a_1^{(1)} = w_{0,1}a_0^{(0)} + w_{1,1}a_1^{(0)} + \mathrm{b}_1^{(1)} \tag{59}$$

$$a^{(n)} = \text{act}\left[w_{n-1}a^{(n-1)} + \text{b}^{(n)}\right] \tag{60}$$

3.5.2 Deep Deterministic Policy Gradient

The theory of Policy Gradient was first developed by Sutton et al. in 1999 [16]. Traditionally, reinforcement learning algorithms chose "greedy" policies to select actions with the highest estimated value. However, this would not always result in the most optimal outcome. Greedy policies have a tendency to get stuck on local maximum rewards rather than the global maximum. Sutton proposed a method known as Policy Gradient where the policy is explicitly represented by a function approximator. With this method, the values of the policy are updated in the direction of the gradient of the expected reward.

As a continuation of this method, Deterministic Policy Gradient was developed by Silver et al. in 2014 [17]. The deterministic version of the policy gradient method requires less computational effort compared to stochastic policy gradients. Also, the policy gradient integrates only over the state space rather than both the state and action spaces.

Finally, Deep Deterministic Policy Gradient was developed in 2016 by Lillicrap et al. [18] which applied a deep neural network as the function approximator. DDPG is a model-free, online, off-policy, actor-critic learning algorithm. Model-free methods do not explicitly reference the model through the use of generated predictions. For online training, a policy is generated at each step. Off-policy means that the policy used to choose an action is separate from the policy that is evaluated and updated. Finally, actor-critic is a reference to the type of neural network used. There are two networks: the actor network chooses actions while the critic evaluates the value of the action and observation states. Figure 3.26 shows a diagram of the actor-critic network.

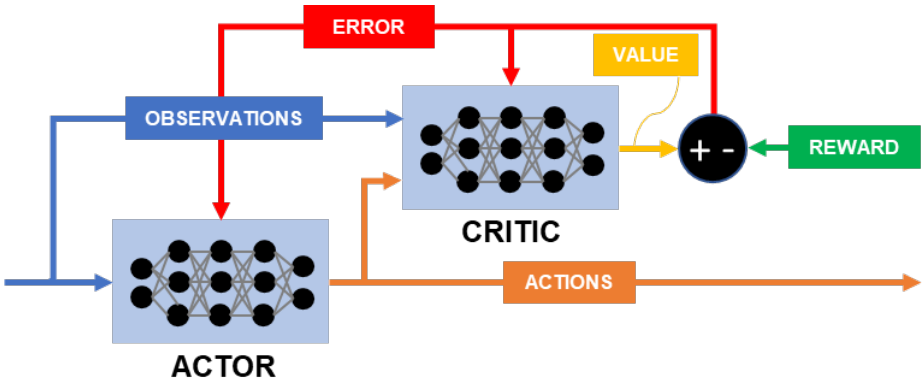

Figure 3.26 Diagram of the actor-critic network for DDPG.

DDPG uses deep neural networks as the function approximator, which allows for continuous action and observation states. Additionally, DDPG uses an experience buffer to minimize correlations between successive samples and target networks to improve the stability of learning. These methods were inspired by Deep-Q Learning, which also uses deep neural networks.

3.5.3 Deep Deterministic Policy Gradient Algorithm

The Deep Deterministic Policy Gradient algorithm is described below for equations 61 through 66 as developed by Lillicrap et al. [18].

During the initialization of a new agent, the critic network $Q(s,a|\theta^Q)$ and actor network $\mu(s|\theta^\mu)$ are assigned random weights θ^Q and θ^μ. In addition, the target networks Q' and μ' are assigned the same weights such that $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$. Finally, the replay buffer R is initialized.

With the necessary values initialized for a new agent, the episode loop can begin. At the beginning of each episode of training a random process N is initialized based upon an arbitrary noise model to allow for action exploration. The initial observation state, s_1, is also obtained at the start of each episode.

Within each training episode is a loop in time with a predefined length and time-step. Each equation is evaluated at every time step within this loop. First, an action a_t is selected according to the current policy and exploration noise N in equation 61.

$$a_t = \mu(s_t|\theta^\mu) + N_t \tag{61}$$

Once the action is taken, the reward r_t and new state s_{t+1} are observed. The experience (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer R. Next, a random minibatch of n experience (s_i, a_i, r_i, s_{i+1}) is sampled from the replay buffer R. Using equation 62, the value function target is set to

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1|\theta^{\mu'}})|\theta^{Q'}) \tag{62}$$

where γ is the discount factor of the expected future reward. Now the critic can be updated by minimizing the loss across all sampled experience using equation 63.

$$L = \frac{1}{n}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2 \tag{63}$$

Equation 64 is used to update the actor policy using the sampled policy gradient.

$$\nabla_{\theta^\mu} J = \frac{1}{n}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i} \tag{64}$$

Finally, the target networks are updated in equations 65 and 66.

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{65}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{66}$$

The loop in time is repeated for each time step until the completion of an episode where the episode loop also repeats until training is complete.

3.5.4 Dynamic Soaring using Reinforcement Learning

There are two proposed approaches to achieving dynamic soaring using reinforcement learning. The first is a single-agent method where the agent controls the UAV and also optimizes the path to perform dynamic soaring. This method is conceptually simple, however, training an agent with no knowledge of its goals to perform a dynamic soaring maneuver is not so simple. A second approach is a double-agent method. In this method, the first agent is trained to follow a predefined path of waypoints using as little engine thrust as possible. The second agent then optimizes the waypoints to create closed-loop dynamic soaring paths. This method is significantly more difficult to set up in the simulation but should also be far more robust since the exact path the UAV should follow is known.

Figure 3.27 shows the internal components of the reinforcement learning block in Simulink. The main component is the "RL_Agent" block in the center of the figure. This block interacts with a MATLAB training script and allows for a Simulink-based model to be trained using reinforcement learning. The observation block holds all the observation states. The reward block contains all the reward functions. "IsDone" is a feature that allows for an episode to be terminated if a specified condition is met. In general, for the Fox simulation, the episode was terminated early if the altitude was less than or equal to zero feet. The error block reads the desired waypoints and determines the positional, angular, and flight path errors to be used for the rewards and observations. Finally, the "RL_Agent" outputs the actions into the environment. In all cases, the actions were filtered through a set of PID controllers which, in turn, convert the agent outputs into the deflection angles of the control surfaces.

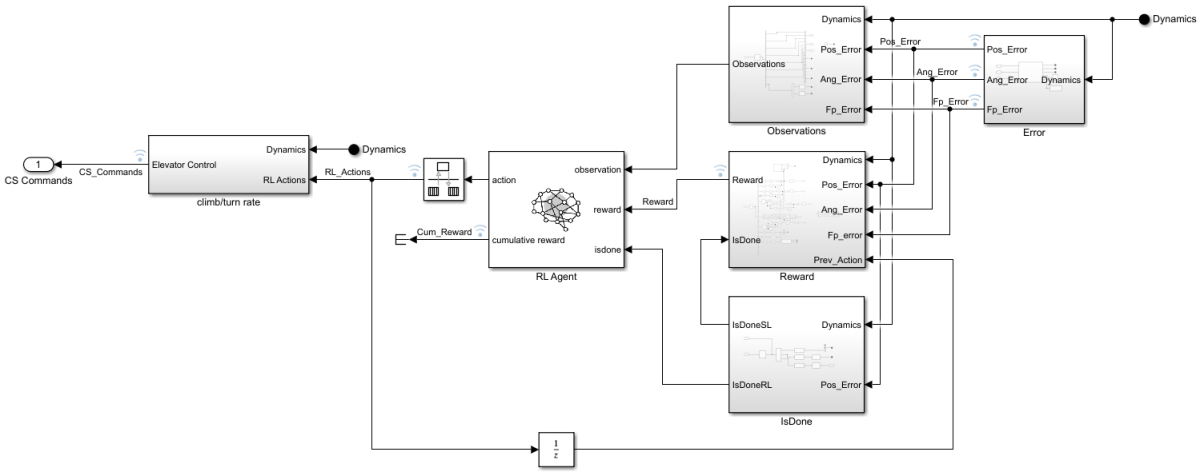

Figure 3.27 Overview of the RL Trainer block.

3.5.5 Waypoint Algorithm

For the double-agent method to work, a waypoint algorithm had to be developed that computes the error between the UAV's current state and the desired state. The algorithm was designed to work specifically for closed-loop waypoint paths. A set of seed-points are input into a spline-based interpolation algorithm to obtain a detailed set of thousands of closed-loop waypoints. These waypoints alongside the UAV's position, attitude, and flight path are input into the waypoint algorithm. From these inputs the error between the UAV and the desired position, angle, and flight path are calculated at every time step. This block can be seen in Figure 3.28.

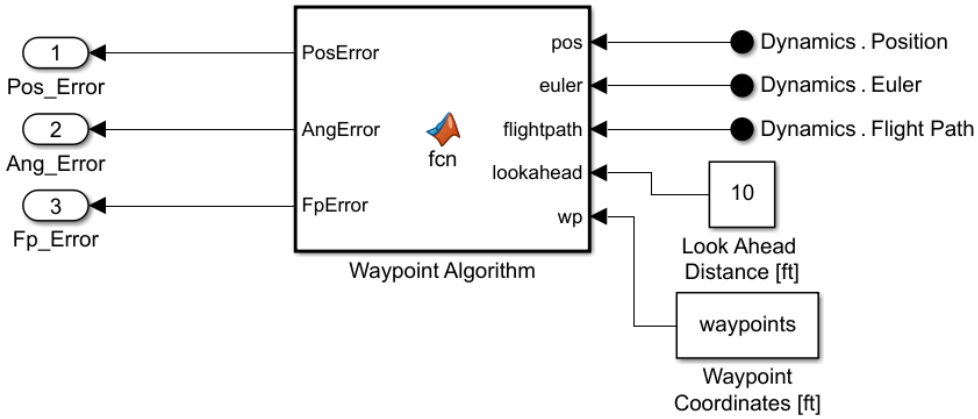

Figure 3.28 Overview of the waypoint algorithm block.

The waypoint algorithm uses a reference point separate from the UAV known as a lookahead point. This is a point an arbitrary distance in front of the UAV on the longitudinal axis. To work with a closed-loop waypoint path, the algorithm needs to determine the closet point to the lookahead point to calculate the positional and angular errors. The angular error is the bearing and pitch to the nearest waypoint from the look-ahead point. Additionally, the algorithm also determines the error between the current flight path of the UAV and the desired flightpath at the nearest waypoint. The flight path is used to enforce an ideal direction either clockwise or counterclockwise along the closed-loop waypoints. Figure 3.29 is a diagram of the waypoint algorithm.

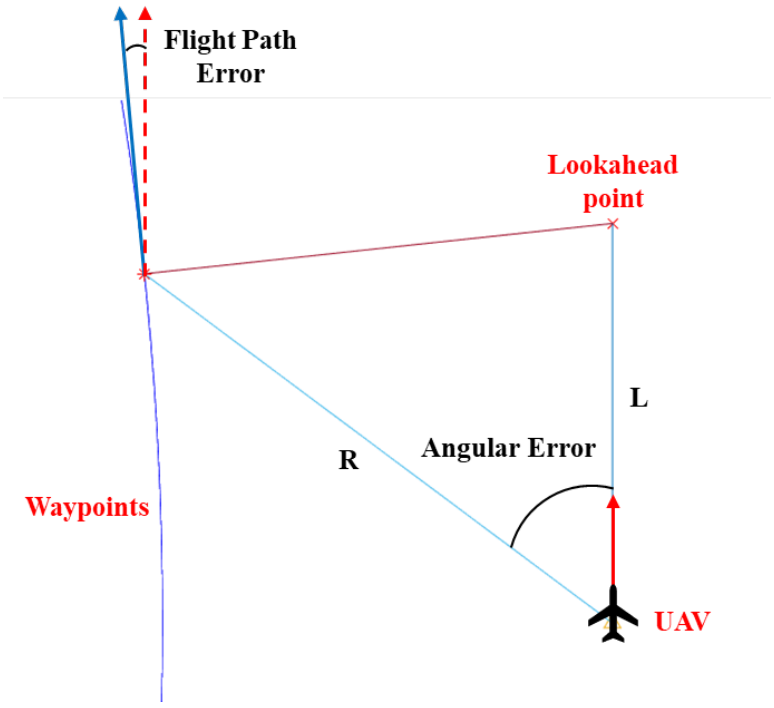

Figure 3.29 Diagram of the waypoint algorithm for the Path-Following reinforcement learning agent.

To obtain the required errors for the Path-Following agent, the lookahead point must be calculated as a function of the UAV's position and desired distance in front of the UAV. Equations 67 through 69 calculate the position of the look ahead point

$$N_{LAP} = N_{UAV} + L\cos\theta\cos\psi \tag{67}$$

$$E_{LAP} = E_{UAV} + L\cos\theta\sin\psi \tag{68}$$

$$h_{LAP} = h_{UAV} + L\sin\theta \tag{69}$$

where L is the desired lookahead distance, N is the northern coordinate, E is the eastern coordinate, and h is the altitude coordinate of the UAV and lookahead points. The lookahead point is used primarily to ensure that the nearest waypoint to the UAV is never behind the UAV relative to the direction it is traveling.

To obtain the error between the lookahead point and all waypoints equations 70 through 73 are used. Notably, equation 73 is the magnitude of the error between the lookahead point and waypoints. This will be used to determine the closest waypoint at any given time.

$$(\varepsilon_{LAP})_N = N_{WP} - N_{LAP} \tag{70}$$

$$(\varepsilon_{LAP})_E = E_{WP} - E_{LAP} \tag{71}$$

$$(\varepsilon_{LAP})_h = h_{WP} - h_{LAP} \tag{72}$$

$$\varepsilon_{LAP} = \sqrt{((\varepsilon_{LAP})_N)^2 + ((\varepsilon_{LAP})_E)^2 + ((\varepsilon_{LAP})_h)^2} \tag{73}$$

Similarly, the error between the UAV position and all waypoints can be calculated using equations 74 through 77.

$$(\varepsilon_{UAV})_N = N_{WP} - N_{UAV} \tag{74}$$

$$(\varepsilon_{UAV})_E = E_{WP} - E_{UAV} \tag{75}$$

$$(\varepsilon_{UAV})_h = h_{WP} - h_{UAV} \tag{76}$$

$$\varepsilon_{UAV} = [(\varepsilon_{UAV})_N, (\varepsilon_{UAV})_E, (\varepsilon_{UAV})_h] \tag{77}$$

Equation 78 is used to determine the location of the closest waypoint relative to the lookahead point.

$$\left[N_{target}\,, E_{target}\,, h_{target}\right] = \min(\varepsilon_{LAP}) \tag{78}$$

With the target waypoint located, the positional error between the UAV and target waypoint can be calculated using equations 79 through 81. This is the "Pos_Error" output in Figure 3.28.

$$\left(\varepsilon_{pos}\right)_N = N_{target} - N_{UAV} \tag{79}$$

$$\left(\varepsilon_{pos}\right)_E = E_{target} - E_{UAV} \tag{80}$$

$$\left(\varepsilon_{pos}\right)_h = h_{target} - h_{UAV} \tag{81}$$

Even though the lookahead point is used to locate the target waypoint, the position error is still determined by the true position of the UAV. A drawback to this method is that the minimum position error possible is equal to the lookahead distance. This does not present a problem for the reinforcement learning algorithm since it simply seeks to minimize the error. It does not need to be zero to work optimally.

Compared to the position error, the angular error requires a more complex analysis. The angular error can be broken into two components: lateral and longitudinal. Lateral angular error is simply the relative heading from the UAV to the target waypoint. Longitudinal angular error is the relative pitch between the UAV and target waypoint.

Beginning with the lateral angular error in equation 82 it is simply the difference of the "atan2" between the vector of the lookahead point and the vector created between the target waypoint and UAV.

$$\varepsilon_\psi = \text{atan2}\left(L_y, L_x\right) - \text{atan2}\left(\left(\varepsilon_{pos}\right)_E, \left(\varepsilon_{pos}\right)_N\right) \tag{82}$$

Ideally, this should output an angle between -180 and +180 degrees, but it exceeds this limit in certain cases. Thus, if the error is less than $-\pi$, 2π is added to the error, Similarly, if the error is more than π, 2π is removed from the error. This fixes the angular limit issue.

The process for longitudinal error is similar to the lateral error. However, the vectors inside the "atan2" functions are different. First, a static reference vector, k, is defined as $k = [0,0,1]$. This will be used to develop a vertical vector as a function of the UAV's current attitude in equation 83.

$$\begin{aligned}[N_{vec}, E_{vec}, h_{vec}] \\ &= \varepsilon_{UAV} \cos \varepsilon_{\psi} + (k \times \varepsilon_{UAV}) \sin \varepsilon_{\psi} + k(k \cdot \varepsilon_{UAV})\big(1 - \cos \varepsilon_{\psi}\big) \\ &+ [N_{UAV}, E_{UAV}, h_{UAV}]\end{aligned} \tag{83}$$

Finally, the longitudinal angular error can be calculated as a difference of "atan2" functions in equation 84. Both equations 82 and 84 are the "Ang_Error" output in Figure 3.28.

$$\begin{aligned}\varepsilon_{\theta} = \operatorname{atan2}&\left(L_z, \sqrt{(L_x)^2 + \big(L_y\big)^2}\right) \\ &- \operatorname{atan2}\left(h_{vec} - h_{UAV}, \sqrt{(N_{vec} - N_{UAV})^2 + (E_{vec} - E_{UAV})^2}\right)\end{aligned} \tag{84}$$

The last value to calculate is the flight path error. This is required in addition to the angular error because the angular error alone is not enough information for the Path-Following agent to function. In dynamic soaring the UAV will need to follow the path as closely as possible even at high angles of attack. This requires the ideal flight path to be known to the agent. The process for this is simple. First, the flight path of the waypoints is found using equations 85 and 86.

$$(FP_{WP})_{long} = \operatorname{atan2}\left(\Delta h_{WP}, \sqrt{(\Delta N_{WP})^2 + (\Delta E_{WP})^2}\right) \tag{85}$$

$$(FP_{WP})_{lat} = \operatorname{atan2}(\Delta E_{WP}, \Delta N_{WP}) \tag{86}$$

Then the flight path error can be calculated as the difference between the UAV's flight path and ideal flight path from the waypoints using equations 87 and 88. These are the "Fp_Error" output in Figure 3.28.

$$(\varepsilon_{FP})_{long} = (FP_{WP})_{long} - (FP_{UAV})_{long} \tag{87}$$

$$(\varepsilon_{FP})_{lat} = (FP_{WP})_{lat} - (FP_{UAV})_{lat} \tag{88}$$

3.6 Configuration of the Reinforcement Learning Simulations

Below is a summary of the actions, observations, and rewards used for the most successful reinforcement learning simulations. The adaptations between each agent are a culmination of several weeks of development using different settings. Only the most important agents were saved to show the evolution of the methods over time.

Unfortunately, due to technical issues with MATLAB and Simulink, only a few agents could be saved for analysis. For example, the first attempts at training the path following agent utilized direct control of the Fox's control surfaces as the action states. It was determined that this method of control is not optimal for exploration as most combinations of elevator and aileron deflections will result in stalling. More robust methods of control were chosen to allow for the UAV to explore the environment to gain experience. With that being said, the first agent that was managed to be saved will be referred to as "generation 1" even though it is not the first agent that was developed.

3.6.1 Path-Following Agent Generation 1

The first generation of path following agent with any amount of success combined the reinforcement learning actions with a set of PID feedback controllers. This allows for the agent to directly control certain states of the UAV given that the PID controllers are turned properly. The PID controller values can be found in Table 3.10. In this case, the agent controls the UAV by specifying the pitch and roll angles. This allows for more exploration compared to direct control of the control surfaces. The desired closed-loop path can be seen in Figure 3.30.

Table 3.10 Generation 1 PID controller gain values used for control of the UAV.

Controller	Output	P-Gain	I-Gain	D-Gain	N-Filter	Saturation
Pitch	Elevator	100	3	30	100	none
Roll	Aileron	1	0	0.5	100	none

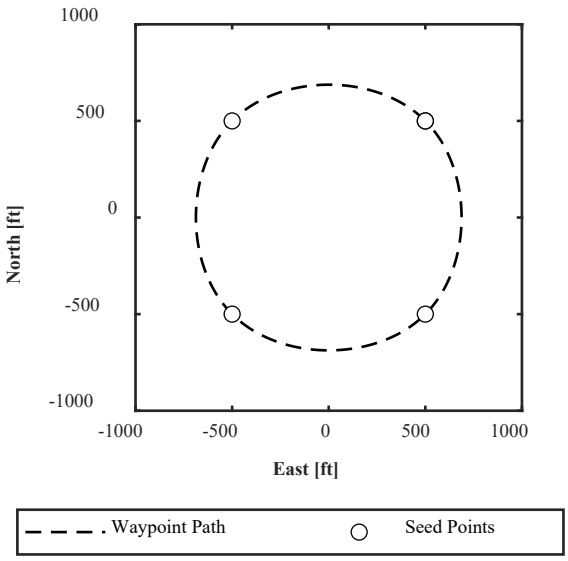

Figure 3.30 The predefined close-loop waypoint path used to train the path following agent.

The waypoints were generated using splines from a set of four seed points. The desired altitude is constant at 400 ft. The agent observed the positional error, the angular error, and the current UAV attitude. The action and observations are summarized in equations 89 and 90 respectively.

$$a_t = [\theta, \phi] \tag{89}$$

$$s_t = \left[\delta_N, \delta_E, \delta_h, \delta_\theta, \delta_\psi, \phi, \theta, \psi\right] \tag{90}$$

The reward function was developed based on the waypoint path error and episode time (equations 91 through 93). The reward is calculated at every time step and cumulated over the full episode. Particularly for the time reward in equation 93, the value is only added to the total episodic reward if a certain condition is met. In this case, if the "IsDone" condition is met, then the time reward included. The "IsDone" condition is triggered if the altitude falls below zero feet, or if the angular rate of the UAV is beyond 100 rad/s. The altitude "IsDone" condition is used to simulate the effect of the UAV crashing into the ground and the angular rate "IsDone" condition is used to prevent the angular rates from growing out of control. At high (beyond 100 rad/s) angular speeds, the simulation may crash and lose all progress.

$$r_1 = 10e^{-0.012\delta_{pos}} \tag{91}$$

$$r_2 = 10e^{-0.07\delta_{att}} - 0.5{\delta_{att}}^{0.5} \tag{92}$$

$$r_3 = [-1E3 - 9E3e^{-0.06t}]\{IsDone = 1\} \tag{93}$$

The final component to the reinforcement learning method is the "Reset" function. This allows certain parameters to be randomized at the beginning of each episode for more robust training. In all generations, the initial position and heading were randomized as in equations 94 through 97.

$$h_0 = [200{:}\,600] \tag{94}$$

$$N_0 = [-500{:}\,500] \tag{95}$$

$$E_0 = [-500{:}\,500] \tag{96}$$

$$\psi_0 = [0{:}\,2\pi] \tag{97}$$

3.6.2 Path-Following Agent Generation 2

For the second generation of the path-following agent several modifications were made to dramatically improve the performance. First, the structure of the deep neural networks was reduced significantly. Ideally, the neural networks should be as small as possible to reduce training time. However, if too small, the optimal policy cannot be obtained. Additionally, the minibatch of randomly sampled experience was increased from 100 to 500 to improve the stability of the actions taken. A larger minibatch decreases the likelihood that the sampled experience is poor, preventing the selection of low-value actions. Thirdly, the lateral flightpath was included in the observations and reward as a way to define the directionality of the waypoints. The new observation states and new reward based on the flight path can be seen in equations 98 and 99 respectively. All other parameters are the same as in the first generation.

$$s_t = \left[\delta_N, \delta_E, \delta_h, \delta_\theta, \delta_\psi, \delta_{FP}, \phi, \theta, \psi\right] \tag{98}$$

$$r_4 = \left[10e^{-0.07\delta_{FP}} - 05{\delta_{FP}}^{0.5}\right]\{\delta_{pos} \leq 50\} \tag{99}$$

3.6.3 Path-Following Agent Generation 3

Two major changes were applied to the third generation of the Path-Following agent. First, the Euler rates were added to the reward and observations as an attempt to damp out unwanted oscillations in the pitch mode. Also, the PID controllers were retuned, and saturations were added so that the output of the PID controllers are limited to the control surface deflection limits. The new PID values are listed in Table 3.11. The new observation states and reward functions are given in equations 100 and 101.

Table 3.11 Generation 3 PID controller gain values used for control of the UAV.

Controller	Output	P-Gain	I-Gain	D-Gain	N-Filter	Saturation
Pitch	Elevator	1	0.1	0.1	100	-40° to +20°
Roll	Aileron	2	0.01	0.5	200	-35° to +35°

$$s_t = \left[\delta_N, \delta_E, \delta_h, \delta_\theta, \delta_\psi, \delta_{FP}, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}\right] \tag{100}$$

$$r_5 = -0.1\left(\left|\dot{\phi}\right| + \left|\dot{\theta}\right| + \left|\dot{\psi}\right|\right) \tag{101}$$

3.6.4 Path-Following Agent Generation 4

The fourth generation makes significant changes to all areas of the reinforcement learning function. As mentioned previously, the changes were made sequentially over a series of generations where the agents were not saved. Thus, the fourth generation Path-Following agent is a result of the combination of improvements made from unsaved agent generations.

First, the agent in this generation controls the climb and turn rate of the UAV. This also requires a new set of PID controllers, the values for which can be found in Table 3.12. These PID controllers also no longer directly control the control surfaces. Instead, the climb rate PID controller feeds into a pitch PID controller which feeds into a pitch rate PID controller that finally outputs the deflection angle of the elevators. A similar loop was designed for the turn rate as well. Additionally, a rate saturation was applied to the control surfaces in addition to the angular

magnitude saturation. This ensures that the control surfaces cannot deflect instantaneously between angles. The rates were measured from the real Fox UAV during flight tests.

Table 3.12 Generation 4 PID controller gain values used for control of the UAV.

Controller	Output	P-Gain	I-Gain	D-Gain	N-Filter	Saturation
Climb Rate	Pitch	0	0.01	0	100	None
Pitch	Pitch Rate	3	0	0	100	None
Pitch Rate	Elevator	0.138	3.15	0.000541	57.1	-25° to +25°
Turn Rate	Roll	0	1	0	100	None
Roll	Roll Rate	5	0	0	100	None
Roll Rate	Aileron	0.0768	2.26	0	100	-25° to +25°

Another major modification is that, in this generation specifically, the aerodynamic model was improved. As discussed in the CFD section, the roll lookup tables for the static aerodynamic coefficients caused major issues with the flight dynamics of the Fox. This was fixed for generation 4 onwards.

Finally, the neural network was modified for further improvements in training performance. The size of the critic was reduced too much in previous generations to accurately predict the long-term reward. Thus, the size of the critic neural network was increased. Also, an action scaling activation function was added to the output of the actor network. Without this function, the range of action values is limited between negative one and positive one. With the scaling layer, the output can be multiplied by a constant to increase or decrease the range to more relevant values.

The new actions and observation states can be seen in equations 102 and 103 respectively. The groundspeed and climb rate were added to the observation in equation 103.

$$a_t = [\dot{h}, \dot{\psi}] \tag{102}$$

$$s_t = [\delta_N, \delta_E, \delta_h, \delta_\theta, \delta_\psi, \delta_{FP}, \dot{N}, \dot{E}, \dot{h}, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}] \tag{103}$$

The new reward functions are given in equations 104 through 109. Slight modifications were made to the error-based reward equations. Also, a reward was designed to minimize the total

magnitude of the actions. Reward six (equation 109) normalizes the actions taken at a point in time by the maximum possible action value and multiplies the output by a negative number. This encourages the UAV to take more conservative actions to maximize the reward.

$$r_1 = 10e^{-0.012\delta_{pos}} - 0.3{\delta_{pos}}^{0.5} \tag{104}$$

$$r_2 = 10e^{-0.07\delta_{att}} - 0.5{\delta_{att}}^{0.5} \tag{105}$$

$$r_3 = \left[-05{\delta_{FP}}^{0.5}\right]\{\delta_{pos} \leq 50\} \tag{106}$$

$$r_4 = [-1E3 - 9E3e^{-0.06t}]\{IsDone = 1\} \tag{107}$$

$$r_5 = -0.1\left(\left|\dot{\phi}\right| + \left|\dot{\theta}\right| + \left|\dot{\psi}\right|\right) \tag{108}$$

$$r_6 = -\left(\left|\frac{\dot{h}_{t-1}}{\dot{h}_{max}}\right| + \left|\frac{\dot{\psi}_{t-1}}{\psi_{max}}\right|\right) \tag{109}$$

3.6.5 Unguided Energy-Seeking Agent

A single-agent dynamic soaring maneuver was attempted by developing a new set of rewards and observations. The UAV was controlled using the same actions as in the fourth-generation path-following agent where climb rate and turn rate are specified by the agent. This allows for the most stable exploration of the environment since there is not a target path to follow in this case. However, the PID controllers were retuned, and saturations were added to the inner-loop controllers to improve the performance to allow for the best control performance. Additionally, for this case the throttle for the engines was set to zero. Thus, there is no thrust force from the engines to maintain the UAV's energy. The agent must gain energy using the wind shear.

In this case, the wind shear is a simple triangular shape that tapers from 0 wind speed at the ground to 40 ft/s windspeed at 50 ft altitude. From 50 ft and above, the windspeed is constant. This is visualized in Figure 3.31. The windshear is orientated such that it is blowing from south to north.

When the simulation initializes, the UAV will be in a tailwind, which is advantageous for immediately gaining energy from the wind shear.

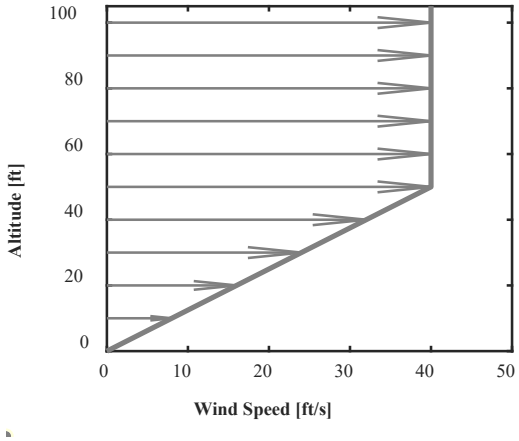

Figure 3.31 Triangular wind shear used to train the UES agent to perform dynamic soaring.

Because there is no ideal path for the UAV to follow, new observations and rewards had to be specified. The actions, which are the same as the fourth-generation path following algorithm are given in equation 110. For the observations in equation 111, the agent observes Euler angles, Euler rates, climb rate, airspeed, windspeed, and angle of attack. The windspeed is the difference between the groundspeed and airspeed. In a tailwind, the windspeed should be positive, and visa-versa for a headwind.

$$a_t = \left[\dot{h}, \dot{\psi}\right] \tag{110}$$

$$s_t = \left[\phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}, \dot{h}, V_A, V_w, \alpha\right] \tag{111}$$

The new rewards for the single-agent dynamic soaring are defined in equations 112 through 117. The first reward in equation 112 is designed such that the UAV must have a positive climb rate in a headwind and negative climb rate in a tailwind. This is meant to force the UAV into performing the windward climb and leeward descent phases of the dynamic soaring cycle. The second reward is designed to teach the agent to keep the angle of attack below the point of stall. The third reward is related to the rate of change of total energy. In a successful dynamic soaring

cycle, the total energy should be the same at the beginning and end of the maneuver. However, this is not easy to determine without a pre-defined path. Thus, this condition can be approximated by training the agent to minimize the total energy rate of change to prevent the loss of total energy. The fourth and fifth reward functions are for the same purpose as they were in the previous path-following agent.

$$r_1 = [10]\{V_w > 0 \,\&\, \dot{h} < 0 \,||\, V_w < 0 \,\&\, \dot{h} > 0\} \tag{112}$$

$$r_1 = [-10]\{V_w > 0 \,\&\, \dot{h} > 0 \,||\, V_w < 0 \,\&\, \dot{h} < 0\} \tag{113}$$

$$r_2 = -10|\alpha| - [100]\{\alpha \geq 12°\} \tag{114}$$

$$r_3 = -0.01\left|\dot{E}_t\right| \tag{115}$$

$$r_4 = -10\left|\frac{\dot{h}_{t-1}}{\dot{h}_{max}}\right| - 10\left|\frac{\dot{\psi}_{t-1}}{\psi_{max}}\right| \tag{116}$$

$$r_5 = t + [-1E3 - 9E3e^{-0.06t}]\{IsDone = 1\} \tag{117}$$

Finally, the new PID configuration can be seen in Table 3.13. Saturations are applied to the outputs of all PID controllers to limit the pitch and roll angles and rates to realistic ranges. Consequently, new gains were tuned for the PID controllers.

Table 3.13 UES agent PID controller gain values used for control of the UAV.

Controller	Output	P-Gain	I-Gain	D-Gain	N-Filter	Saturation
Climb Rate	Pitch	0	0.005	0	100	π/3 to -π/3
Pitch	Pitch Rate	3	0	0	100	-1 to 1
Pitch Rate	Elevator	0	1	0	100	-25° to +25°
Turn Rate	Roll	0	2	0	100	π/3 to -π/3
Roll	Roll Rate	5	0	0	100	-1 to 1
Roll Rate	Aileron	0	1	0	100	-25° to +25°

4 Results

Each reinforcement learning agent was simulated for 20 episodes. The episode with the highest total reward was used to demonstrate the performance of each agent.

4.1.1 Path-Following Agent Generation 1 Results

The history of the reward for each training episode can be seen in Figure 4.1. The vertical dashed black bars represent a 24-hour period in real-time. Evaluating the reward shows that the first-generation agent can maximize the reward over time. However, this agent also exhibits very inconsistent performance as the reward has a large trough around 3000 episodes and decreases drastically from episode 7000 onwards. Additionally, the critic does not predict an accurate long-term reward. Over time, both the actual and predicted rewards should converge. This may be a result of the extremely inconsistent performance of the agent. The reward oscillates constantly between -7500 and 7500.

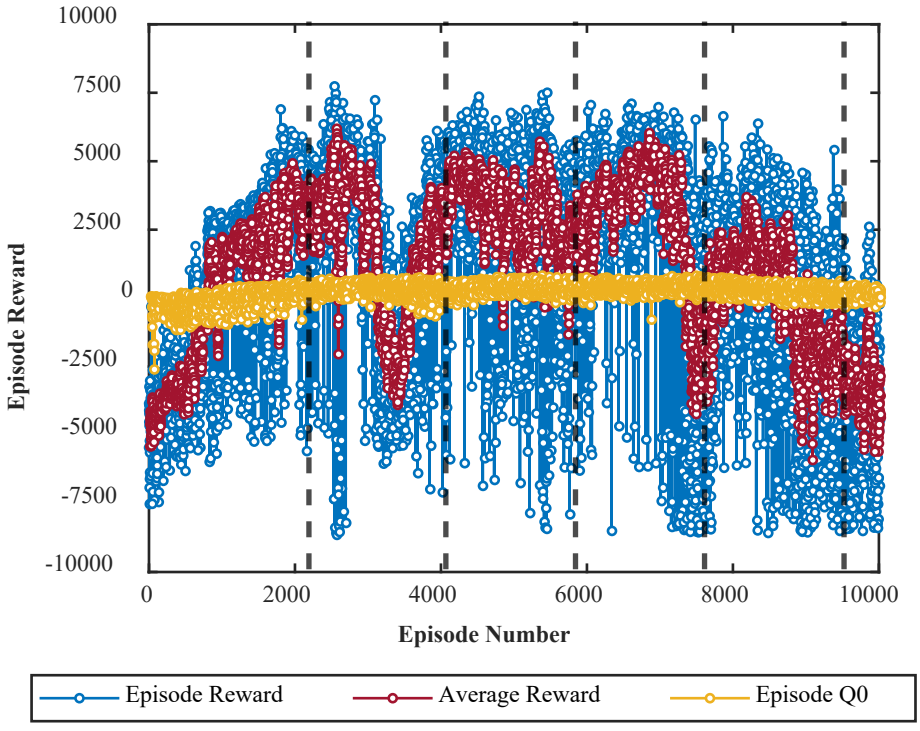

Figure 4.1 Episode reward and predicted reward for the first-generation agent.

Other aspects of the training statistics can be used to analyze the performance. Figure 4.2 displays the episode clock time and total episode steps. For the clock time, the time to complete each episode increases consistently as the agent gains experience. For the episode steps plot, if an episode does not complete the maximum number of steps, then it was terminated by the IsDone function. Therefore, it is clear that there are a large number of failed training episodes. This is especially true closest to episode 10000 where the highest density group of failed episodes is present.

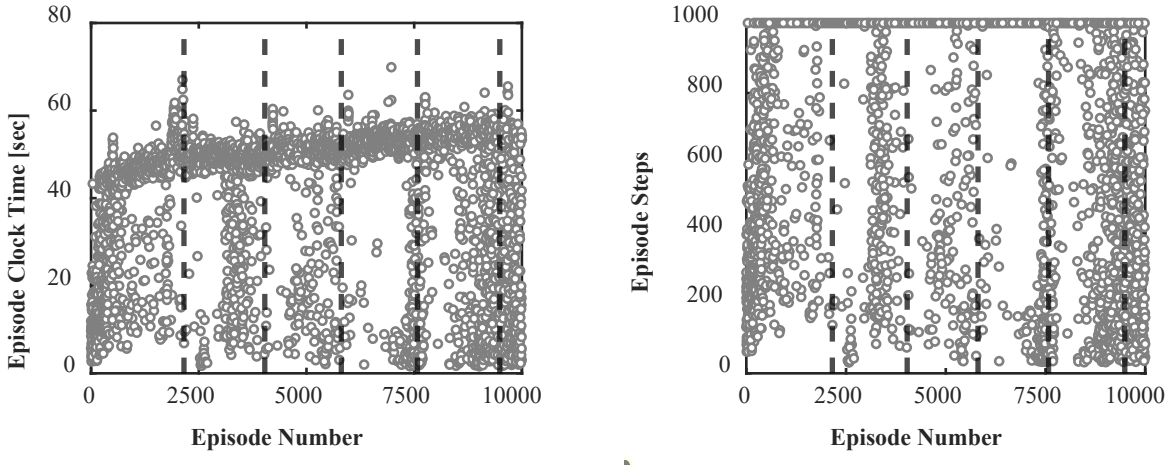

Figure 4.2 Episode clock time and steps for the first-generation agent.

Further analysis of the agent's performance can be carried out by reviewing the logged data of the UAV during the simulated episode. Figure 4.3 shows the pitch and roll actions chosen by the agent over time. It is obvious that these actions are poorly optimized as both the pitch and roll actions oscillate rapidly between the maximum and minimum values.

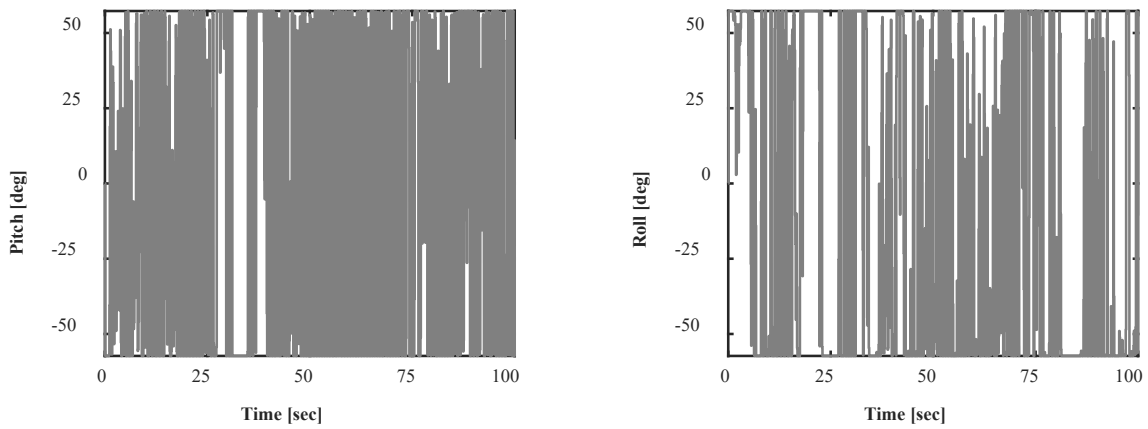

Figure 4.3 Pitch and Roll actions performed by the first-generation agent.

The ground track and altitude of the UAV can be seen in Figure 4.4 compared to the ideal path. The UAV is able to detect and loosely follow the ideal ground track. However, it does a poor job at maintaining the altitude, especially during a sharp turn.

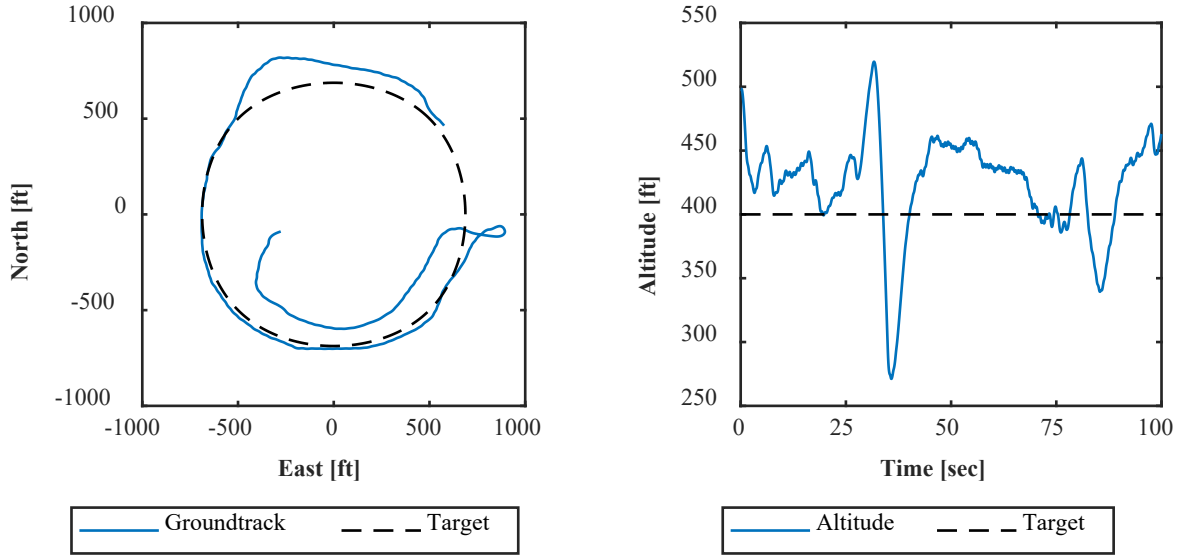

Figure 4.4 Ground track and altitude for the first-generation agent.

The true roll and pitch angles as well as their corresponding rates can be seen in Figure 4.5. They have large oscillations as a result of the poorly optimized actions.

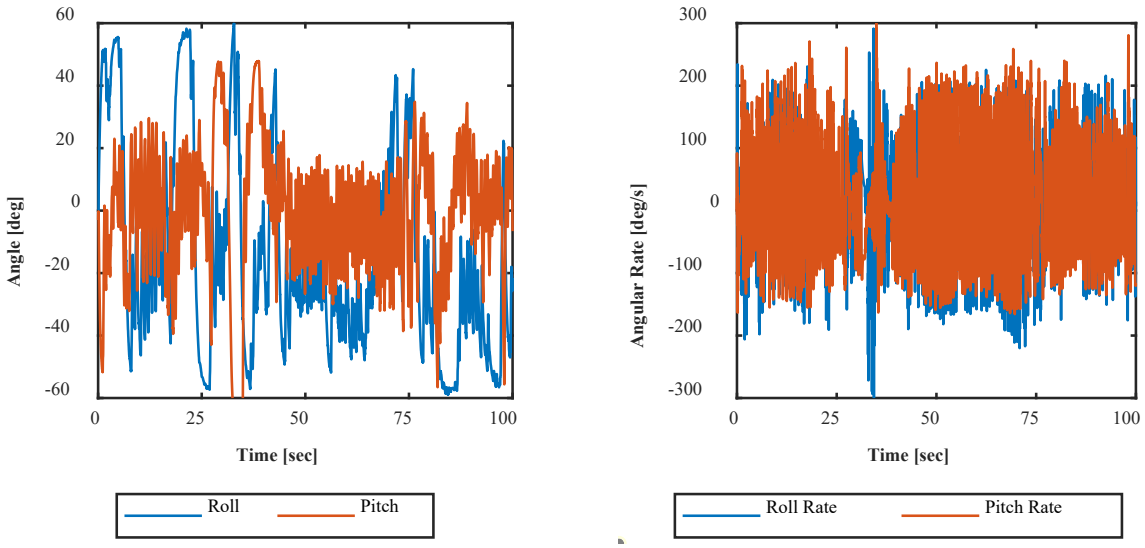

Figure 4.5 Euler angles and rates for the first-generation agent.

4.1.2 Path-Following Agent Generation 2 Results

The addition of the flight path error and increase in size for the minibatch samples significantly improved the overall performance of the second-generation agent. From Figure 4.6, while it takes significantly more episodes for a good policy to be learned that optimizes the reward, it is far more consistent because the variation of the reward between episodes is smaller compared to the first generation. This means that the performance is consistent until episode 2000 where the reward increases sharply, and the reward varies drastically between episodes. However, the critic predicted reward does not converge to the true reward. This indicates that the neural network is not properly optimized for this case.

Analysis of the episode clock time and steps in Figure 4.7 proves that the performance of the agent improved from the last generation. The only large grouping of failed episodes is near the start of training and only a few failed episodes exist outside of that group. The clock time plot again shows the episode time increases as experience is gained. However, there is a large time spike between episodes 3000 and 4000. This is a result of an attempt to train multiple agents using the same CPU. Parallel training of an agent is not practical using DDPG since it is only

advantageous if the computational cost to simulate the environment exceeds the cost to learn the policy. This is not the case since the simulation clock time is slowed significantly during training.

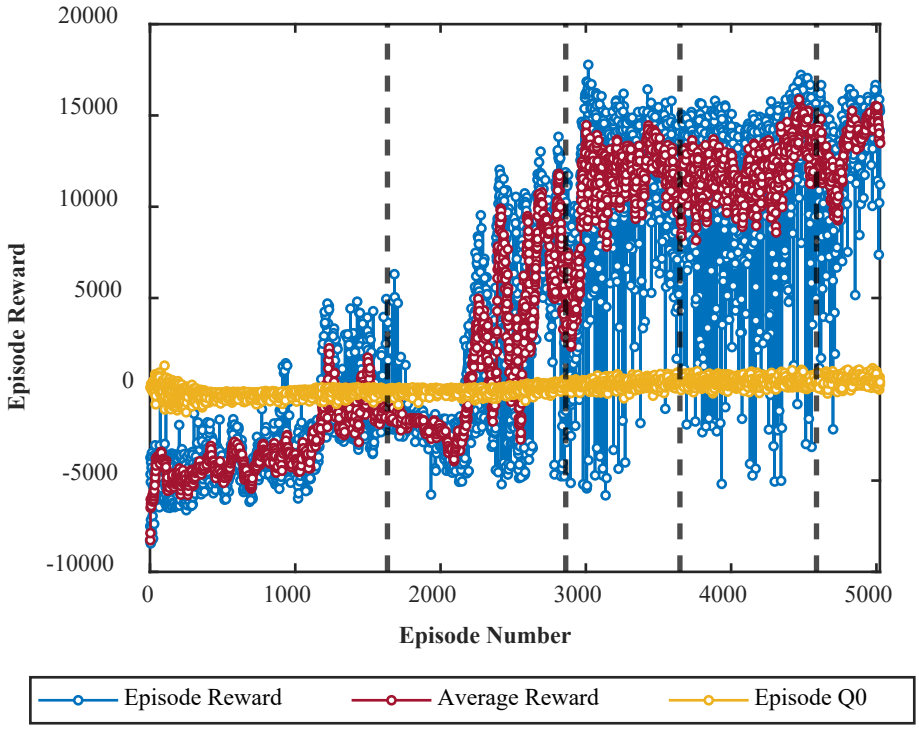

Figure 4.6 Episode rewards for the second-generation agent.

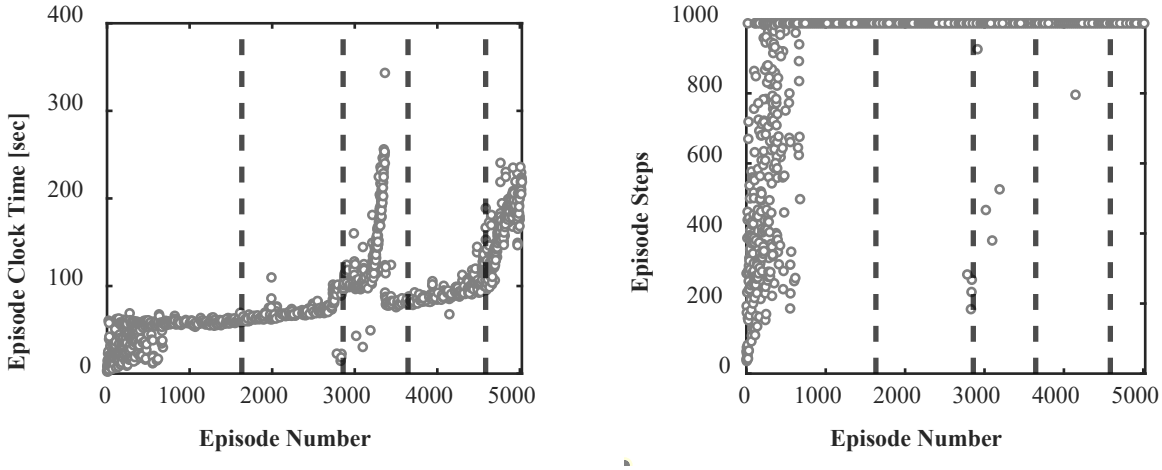

Figure 4.7 Episode clock time and steps for the second-generation agent.

Analysis of the pitch and roll actions in Figure 4.8 shows a slight improvement from the first-generation. But it is still too chaotic to be used to perform efficient dynamic soaring.

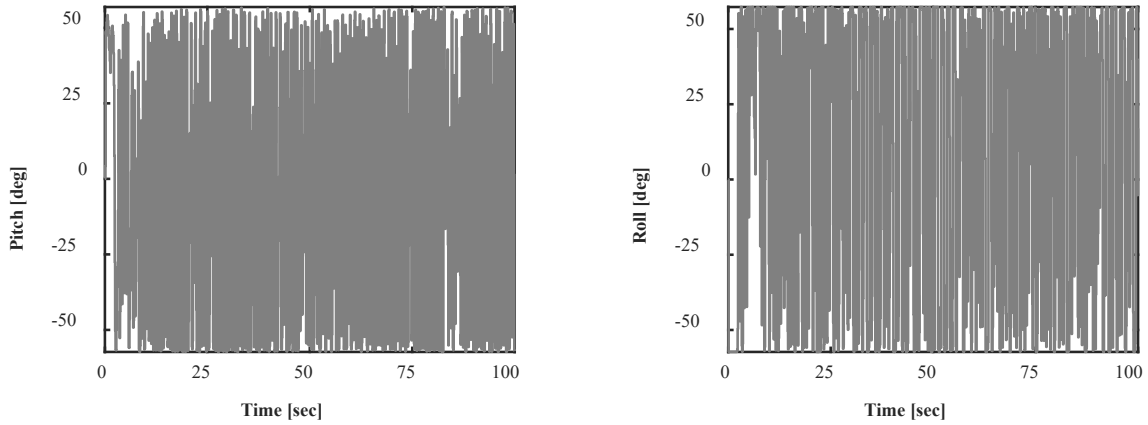

Figure 4.8 Pitch and roll actions performed by the second-generation agent.

The ground track and altitude plots in Figure 4.9 demonstrate the most drastic improvement from the previous agent generation. The ground track has very little error in comparison to the first generation. The UAV immediately turns toward the nearest waypoint and precisely follows the path. However, the altitude track does not perform as well. There are large pitch oscillations present in the solution. This is a result of the error in the roll aerodynamic model. The oscillations would not be so drastic if the UAV did not have to compensate for the instability created by the faulty aerodynamic lookup tables.

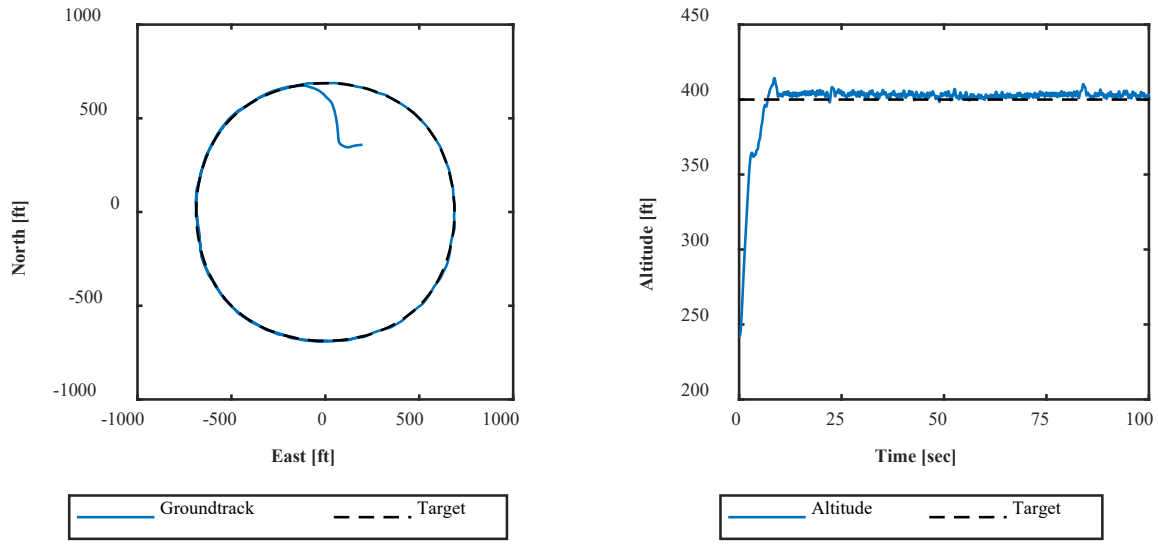

Figure 4.9 Ground track and altitude for the second-generation agent.

Finally, the pitch and roll angles and rates plots can be seen in Figure 4.10. They tell the same story as the altitude plot: excessive pitch oscillation. However, this also shows that despite the ground track having little error, there are also significant oscillations in roll. Most likely, this a consequence of the faulty aerodynamic model again.

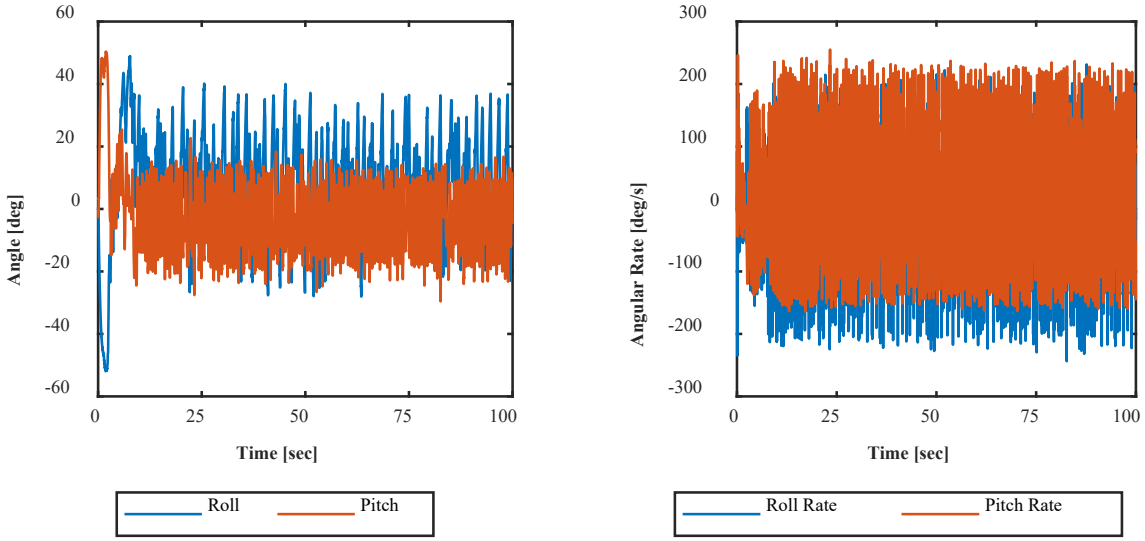

Figure 4.10 Euler angles and rates for the second-generation agent.

4.1.3 Path-Following Agent Generation 3 Results

The third-generation agent does not show a drastic improvement compared to the second-generation agent. Clearly, the episode reward in Figure 4.11 is optimized sooner, but little else is noticeably different.

The clock time in Figure 4.12 is more consistent for the agent. Furthermore, there are significantly less failed episodes compared to the previous generation. This corresponds to the improvement in the reward plot. The agent almost immediately gains high-value experience to follow the ideal path.

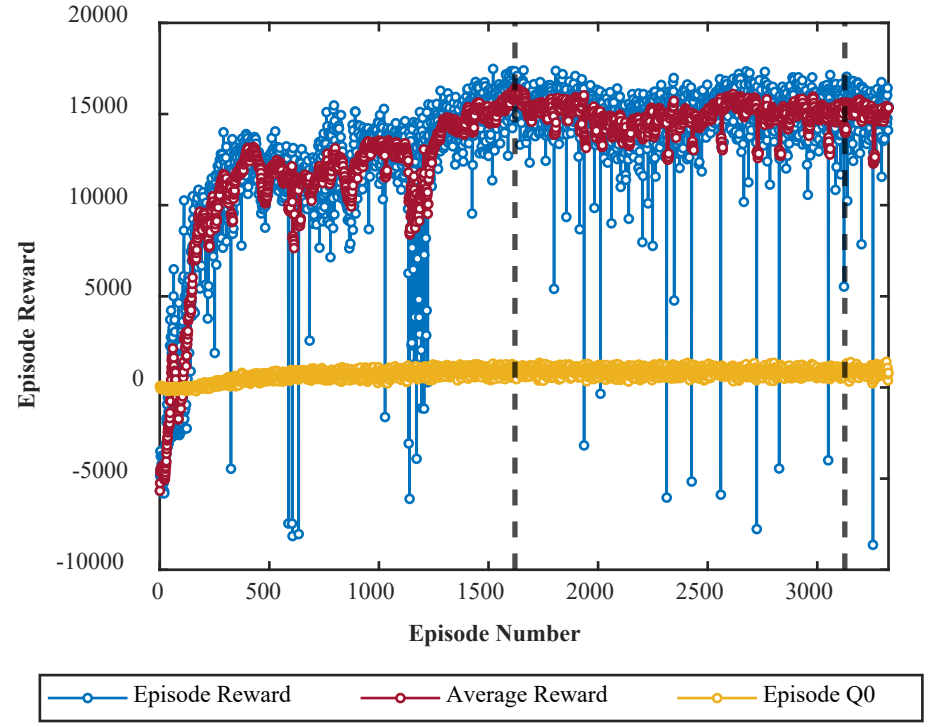

Figure 4.11 Episode rewards for the third-generation agent.

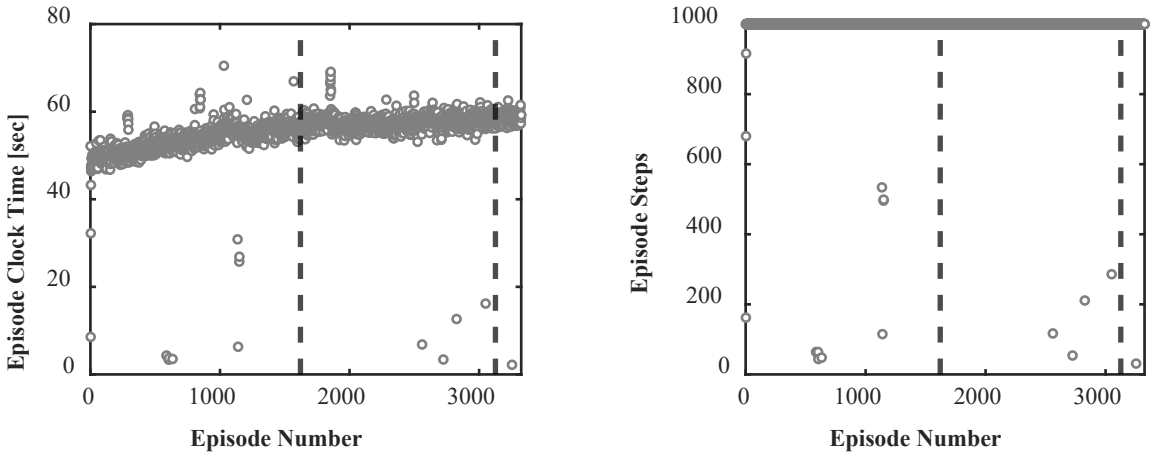

Figure 4.12 Episode clock time and steps for the third-generation agent.

The pitch and roll actions in Figure 4.13, ground track and altitude in Figure 4.14, and pitch and roll angles and rates in Figure 4.15 show no major improvements from the previous generation. This proves that while improvements were made in the rate of learning, the overall performance is still the same. A different approach was tested in Generation 4.

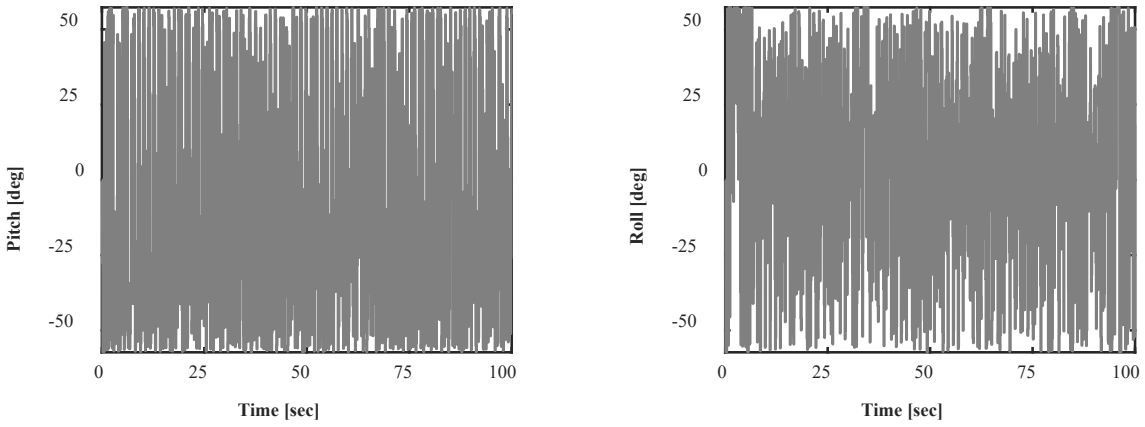

Figure 4.13 Pitch and roll actions performed by the third-generation agent.

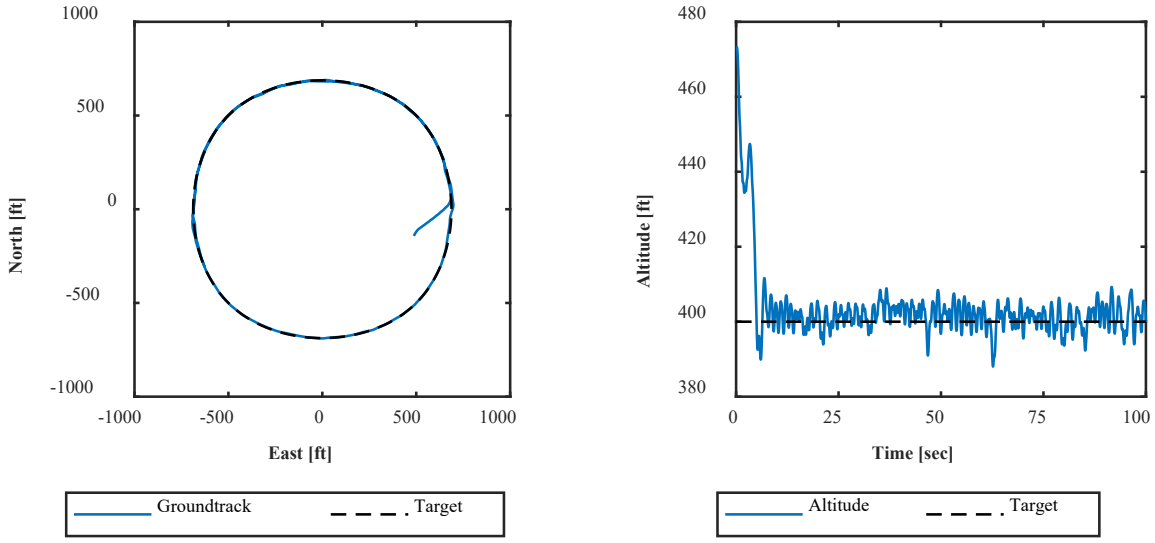

Figure 4.14 Ground track and altitude for the third-generation agent.

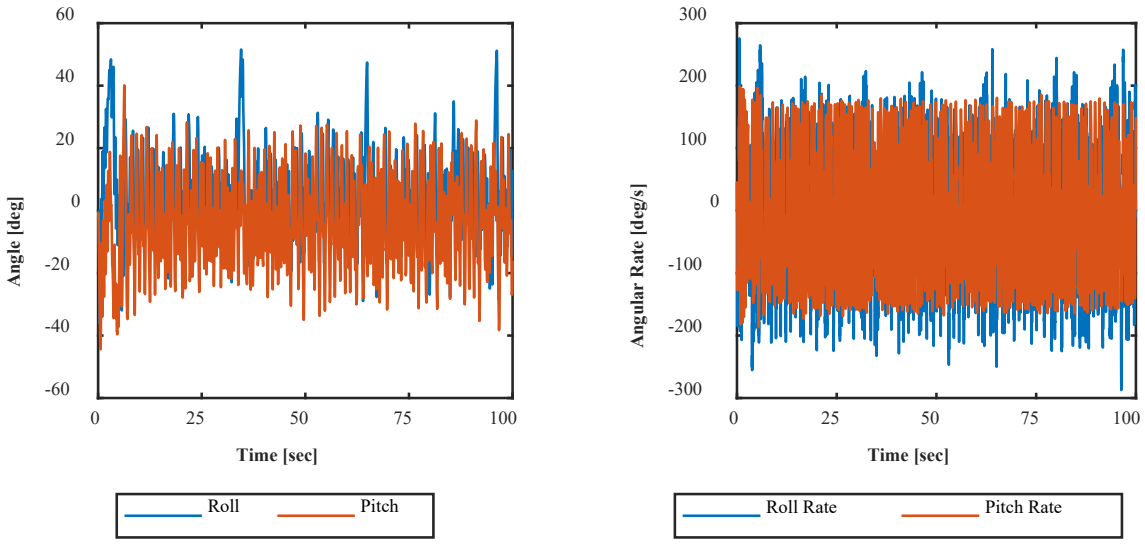

Figure 4.15 Euler angles and rates for the third-generation agent.

4.1.4 Path-Following Agent Generation 4 Results

Unlike the first three generations, the fourth generation must follow a path with a variable target altitude. This path is similar to the path from the previous generation except that the path is "tilted" to more closely resemble a closed-loop dynamic soaring maneuver.

As a consequence of the significant improvements applied to the model, the fourth-generation agent has by far the best performance. Instead of the agent choosing the UAV's attitude as the action, the climb and turn rates are used. Examining Figure 4.16, it is immediately clear that unlike the previous generations, the critic is able to correctly predict the long-term reward. Also, the new actions, paired with a better designed PID controller loop, result in more consistent experiences between episodes. While the reward plateaus between episode 1000 and 2500, it suddenly begins to improve beyond episode 2500.

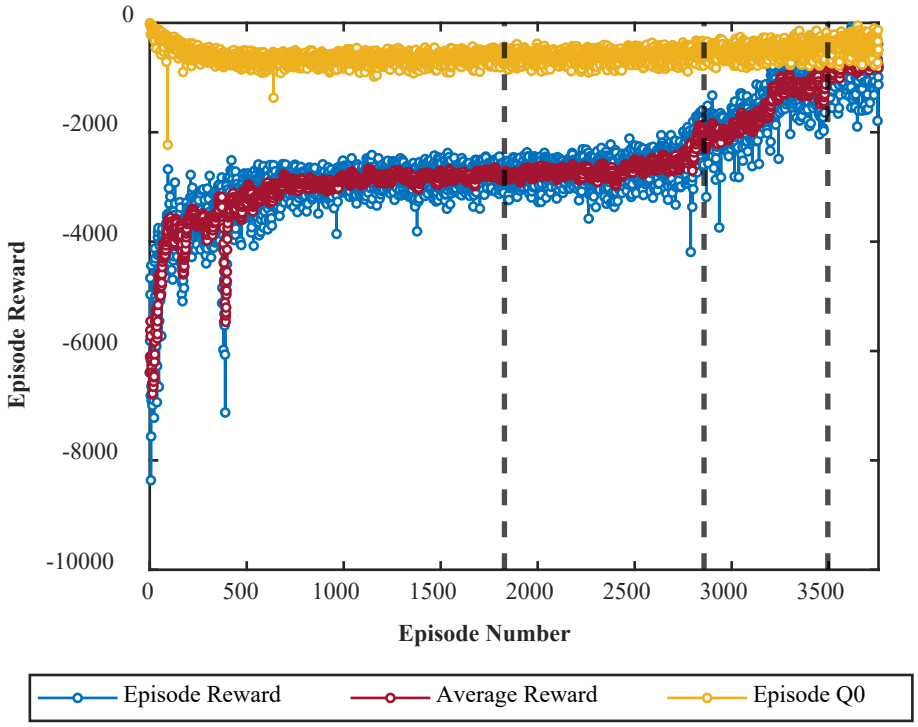

Figure 4.16 Episode reward for the fourth-generation agent.

The effect of more consistent experience can be seen in the episode steps plot in Figure 4.17. Compared to the third-generation agent, there are even fewer failed training episodes. Notably, there are no failed episodes beyond episode 100. Another improvement as a result of the new control method. The clock time plot demonstrates that the agent gained experience consistently for the first 2500 episodes. Episode time increases at a greater rate compared to all previous generations. When the training overcomes the reward plateau around episode 3000, the clock time spikes and then remains consistent. Perhaps this is a result of the convergence of the predicted and true rewards.

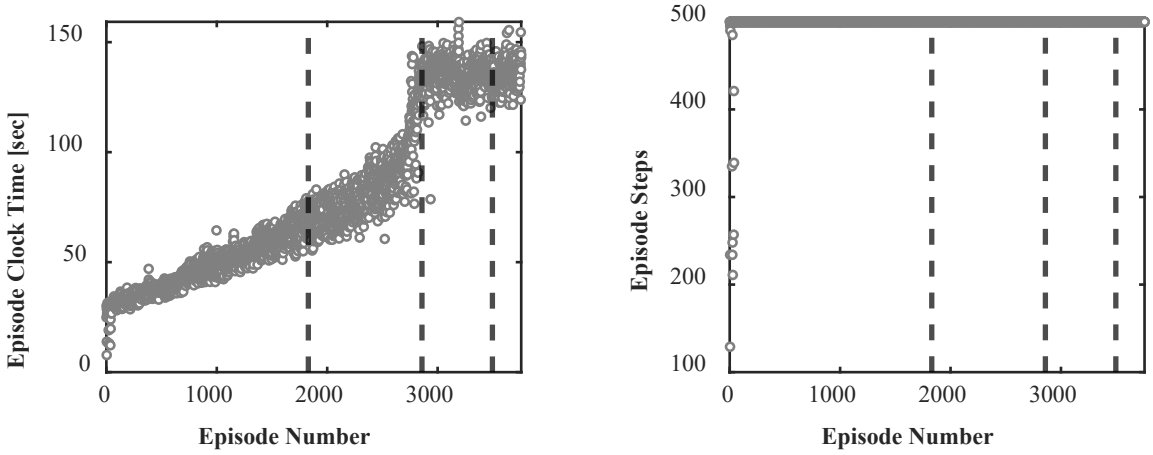

Figure 4.17 Episode clock time and steps for the fourth-generation agent.

In continuation of the trend of improvements, the action plots in Figure 4.18 have significantly less oscillations compared to the pitch and roll actions in the previous generations. Particularly, the climb rate action has almost no oscillations. In contrast, the turn rate action oscillations are too large. Further improvements can be made to prevent this behavior.

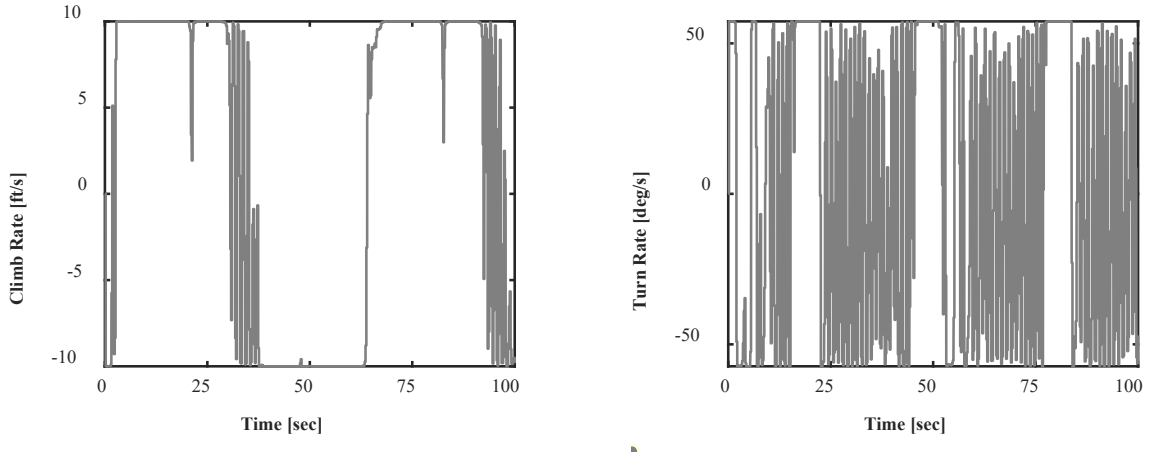

Figure 4.18 Climb and turn rates actions performed by the fourth-generation agent.

The ground track and altitude graph can be seen in Figure 4.19. The agent is able to track the target waypoints closely however there are two smaller circular maneuvers at opposite ends of the maneuver. These are likely a result of the maximum climb rate not being high enough to follow the path. To maximize the reward, the agent performs these smaller circular turns to overshoot the

target altitude. This is made clear by the altitude graph. The target altitude is the altitude of the closest waypoint to the UAV. Noticeably, the slope of the target altitude is higher before and after the small corrective turns compared to the true altitude slope. This reinforces the idea that the maximum climb rate was not sufficient for this maneuver. If either the climb rate was increased or maximum altitude slope decreased, the ground track would not have such corrective turns.

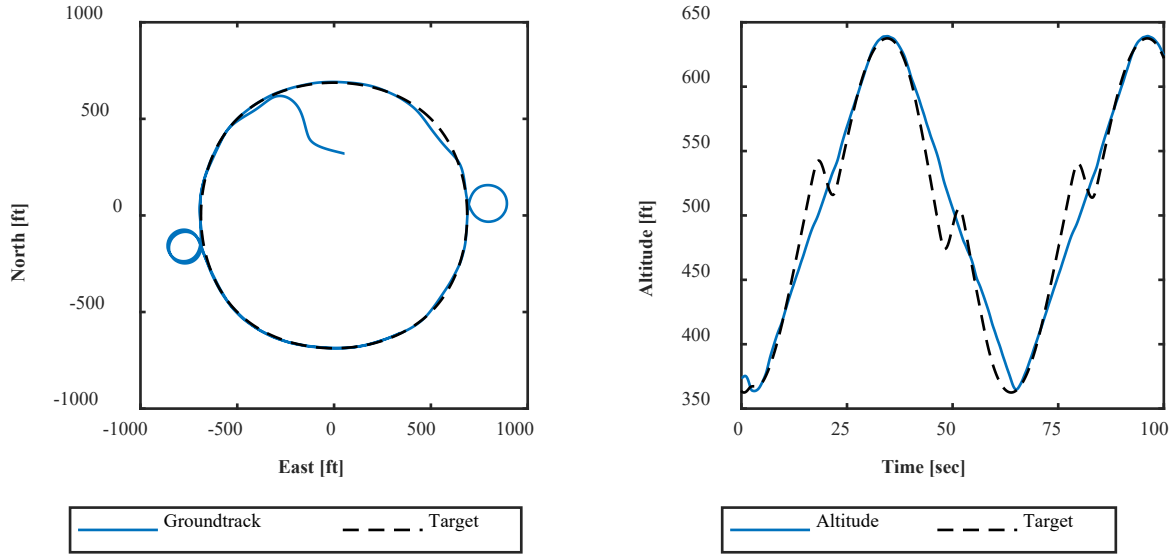

Figure 4.19 Ground track and altitude for the fourth-generation agent.

Evaluation of the Euler angles and rates in Figure 4.20 shows that the magnitude and period of the oscillations have been reduced. Since the pitch and roll angles and rates are determined from the output of the climb rate and turn rate PID controllers, the oscillations are linked to the actions indirectly. Reduction of the oscillations in the actions will also reduce the roll and pitch oscillations.

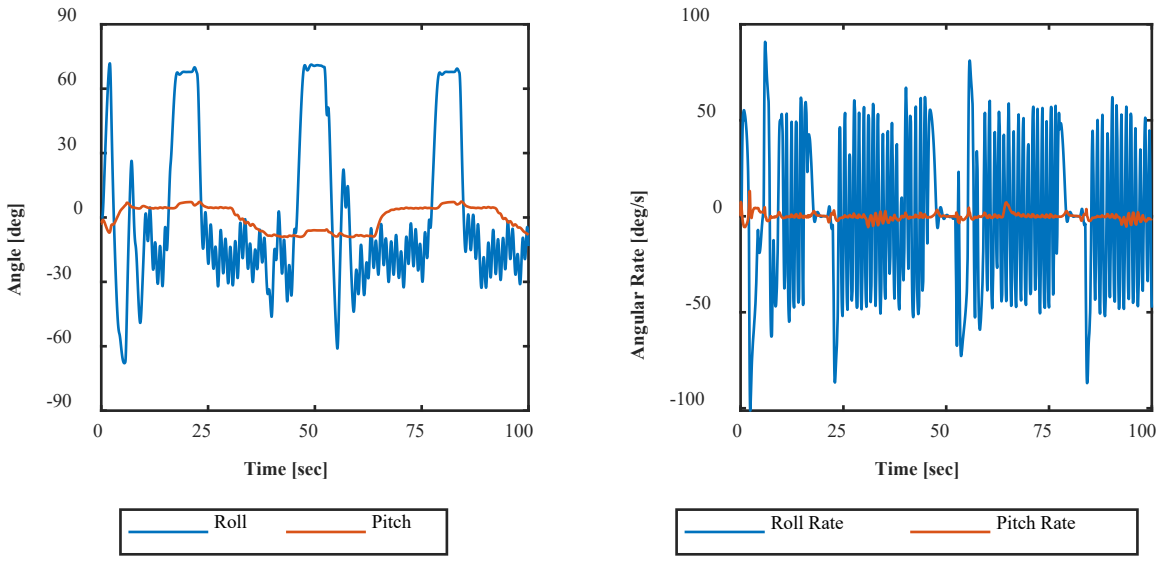

Figure 4.20 Euler angles and rates for the fourth-generation agent.

Finally, to evaluate the robustness of the fourth-generation agent, the waypoint path was modified. In this case, there is no variation in altitude just as with the first three generations. Figure 4.21 presents the results. The agent is able to follow the ground track, albeit not perfectly in comparison to the previous waypoint case. Also, there are no smaller corrective turns as with the previous case, reinforcing the point that they only exist in the solution as a result of the maximum climb rate being too small. However, the altitude performance is extremely poor. It is clear that the agent's experience causes it to mimic the original path with varying altitude. With more training, this could be improved.

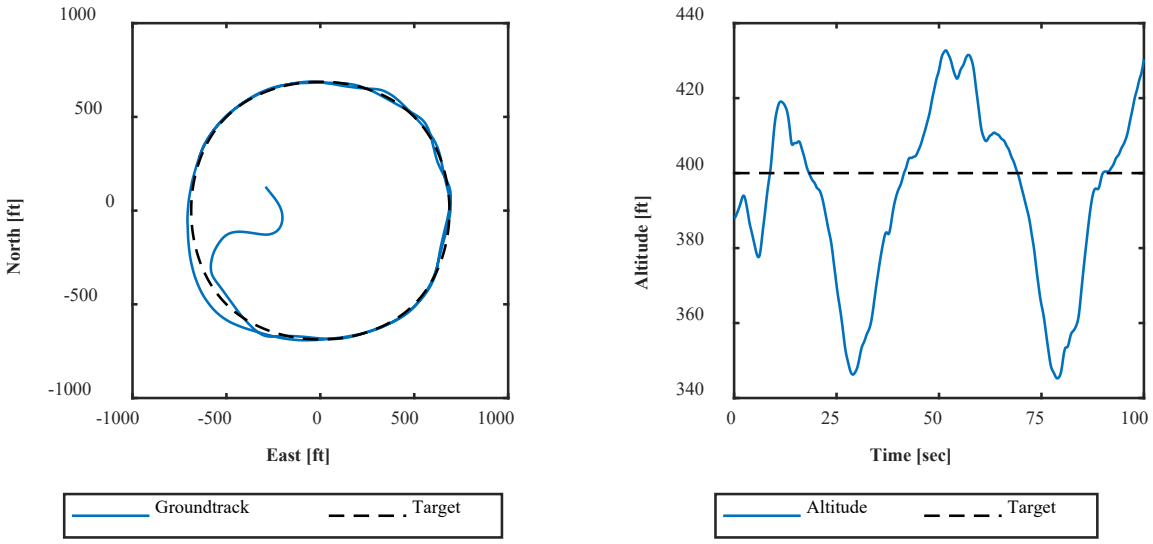

Figure 4.21 Ground track and altitude for second waypoint mission for the fourth-generation agent

4.1.5 Unguided Energy-Seeking Agent Results

Because the fourth-generation path-following agent demonstrated that direct control of the climb and turn rates was the most stable action method, it was also applied to the unguided energy-seeking agent. These actions will allow for stable and reliable exploration of the environment by minimizing the likelihood of a stall. The pitch and roll angles and rates are saturated to further reduce the chances of stall.

The training results can be seen in Figure 4.22. In contrast to the final path-following case, the UES agent struggles to gain consistent experience. However, the reward does not deviate greatly between successive episodes, proving that some type of behavior is being taught by the reward function. This agent was only trained for 700 episodes due to time constraints. However, by this episode, the reward is locally maximal.

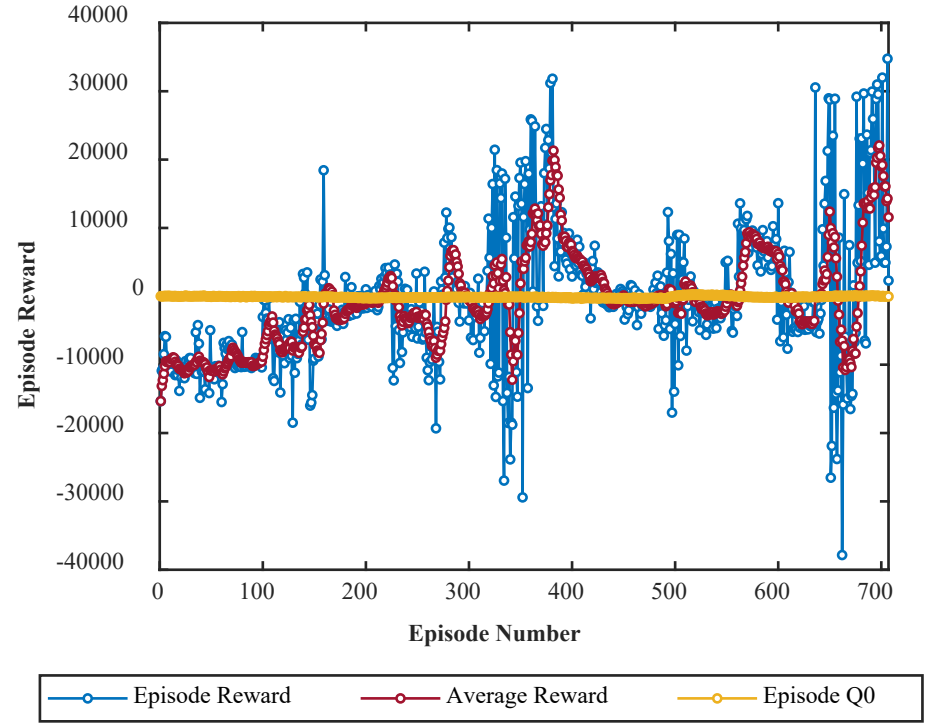

Figure 4.22 Episode reward for the UES agent.

Analysis of the episode clock time and steps in Figure 4.23 shows that the episodes are increasing in length as experience is gained. Near the final episodes, the agent manages to keep the UAV from crashing for the full duration. This shows that the agent is learning how to gain energy from the wind shear in some form.

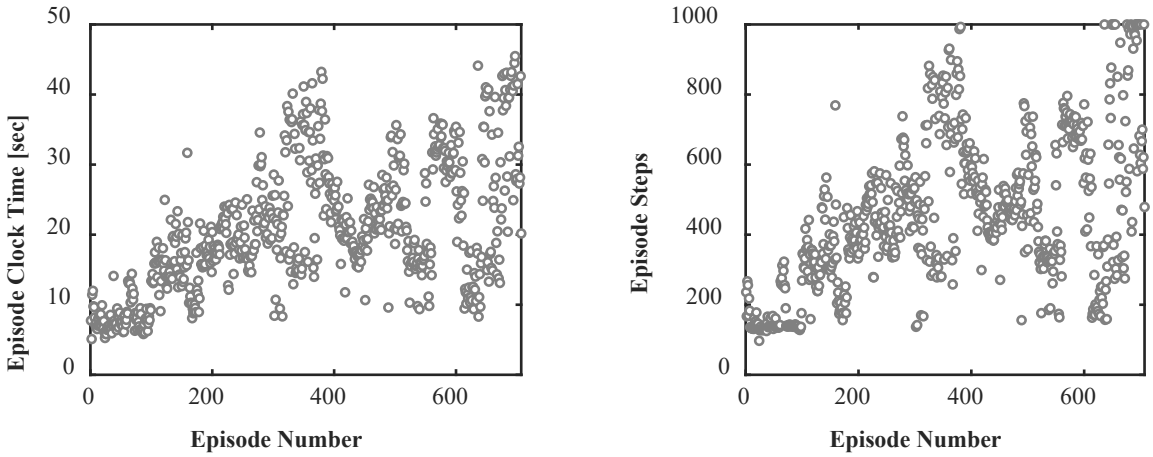

Figure 4.23 Episode clock time and steps for the UES agent.

The resulting climb and turn rate actions from the agent are below in Figure 4.24. These actions are extremely oscillatory unlike the fourth-generation path-following agent. However, this is not necessarily a poor result since the dynamic soaring maneuver may require rapid changes in climb and turn rate. Further analysis of the solution is necessary

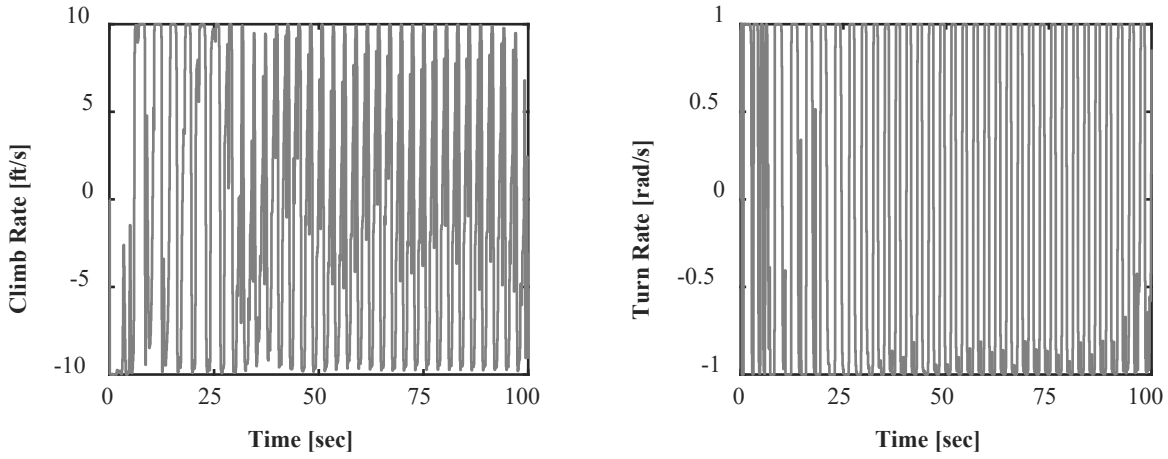

Figure 4.24 Climb and turn rate actions performed by the UES agent.

Figure 4.25 is the resulting ground track and altitude of the UES agent. There are clear signs of repeated cycles in both the ground track and altitude that resembles traveling dynamic soaring. The wind shear in this case is blowing from the south to the north, so the agent should attempt to lose altitude when in a tailwind and gain altitude in a headwind. Unfortunately, this agent was not able to maintain altitude between cycles. But again, from the episode step results clearly some energy is gained from the wind shear to maintain the flight for the full episode duration.

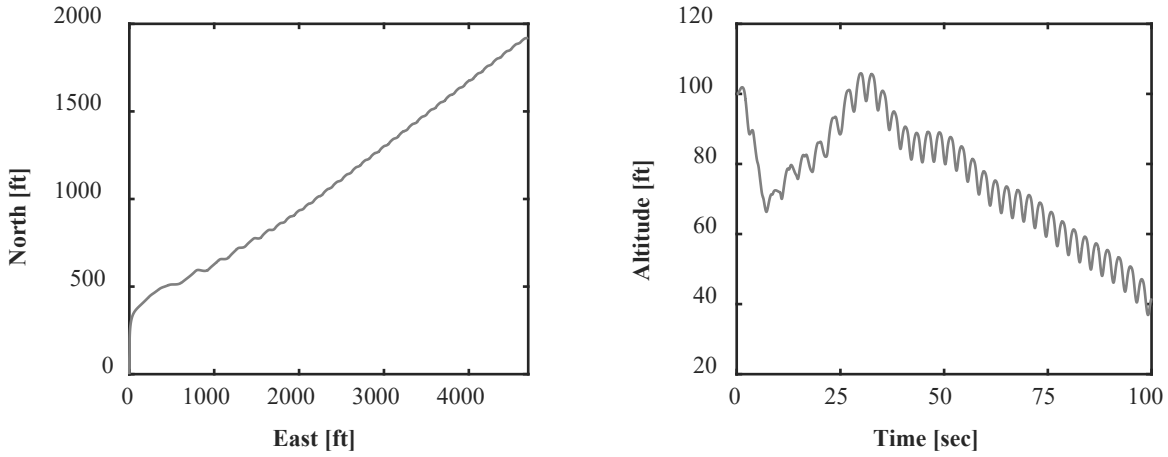

Figure 4.25 Ground track and altitude for the UES agent.

While the altitude is not maintained between cycles, Figure 4.26 shows that the airspeed is maintained. Ground speed is also plotted alongside airspeed.

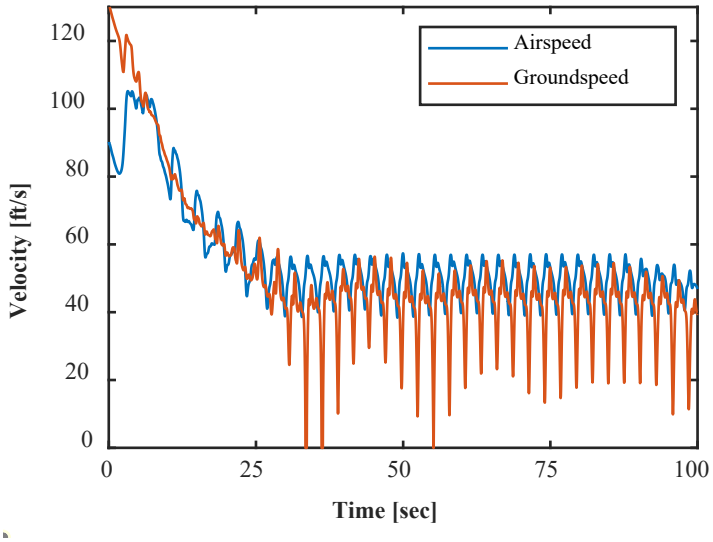

Figure 4.26 Airspeed and ground speed for the UES agent.

Finally, the total energy and total energy rate in Figure 4.27 can be examined. Referring to the reward function from equation 115, the total energy rate should tend to zero. However, overall, it is more negative, causing the total energy to decrease over time.

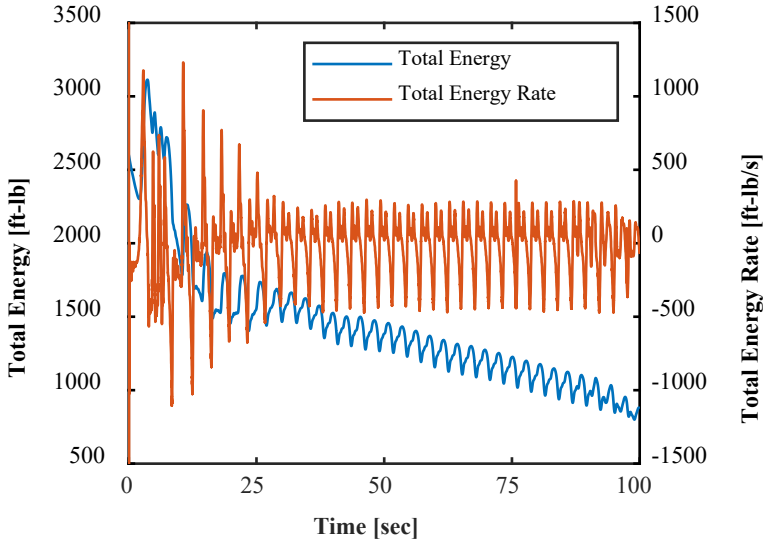

Figure 4.27 Total energy and total energy rate for the UES agent.

It is difficult to deduce from these results whether the issue preventing repeatable dynamic soaring is the wind shear not being sufficient or if the agent simply needs further optimizations through training and reward function design. However, these results show a lot of potential, proving that reinforcement learning can be used for dynamic soaring if the model is properly configured.

5 Discussions, Conclusions, and Recommendations

Reinforcement learning is a powerful tool for intelligent control of complex systems. Dynamic soaring is a great application for this method because it requires optimization against a large number of everchanging variables. In this analysis, the Fox UAV was chosen as a platform to develop reinforcement learning based dynamic soaring simulations and for future application in the real world. A variable-fidelity hybrid-aerodynamic model was calculated for the Fox using RANS computational fluid dynamics and the vortex lattice method. A six-degrees-of-freedom aircraft simulation was developed using the aerodynamic model and reinforcement learning was performed to assess the potential for its application to dynamic soaring. Future application of these methods will use reinforcement learning to perform true sustainable dynamic soaring for a variety of wind shear conditions.

5.1 Discussion

For path-following, control of the UAV using climb rate and turn rate has been proven to be more robust compared to control of the UAV using pitch and roll angles or direct control of the UAV's control surfaces. The UAV is shown to follow a closed-loop path with altitude variation with significantly smaller angular rate oscillations compared to the other methods of UAV control. Additionally, the combination of closed-loop feedback control in this method may allow for the reinforcement learning agent to be deployed on a different UAV without the need for significant retraining. Only the PID controllers need to be tuned to make the new UAV capatable. This characteristic will hopefully allow for the safe deployment of the agent on the real Fox UAV. Additionally, this agent can be improved to adapt to different paths through further reinforcement learning. The next step for this method is to pair the closed-loop path-following agent with a path-optimization agent in a wind shear to develop dynamic soaring cycles. Both agents can be trained at the same time, allowing for the path-following agent to learn to adapt to variations in the path

without constant thrust and for the path-optimization agent to design realistic paths for a given wind shear condition. Finally, this method can be expanded in the future to allow for open-loop "traveling" dynamic soaring.

While dynamic soaring was not fully achieved using the unguided energy seeking method, it has been shown that it is possible with further optimization of the model. By assigning a reward to force the UAV to dive in a tailwind and climb in a headwind assures that the windward climb and leeward descent phases of the dynamic soaring cycle are achieved. Furthermore, applying a reward to the derivative of the total energy with respect to time encourages the UAV to utilize the wind shear near the ground to maintain total energy over time. It is shown that the agent learned to fly in repeated cycles to maintain the airspeed through this method. However, the altitude was not maintained between cycles. Consequently, the total energy is not conserved. Again, further optimization and a more sufficient wind shear may yield better results.

5.2 Conclusions

The results demonstrate that dynamic soaring is feasible using reinforcement learning. Both methods, closed-loop path-following and unguided energy seeking show potential to achieve stable dynamic soaring with further optimizations. In the future, these methods can be expanded to work for a variety of wind shear conditions and for traveling dynamic soaring. The final test will ultimately be in deployment of the reinforcement learning agents on the real Fox UAV to show that RL-based control is stable.

5.3 Recommendations

Extending reinforcement learning methods to the real world will require a series of analyses to prove the application of the control method. Deploying a fully-fledged dynamic soaring agent directly on the Fox UAV without any testing will almost certainly result in a rapid collision with the ground. To prevent this failure, reinforcement learning will have to be integrated in safe steps.

First, a simple test to demonstrate the ability for reinforcement learning to interface with the UAV systems should be performed. As discussed previously, the Fox has a Pixhawk autopilot for as the flight controller and a Raspberry Pi as the companion computer. A simple test could be performed on the ground where the RL agent attempts to control the elevator as a response to the observation of certain states that can be affected without flying such as attitude or airspeed. This process will also be crucial to understanding how to deploy an agent trained offline on the simulator.

The second analysis can be performed in flight. The accuracy of the path following agent can be analyzed by comparing the actions taken by the flight controller to the desired action of the agent. The agent could be trained to follow a simple closed-loop path at a fixed altitude in the simulator. Then, the same path can be followed using the Pixhawk autopilot on the real Fox UAV. While the Fox is controlled by the autopilot, the path-following RL agent will be deployed on the flight computer without the ability to control the UAV. This will allow for the agent to observe the state of the UAV and choose actions based on those states. Finally, the actions taken by the agent should be compared to the real inputs from the autopilot to assess the performance of the pretrained agent. If the actions are reasonably close, then the next step can be progressed to.

After testing the RL agent without allowing it to control the UAV, the next logical step is to allow it to take direct control of the UAV. There are two methods that can be applied here. First is to again control the UAV using the traditional autopilot. In this case, however, the agent will use guided reinforcement learning to gain experience directly from the autopilot commands. This method can help transition the agent from simulation to real-world safely. Alternatively, if the previous analysis was extremely successful, the agent can be train using unguided reinforcement

learning in the real world. Once reinforcement learning based control of the UAV is achieved, then real world dynamic soaring can be attempted.

For the Fox UAV to dynamically soar as does the Albatross, a sufficient wind shear needs to be present. Of course, as discussed earlier, the closed-loop path-following reinforcement learning agent should be able to perform dynamic soaring like maneuvers with or without the presence of wind shear. The UAV will need to be able to intelligently switch between a normal cruise flying mode and dynamic soaring. Initially, the agent will assume no wind shear is present, performing the most conservative maneuver possible using engine power to prevent stalling during the windward climb. During subsequent cycles, the path will be optimized by the path-optimizing agent to reflect the true wind shear. Over time, the UAV will either perform true dynamic soaring without using internal thrust or will simply continue to perform dynamic soaring-like maneuver that are more energy efficient compared to flying at cruise speed.

6 REFERENCES

[1] Joseph, G., "Design and Flight-Path Simulation of a Dynamic-Soaring UAV," Embry-Riddle Aeronautical University, 2021.

[2] Chung, J., J., Lawrence, N. R.J., Sukkarieh, S., "Learning to Soar: Resource-constrained exploration in reinforcement learning," The International Journal of Robotics Research, February 2015.

[3] Reddy, g., Wong-Ng, j., Celani, A., Sejnowski, T. J., Vergassola, M., "Glider Soaring via Reinforcement Learning in the Field," Nature, Vol. 562, 2018.

[4] Woodbury, T., Dunn, C., Valasek, J., "Autonomous Soaring using Reinforcement Learning for Trajectory Generation," AIAA SciTech Forum, 2014.

[5] Perez, R. E., Arnal, J., Jansen, P. W., "Nero-Evolutionary Control for Optimal Dynamic Soaring," AIAA SciTech Forum, 2020.

[6] Barate, R., Doncieux, S., Meyer, J. A., "Design of a bio-inspired controller for dynamic soaring in a simulated UAV", Bioinspiration & Biomimetics, October 2006.

[7] Montella, C., Spletzer, J., "Reinforcement Learning for Autonomous Dynamic Soaring in Shear Winds," IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014.

[8] Boslough, M., "Autonomous Dynamic Soaring," IEEE Aerospace Conference, 2017.

[9] Zhu, B. J., Hou, Z. X., Ouyang, H. J., "Trajectory Optimization of Unmanned Aerial Vehicle in Dynamic Soaring," Journal of Aerospace Engineering, 2017.

[10] Montella, C., "An End-to-End Platform for Autonomous Dynamic Soaring in Wind Shear," Lehigh University, January 2019.

[11] Bronz, M., Gavrilovic, N., Drouin, A., Hattenberger, G., Moschetta, J. M., "Flight Testing of Dynamic Soaring Part-1: Leeward Inclined Circle Trajectory," AIAA SciTech Forum, January 2021.

[12] Lyon, C., Broeren, A., Giguere, P., Gopalarathnam, A., Selig, M., "Summary of Low-Speed Airfoil Data Vol. 3," SoarTech Publications, 1997.

[13] Stevens, B. L., Lewis, F. L., Johnson, E. N., "Aircraft Control and Simulation," 3rd ed., John Wiley & Sons, Inc., New Jersey, 2016.

[14] Frost, W., Bowles, R. L., "Wind Shear Terms in the Equations of Aircraft Motion," Journal of Aircraft, Vol. 21, Nov. 1984, pp. 866-872.

[15] Military Specification MIL-F-8785C, "Flying Qualities of Piloted Airplanes," November 1980.

[16] Sutton, R. S., McAllester, D., Singh, S., Mansour, Y., “Policy Gradient Methods for Reinforcement Learning with Function Approximation,” Advances in Neural Processing Systems, 1999.

[17] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M., “Deterministic Policy Gradient Algorithms,” International Conference on Machine Learning, 2014.

[18] Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., “Continuous Control with Deep Reinforcement Learning,” International Conference on Machine Learning, 2016.

7 APPENDIX A – CFD Solution Settings

Table 7.1 2D CFD setup information.

2D Methods			
	VARIABLE	**VALUE**	**UNIT**
Grid	Cell Count	22885	-
	Wall Spacing	1.85E-05	m
Boundary Conditions	Type	Pressure Farfield	-
	Mach #	0.053	-
	AoA	-10 to 20	deg
	Temperature	300	K
	Operating Pressure	101325	Pa
	Gauge Pressure	0	Pa
	Turbulent Intensity	5	%
	Turb Viscosity Ratio	10	-
Reference Values	Area	0.257	m^2
	Density	1.1768	kg/m^3
	Depth	0	m
	Enthalpy	2029.449	j/kg
	Length	0.257	m
	Pressure	0	Pa
	Temperature	300	k
	Velocity	18.31	m/s
	Viscosity	1.8462E-05	kg/m*s
	Specific Heat Ratio	1.4	-
Model Information	Solver	Pressure-Based	-
	Energy Equation	on	-
	Viscous Model	Variable	-
	Numerical Scheme	AUSM	-
	Spatial Accuracy	Second-order	-

Table 7.2 3D CFD setup information.

3D Methods			
	VARIABLE	**VALUE**	**UNIT**
Grid	Cell Count	1934318	-
	Wall Spacing	9.25E-05	m
Boundary Conditions	Inlet	Velocity Inlet	-
	Outlet	Pressure Outlet	-
Case Information	Inlet Velocity	18.313	m/s
	Mach #	0.053	-
	AoA	-40 to 40	deg
	Sideslip	-40 to 40	deg
	Roll	-90 to 90	deg
	Temperature	300	K
	Operating Pressure	101325	Pa
	Gauge Pressure	0	Pa
	Turbulent Intensity	5	%
	Turb Viscosity Ratio	10	-
Reference Values	Area	0.7465	m^2
	Density	1.1768	kg/m^3
	Enthalpy	0	j/kg
	Length	0.257	m
	Pressure	0	Pa
	Temperature	300	k
	Velocity	18.313	m/s
	Viscosity	1.8462E-05	kg/m*s
	Specific Heat Ratio	1.4	-
Model Information	Solver	Pressure-Based	-
	Energy Equation	off	-
	Viscous Model	k-omega SST	-
	Numerical Scheme	SIMPLE	-

8 APPENDIX B – Summary of aerodynamic coefficients and derivatives

Table 8.1 Angle of attack static lookup table for the Fox UAV.

AOA [deg]	C_D	C_Y	C_L	C_l	C_M	C_N
-40	7.535E-01	-4.452E-03	-8.142E-01	1.825E-03	5.458E-01	-2.567E-03
-30	5.083E-01	1.701E-03	-7.258E-01	-3.605E-03	4.268E-01	5.095E-03
-25	3.878E-01	-3.265E-04	-6.344E-01	1.690E-03	3.555E-01	1.448E-03
-20	2.792E-01	-5.834E-04	-5.234E-01	-2.812E-03	2.872E-01	-3.528E-03
-18	2.412E-01	1.193E-04	-4.833E-01	-2.265E-03	2.777E-01	-1.862E-03
-16	2.032E-01	-6.132E-04	-4.509E-01	-4.237E-03	2.897E-01	-2.905E-03
-14	1.623E-01	-3.812E-04	-4.392E-01	-5.514E-04	3.933E-01	-8.674E-04
-12	1.281E-01	-3.762E-04	-3.712E-01	1.638E-03	3.358E-01	-1.580E-04
-10	9.876E-02	-9.944E-04	-3.016E-01	6.707E-04	2.713E-01	-1.702E-03
-8	5.134E-02	4.707E-04	-3.125E-01	-4.256E-04	1.585E-01	-1.728E-03
-6	3.000E-02	-7.632E-04	-1.468E-01	1.087E-03	8.607E-02	-1.412E-03
-4	2.683E-02	3.812E-05	3.954E-02	1.580E-03	4.799E-02	-4.063E-04
-2	2.732E-02	-6.397E-04	2.276E-01	3.095E-04	1.085E-02	-3.918E-04
0	3.082E-02	-2.958E-04	4.146E-01	7.739E-04	-1.982E-02	2.015E-06
2	3.731E-02	-3.050E-04	5.958E-01	-1.038E-03	-3.932E-02	-7.126E-04
4	4.687E-02	-1.008E-03	7.756E-01	-5.224E-04	-7.291E-02	-3.050E-03
6	5.944E-02	-4.027E-04	9.396E-01	-1.533E-03	-1.144E-01	-1.606E-03
8	8.476E-02	-1.213E-03	1.028E+00	-7.497E-04	-2.280E-01	-2.502E-03
10	1.108E-01	2.551E-03	1.131E+00	1.702E-03	-2.821E-01	1.912E-03
12	1.390E-01	-4.293E-03	1.215E+00	-5.293E-03	-3.496E-01	-3.237E-03
14	1.849E-01	-2.711E-03	1.144E+00	-2.520E-02	-4.116E-01	-1.228E-02
16	2.410E-01	4.530E-03	1.077E+00	3.527E-03	-4.217E-01	1.575E-02
18	3.026E-01	-1.041E-03	9.709E-01	-2.670E-03	-3.389E-01	-5.224E-04
20	3.629E-01	1.114E-03	9.573E-01	-9.106E-03	-3.828E-01	-3.095E-03
25	5.016E-01	-5.990E-03	9.684E-01	5.843E-03	-4.288E-01	-1.827E-02
30	6.347E-01	1.435E-02	9.984E-01	4.281E-02	-4.912E-01	-2.672E-02
40	8.995E-01	2.609E-02	1.008E+00	4.908E-02	-6.102E-01	1.055E-02

Table 8.2 Sideslip static aerodynamic coefficient lookup table for the Fox UAV.

Beta [deg]	**C_D**	**C_Y**	**C_L**	**C_l**	**C_M**	**C_N**
-40	2.447E-01	2.927E-01	2.661E-01	7.711E-02	-7.532E-02	5.296E-01
-30	1.500E-01	2.449E-01	3.451E-01	3.285E-02	-1.066E-01	4.874E-01
-25	1.059E-01	1.974E-01	3.646E-01	2.868E-02	-7.397E-02	4.179E-01
-20	7.524E-02	1.548E-01	3.841E-01	2.244E-02	-5.217E-02	3.361E-01
-18	6.565E-02	1.385E-01	3.898E-01	2.073E-02	-4.095E-02	3.042E-01
-16	5.743E-02	1.238E-01	3.951E-01	1.681E-02	-3.485E-02	2.779E-01
-14	4.887E-02	1.130E-01	4.020E-01	5.272E-03	-3.953E-02	2.637E-01
-12	4.155E-02	9.347E-02	4.063E-01	3.998E-03	-4.091E-02	2.169E-01
-10	3.767E-02	7.621E-02	4.077E-01	4.340E-03	-3.072E-02	1.795E-01
-8	3.495E-02	6.058E-02	4.091E-01	4.153E-03	-2.265E-02	1.436E-01
-6	3.291E-02	4.508E-02	4.097E-01	3.121E-03	-1.740E-02	1.072E-01
-4	3.154E-02	2.961E-02	4.111E-01	1.999E-03	-1.465E-02	6.991E-02
-2	3.096E-02	1.350E-02	4.118E-01	8.770E-04	-1.662E-02	3.005E-02
0	3.082E-02	-2.958E-04	4.146E-01	7.739E-04	-1.982E-02	2.015E-06
2	3.096E-02	-1.350E-02	4.118E-01	-8.770E-04	-1.662E-02	-3.005E-02
4	3.154E-02	-2.961E-02	4.111E-01	-1.999E-03	-1.465E-02	-6.991E-02
6	3.291E-02	-4.508E-02	4.097E-01	-3.121E-03	-1.740E-02	-1.072E-01
8	3.495E-02	-6.058E-02	4.091E-01	-4.153E-03	-2.265E-02	-1.436E-01
10	3.767E-02	-7.621E-02	4.077E-01	-4.340E-03	-3.072E-02	-1.795E-01
12	4.155E-02	-9.347E-02	4.063E-01	-3.998E-03	-4.091E-02	-2.169E-01
14	4.887E-02	-1.130E-01	4.020E-01	-5.272E-03	-3.953E-02	-2.637E-01
16	5.743E-02	-1.238E-01	3.951E-01	-1.681E-02	-3.485E-02	-2.779E-01
18	6.565E-02	-1.385E-01	3.898E-01	-2.073E-02	-4.095E-02	-3.042E-01
20	7.524E-02	-1.548E-01	3.841E-01	-2.244E-02	-5.217E-02	-3.361E-01
25	1.059E-01	-1.974E-01	3.646E-01	-2.868E-02	-7.397E-02	-4.179E-01
30	1.500E-01	-2.449E-01	3.451E-01	-3.285E-02	-1.066E-01	-4.874E-01
40	2.447E-01	-2.927E-01	2.661E-01	-7.711E-02	-7.532E-02	-5.296E-01

Table 8.3 Roll angle static aerodynamic coefficient lookup table for the Fox UAV.

Roll [deg]	**C_D**	**C_Y**	**C_L**	**C_l**	**C_M**	**C_N**
-90	3.050E-02	4.186E-01	-1.193E-04	-3.224E-04	2.580E-05	2.197E-02
-80	3.054E-02	4.117E-01	7.259E-02	5.643E-05	-3.785E-03	2.179E-02
-70	3.048E-02	3.928E-01	1.429E-01	-3.627E-04	-7.452E-03	2.015E-02
-60	3.027E-02	3.623E-01	2.090E-01	2.048E-04	-1.033E-02	1.866E-02
-50	3.046E-02	3.201E-01	2.689E-01	-9.673E-06	-1.310E-02	1.607E-02
-40	3.043E-02	2.691E-01	3.201E-01	-3.692E-04	-1.591E-02	1.373E-02
-30	3.037E-02	2.086E-01	3.606E-01	-9.560E-04	-1.732E-02	1.019E-02
-20	3.041E-02	1.434E-01	3.931E-01	1.419E-04	-2.004E-02	7.058E-03
-10	3.032E-02	7.288E-02	4.116E-01	-4.127E-04	-2.091E-02	3.452E-03
0	3.082E-02	-2.958E-04	4.146E-01	7.739E-04	-1.982E-02	2.015E-06
10	3.032E-02	-7.288E-02	4.116E-01	4.127E-04	-2.091E-02	-3.452E-03
20	3.041E-02	-1.434E-01	3.931E-01	-1.419E-04	-2.004E-02	-7.058E-03
30	3.037E-02	-2.086E-01	3.606E-01	9.560E-04	-1.732E-02	-1.019E-02
40	3.043E-02	-2.691E-01	3.201E-01	3.692E-04	-1.591E-02	-1.373E-02
50	3.046E-02	-3.201E-01	2.689E-01	9.673E-06	-1.310E-02	-1.607E-02
60	3.027E-02	-3.623E-01	2.090E-01	-2.048E-04	-1.033E-02	-1.866E-02
70	3.048E-02	-3.928E-01	1.429E-01	3.627E-04	-7.452E-03	-2.015E-02
80	3.054E-02	-4.117E-01	7.259E-02	-5.643E-05	-3.785E-03	-2.179E-02
90	3.050E-02	-4.186E-01	-1.193E-04	3.224E-04	2.580E-05	-2.197E-02

Table 8.4 Roll rate damping derivative lookup table for the Fox UAV.

AoA [deg]	C_{l_p}	C_{Y_p}	C_{N_p}
-2.778	-5.6940E-01	4.8432E-02	-3.3039E-02
-2.360	-5.6919E-01	4.8601E-02	-3.3120E-02
-1.776	-5.6990E-01	4.8759E-02	-3.3095E-02
-0.9234	-5.6977E-01	4.8928E-02	-3.3118E-02
0.3927	-5.7410E-01	4.9316E-02	-3.3123E-02
2.532	-5.7614E-01	4.9891E-02	-3.3281E-02
6.593	-5.8955E-01	5.0902E-02	-3.3654E-02
16.18	-6.3724E-01	4.9647E-02	-3.4210E-02
25.14	-7.1281E-01	4.9722E-02	-3.6229E-02

Table 8.5 Pitch rate damping derivative lookup table for the Fox UAV.

AoA [deg]	C_{L_q}	C_{D_q}	C_{M_q}
-2.778	9.5689E+00	7.0661E-02	-1.8261E+01
-2.360	9.4968E+00	8.9401E-02	-1.8265E+01
-1.776	9.3592E+00	1.1879E-01	-1.8264E+01
-0.9234	9.4739E+00	1.6488E-01	-1.8262E+01
0.3927	9.3690E+00	2.3103E-01	-1.8263E+01
2.532	9.6508E+00	3.4921E-01	-1.8277E+01
6.593	9.6738E+00	5.6283E-01	-1.8336E+01
16.18	9.3985E+00	9.7205E-01	-1.8976E+01
25.14	8.5543E+00	1.2014E+00	-1.9555E+01

Table 8.6 Yaw rate damping derivative lookup table for the Fox UAV.

AoA [deg]	C_{l_r}	C_{Y_r}	C_{N_r}
-2.778	8.3999E-02	2.7420E-01	-8.8173E-02
-2.360	8.9403E-02	2.7407E-01	-8.8039E-02
-1.776	9.8348E-02	2.7393E-01	-8.7667E-02
-0.9234	1.0734E-01	2.7371E-01	-8.7449E-02
0.3927	1.2158E-01	2.7334E-01	-8.6752E-02
2.532	1.4506E-01	2.7276E-01	-8.5761E-02
6.593	1.9222E-01	2.7174E-01	-8.4003E-02
16.18	2.8445E-01	2.6906E-01	-7.9988E-02
25.14	3.6152E-01	2.6733E-01	-7.6819E-02

Table 8.7 Control surface coefficients for the Fox UAV.

Coefficient	Value
$C_{D_{\delta_e}}$	1.815E-02
$C_{D_{\delta_f}}$	4.235E-02
$C_{Y_{\delta_a}}$	-3.065E-02
$C_{Y_{\delta_r}}$	-2.397E-01
$C_{L_{\delta_e}}$	6.240E-01
$C_{L_{\delta_f}}$	1.368E+00
$C_{l_{\delta_a}}$	3.556E-01
$C_{M_{\delta_e}}$	-2.200E+00
$C_{N_{\delta_a}}$	-1.539E-02
$C_{N_{\delta_r}}$	-8.129E-02