2019

An End-to-End Platform for Autonomous Dynamic Soaring in Wind Shear

Corey Montella
Lehigh University, cmontella@live.com

An End-to-End Platform for Autonomous Dynamic Soaring in Wind Shear

by

Corey Montella

A Dissertation
Presented to the Graduate and Research Committee
of Lehigh University
in Candidacy for the Degree of
Doctor of Philosophy
in
Computer Engineering

Lehigh University

January 2019

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

_____

Date

_____

Dr. John Spletzer,

Dissertation Director, Chair

_____

Accepted Date

Committee Members:

_____

Dr. Héctor Muñoz-Avila

_____

Dr. Mooi Choo Chuah

_____

Dr. Joachim Grenestedt

Acknowledgements

Thank you to my advisor John Spletzer for his patience through my destruction of airframes and avionics equipment.

Thank you to Joachim Grenestedt, Bill Maroun, and Ted Bowen, for repairing said destroyed equipment.

Thank you to Jack Langelaan, John Bird, Nate Depenbusch, and Jack Quindlen for their support in flight testing

This dissertation is dedicated to my wife, Rachel. I couldn't have done this without you.

Contents

List of Tables viii

List of Figures ix

Abstract 1

Chapter 1 – Introduction 3

1.1. Dynamic Soaring of Birds 5

1.2. Applications of Dynamic Soaring 6

1.3. Dissertation Objectives 9

1.4. Dissertation Contributions 10

1.5. Dissertation Outline 11

Chapter 2 – Background 13

2.1. Reinforcement Learning Framework 13

2.2. Approaches to Reinforcement Learning 15

2.3. Challenges for Mobile Robots 20

2.4. Tractability 22

2.5. Recent Trends in Robot Control 25

Chapter 3 – Fox Soaring Platform 28

3.1. Related Work 28

3.2. Chapter Contributions 29
3.3. Design Considerations 30
3.4. Airframe 31
3.5. Hardware Architecture 33
3.6. Portable Ground Station 42
3.7. Software Architecture 43
3.8. Platform Validation 48
3.9. Flight Tests 66
3.10. Summary 67

Chapter 4 – Trajectory Generation and Following 68

4.1. Related Work 68
4.2. Chapter Contributions 70
4.3. Aircraft Motion Model 71
4.4. Atmosphere Model 73
4.5. Global Planning 75
4.6. Local Planning 83
4.7. Experiments 87
4.8. Summary and Discussion 97

Chapter 5 – Learned Controller 99

5.1. Related Work 100
5.2. Chapter Contributions 101
5.3. Problem Formulation 101
5.4. Experiments 106

5.5. Summary and Discussion.....119

Chapter 6 – Discussion & Future Work..... 122

References 126

Appendices 132

Vita 135

List of Tables

3.1 A summary of the 8 directions and angles of the signal strength test. Each direction and angle of the aircraft with respect to the ground was tested at 200m. The -60 degree roll test was chosen to represent the maximum angle of roll we expect to see during dynamic soaring maneuvers. 54

3.2 Summary statistics for the signal strength test 56

3.3 Summary statistics for the result of the vibration test 61

4.1 Summary of aircraft parameters used in the point-mass aircraft model. 73

4.2 Summary of atmosphere parameters used in the point-mass aircraft model. 73

4.3 Linear equality constraints of the dynamic soaring trajectory solver . . 76

4.4 Nonlinear inequality constraints of the dynamic soaring trajectory solver 77

5.1 Results for the point mass model simulation after learning converged. Means are computed for 10 episodes for each controller 109

5.2 The dynamic soaring sample-based controller performance compared to the reinforcement learning controller. The sample-based controller takes 6 cycles to reach a stead state, while the learning controller takes only 4. 113

List of Figures

1.1 A sketch of a bird performing a soaring maneuver in Leonardo da Vinci's Manuscript E (circa 1513). 4

1.2 A shy albatross performing a dynamic soaring maneuver. Its massive wingspan of up to 3.7m and weight of up to 5kg allow to harness the power of wind shear blowing over the ocean to gain kinetic energy without flapping its wings 5

1.3 A dynamic soaring loiter maneuver. A glider with 2.8m wingspan is pictured to scale in red. There is an implicit shear wind in this figure blowing from the back wall of the picture. Below 100 m altitude the wind is 0, and it increased to a value of w_{max} over a height of Δz. 9

2.1 Q Learning algorithm pseudocode. 19

3.1 Fox dynamic soaring platform balanced on a center of mass calibration device. The Fox has a wingspan of 2.8 m and an empty mass of 2.90kg. 31

3.2 A schematic of the Fox soaring airframe. It features ailerons on each wing, an elevator, and a rudder as control surfaces. A pitot tube is mounted on the right wing. Static pressure sensors are installed behind the wings on either side. A single brushless motor at the nose of the glider powers the aircraft for takeoff and landing, but is not operational during dynamic soaring maneuvers. 32

3.3 A custom steel chassis for avionics. It is installable in the fuselage of the glide. Pictured here is the Piccolo SL autopilot mounted on rubber standoffs to isolate it from vibrations. The two plastic tubes connect to the static and dynamic portions of the pitot sensing system. The antenna pictured here was a preliminary design that was abandoned in favor of a whip antenna that extends along the fuselage into the vertical stabilizer 34

3.4 The pitot sensing system. (right) A steel tube is mounted on the wing and painted red for contrast against the ground. This steel tube is connected to a rubber tube that extends through the wing into the fuselage. (left) static pressure sensors are mounted behind the wings on either side of the fuselage. 36

3.5 The full autonomy subsystem. Pictured here, the On-Boar-Computer (OBC) is mounted on the steel chassis. The OBC is an Advantech PCM-3363 PC/104 single board computer running an Ubuntu operating system and MATLAB computer programming language. 37

3.6 (top) logical connections between the Piccolo SL autopilot and on-board-computer OBC. This is a serial interface. (bottom) the physical connection between the autopilot and OBC. 38

3.7 Power discharge of OBC as it idles after boot. This represents the maximum running time of the OBC. Nominal voltage of the battery is 12V with a capacity of 2200mAh. The battery was charged to 12.60 V and the OBC was powered until the battery discharged. The final voltage was 10.50V. Total running time was 135 minutes. 39

3.8 First part of the power subsystem of the Fox soaring platform. A single 3300mAh battery powers the motor and control surface servos. The power to the servos is passed through the Piccolo SL autopilot. Physically, the 3300mAh battery is connected to the switching regulator, but power is passed directly to the motor. A 1350mAh battery powers the Piccolo SL directly, which is protected by a 0.75A fuse. 40

3.9 Second part of the power subsystem of the Fox soaring platform. A single 2200mAh battery powers the Advantech PC/104 single board computer. Voltage is regulated from 12V to 5V through a switching regulator. The computer and regulator are protected by a 5A fuse on the battery side. 41

3.10 Batteries and autonomy subsystem installed into the Fox fuselage. Pictured here are two 3300mAh batteries, one for powering the motor and actuators, and the other for powering the on-board computer; and a 1350mAh battery for powering the Piccolo autopilot. In alternative configurations, the 1350mAh battery is mounted behind the center of mass. 41

3.11 The portable ground station. The ruggedized case contains a radio transceiver that is used to receive telemetry packets from the Piccolo autopilot and send autopilot loop control packets upstream. Telemetry information is monitored on the laptop computer, which connects via serial port to the ground station. The pilot console (pictured) connects to physically to the ground station via a serial link and allows a pilot to take over manual control of the aircraft, bypassing the autopilot. . . . 42

3.12 The logical architecture of the dynamic soaring platform software system. The system is divided into two main components: (left) the Cloud Cap Piccolo SL autopilot. This autopilot is regarded functionally as a black box. It generates telemetry packets at 25 Hz and accepts incoming commands. (right) the Advantech PC/104 On Board Computer consumes telemetry from the autopilot and issues commands to enable following of dynamic soaring trajectories. The architecture of this system is a loosely connected graph of nodes that pass messages. 43

3.13 The Fox platform during the wing load test. One-kilogram bags of pea gravel were placed on the wing to simulate a high load turn. The load test was performed for 5 minutes, and then the bags were removed, and the structure was inspected for damage. Preliminary testing revealed damage to the structure of the wings and wing spar. 49

3.14 An inside look at the wing joint inside of the Fox fuselage. The stock joint is two eye hooks connected by a spring. The wing spar runs left to right through the tube in the center of the image. Results of the test led us to reinforce this join with more robust eye hooks, and a plastic security tie 50

3.15 Results of the temperature test. A steady state internal temperature was achieved at 50C over roughly 30 minutes. This is within the operational range of the CPU and Piccolo SL at 80C. 52

3.16 The whip antenna design replaced an earlier design after signal strength issues were encountered. This antenna is approximately 2 m long, with .6 m of shielded coaxial cable, and 1.2m of exposed radiator. The antenna runs through the length of the fuselage, starting in the vertical stabilizer and terminating near the cabin for connection to the Piccolo SL autopilot. 53

3.17 The received signal strength test. (top) performed at 1.0W (bottom) performed at 0.1W. This test shows even at the lower power setting, the received signal strength never drops below the recommended -85 at 200m range. At the higher power setting, which is the target test setting, the RSS never drops below -65. 55

3.18 The acknowledgement ratio at 200m for 1.0 W (top) and 0.1W (bottom). This measure represents the percentage of packets sent from the ground station that are acknowledged by the autopilot as received. At full power, this ratio never drops below 94%, with a mean acknowledgement of 99.42%. At the lower power, this ratio never drops below 81%, with a mean acknowledgement of 94.18%. 56

3.19 The experimental setup of the vibration test. The fuselage was leveled under two compressible rolls and secured to a table using ratcheting ties. The motor was then run through its operational range and gyroscope data was recorded via the Cloud Cap Command Center logging facility. 58

3.20 Orientation results for the vibration test. The mean absolute deviation from the mean orientation across all axes stayed below 0.7 degrees. Since the airframe did not move during the test, a very small deviation was expected. 60

3.21 Gyroscope results for the vibration test. The three stages of the test are indicated by vertical lines. The first stage beings at ~20 seconds, where a spike in roll rate is observed as the motor is started. The second stage begins at ~60 seconds, when the RPM of the motor is increased. The final stage begins at ~100 seconds, when the motor is set to its highest setting. The motor is turned off at ~180 seconds. 60

3.22 Balloon horizontal velocity magnitude estimates vs. time for the vision system (solid black line) and GPS (dashed blue line). In this trial, the mean absolute deviation vs. time between the two approaches was 0.24 m/s. 63

3.23 Testing platform used in this work. During testing, the aircraft was flown manually while telemetry data were logged via the on-board Piccolo SL autopilot system. 64

3.24 Aircraft flight path (black solid line) and balloon tracks (dotted color lines) from flight testing. The latter were recovered by the vision tracking system 65

3.25 The Fox gliding platform landing in a snow-covered field after a flight test. This particular test was conducted under manual control. 66

4.1 An example wind profile. The parameters for this profile are w_{max} = 8m/s and Δz = 10m. The boundary layer is between 100 and 110 m altitude. Below 100 m the air is still (w = 0m/s) and above 110 m the wind blows at w_{max}. The aircraft gains kinetic energy as it transitions through the boundary layer. 74

4.2 The seed trajectory used to initialize the planner. This trajectory is just a banked orbit with a constant roll. Using a seed trajectory of this form speeds up the solution process significantly. 79

4.3 An example loitering dynamic soaring trajectory. The aircraft, drawn to scale, is visualized at several points around the trajectory. The starting point of the aircraft is marked in green. A wind gradient is blowing in the positive y direction at 8m/s at 110 m altitude, and 0 m/s at 100 m altitude. 79

4.4 Traces of each of the state variables of the aircraft over the course of the trajectory. The aircraft starts at 20 m/s airspeed and completes the loop at 23.8 m/s airspeed. The altitude of the aircraft at the start and end of the trajectory is the same, so this increase in airspeed is a net energy gain. 81

4.5 Two sets of similar dynamic soaring trajectories. (left) a set of trajectories that hold w_{max} and the height of the boundary layer constant, while varying the initial airspeed of the aircraft. (right) a set of trajectories that hold the initial airspeed of the aircraft and the height of the boundary layer constant, while varying w_{max}. The red trajectory in each of the figures has the same parameters of 15m/s initial airspeed and 8 m/s w_{max}. 82

4.6 A schematic of the local planning algorithm in action. Several trajectories are shown in black, with the lowest-cost trajectory highlighted in green, since the position is closest to the global plan, shown in blue. 85

4.7 Trajectory Rollout Algorithm 86

4.8 Point mass simulation over 20 DS cycles. Wind in this figure blows from the negative Y direction. 88

4.9 A side view of the path of the aircraft, looking toward the YZ plane. Wind is blowing from left to right in this figure. 88

4.10 A top view of the point mass simulation, looking at the XY plane. Wind is blowing from left to right in this figure. 89

4.11 True airspeed of the glider over 20 cycles. The maximum steady state airspeed reached is 34 m/s. The initial aircraft airspeed is 20 m/s. 89

4.12 Altitude of the glider over 20 simulated loops 90

4.13 Total energy of the glider, defined as kinetic energy plus potential energy due to gravity, over 20 simulated loops 90

4.14 Hardware in loop simulation results at a steady state. The blue line represents the target trajectory, the red line represents the flown steady state trajectory, and the green line shows a planner rollout. . . . 92

4.15 A side view of the hardware in loop simulation. The glider reaches the target maximum altitude and minimum altitude, but glides wide on the low-end turn. 92

4.16 A top view of the hardware in loop simulation. 93

4.17 Results for the concentric level orbit following. Blue lines represent the target trajectory, red line represent the flown path. The glider was commanded to follow 4 concentric orbits of varying radii from 40m to 100m. Transitions between orbits can be seen in the lower portion of the figure. 94

4.18 A horizontal view of the orbit following results, looking at the YZ plane. Deviation from the level orbits did not exceed 2 meters in either direction. 94

4.19 A top view of the orbit following results, looking at the XY plane. A constant offset from the target track is noted, due to the lookahead setting on the sample-based controller. 95

4.20 Results of the field test for tilted orbit following. 96

4.21 Side view of the tilted orbit test, looking at the YZ plane. The glider did not quite reach the maximum altitude on the highest tilted orbit. This was due to a lack of power in the motor, which was fixed in a subsequent platform revision. 96

4.22 Top view of the tilted orbit test, looking at the XY plane. 97

5.1 The training process for one DS cycle. The teaching controller performs 10 examples of following the DS trajectory. Then, the learning controller assume agency over the aircraft, initially experiencing a decrease in performance, but soon surpasses the teaching controller. The learner is then held constant and tested at various starting locations to follow the DS trajectory. 107

5.2 Ten training trajectories (red) and ten learned trajectories (black). The training trajectories aim to track the blue trajectory. The black trajectories aim to maximized energy gained after one cycle. 108

5.3 Soaring results for the HiL simulation. The target DS trajectory is depicted in blue, with the RL controller path in red, and the sample-based controller path in green. The RL Controller does not track the blue trajectory but creates similar path from learned experience. 110

5.4 Results from multiple learning loops shown above. The top line (blue) is the energy gain per cycle of a set of dynamic soaring trajectories generated by the global planner. The bottom line (red) is the energy gain per cycle of the sample-based controller following those trajectories. The middle line (orange) is the energy gain of the learned controller following those trajectories after training was finished. 111

5.5 Path of the learning controller followed over multiple DS cycles to a steady state. No global planning or wind mapping is necessary for this controller to function. A steady state is reached after about 4 cycles. Wind is blowing at w_{max} = 8 m/s from the negative y direction. The boundary layer exists between 100 and 110 m altitude. 112

5.6 A trace of the aircraft state over the course of several cycles. The initial airspeed of the glider was 15 m/s, and the maximum achieved airspeed was ~27 m/s. 113

5.7 A comparison of the learning controller (red) and the sample-based controller (green) under conditions of uncertainty in the wind field. The expected wind was w_{max} = 8m/s, but the actual wind was w_{max} = 10m/s. The sample-based controller crashed after only 3 cycles. The reinforcement learning controller soared until a steady state was reached 116

5.8 A side view of the robustness comparison. 117

5.9 A top view of the robustness comparison. 117

5.10 A trace of the aircraft altitude for the robustness comparison. The green line is the sample based controller, the red line is the reinforcement learning controller. The sample-based controller crashes after only 3 cycles. 118

5.11 A trace of the aircraft airspeed for the robustness comparison. The green line is the sample based controller, the red line is the reinforcement learning controller. This trace reveals that the aircraft was not able to gain enough airspeed on the third cycle and entered a stall state as it began the fourth loop. 118

5.12 A dynamic soaring trajectory is pictured as the blue dashed line. The green path was generated by the sample-based controller. It makes a positive bank angle to follow the trajectory. The red path was generated by the reinforcement learning controller. It flies straight to maximize energy gain. 120

5.13 A trace of the bank angle over the DS cycle. The green line is the sample based controller. The learning controller bank angle stays negative, which nets it an additional 50J of energy over the teaching controller. . 120

Abstract

Despite advancements in our understanding of flight in modern times, birds remain unmatched when it comes to maneuverability and energy efficiency in flight; in particular seabirds like the albatross are known to travel vast distances without stopping for food by performing an aerobatic maneuver called dynamic soaring. When the maneuver is executed in the presence of a wind field that varies in strength of direction, the albatross extracts kinetic energy from the field. In this dissertation, we present an end-to-end system designed to exploit wind as the albatross does. The system we designed consists of a gliding platform outfitted with sensors and computational hardware, an on-board software platform that enables autonomy, and a ground platform for monitoring mission performance and issuing commands.

We contribute the design of an airframe, the Fox, capable of performing dynamic soaring at low altitudes (~400m above sea level). We validate the airframe against expected stressors (vibration, coefficient of lift, temperature, and communication signal strength), and show in simulation it can complete a dynamic soaring orbit in wind shear that varies in maximum wind speed from 8 to 12 m/s. We show that this airframe can reach speeds exceeding 40 m/s while soaring.

We fit the airframe with a commercial off-the-shelf autopilot, as well as a custom on-board-computing (OBC) solution to provide the necessary facilities to enable

autonomy. The OBC generates dynamic soaring trajectories that fit a wind-field map that is built as the aircraft is deployed and controls the Fox to follow them by sending commands to the autopilot using a sample-based controller scheme. This process is monitored by human operators on the ground via a portable ground station that is linked to the Fox via a radio antenna. Field tests are presented that validate real-world controller performance against simulated results.

Finally, we present a learning controller that learns from and out-performs the sample-based controller in simulation. While not field tested, we believe a self-optimizing controller of this form is necessary to enable autonomy of a soaring aircraft subject to extended mission durations.

While dynamic soaring field tests were not pursued in this work, we hope this dissertation will be a blueprint for future researchers to finally achieve autonomous soaring.

Chapter 1

Introduction

In his search to endow humans with the power of flight, Leonardo da Vinci looked to birds in order to understand how they defied gravity. Unlike his contemporaries, da Vinci recognized that the key to understanding flight was understanding the medium in which the birds travelled, the air, just as the key to understanding how fish swim is understanding how water flows. While da Vinci is largely credited as one of the pioneers in the study of fluid dynamics, he also made several observations that until recently have been lost with time; he recognized that many birds are able to stay aloft and travel great distances without flapping their wings [1]. Leonardo realized if he could harness the same power the birds use to stay aloft without flapping, he could power his own flying machines, which suffered from a lack of adequate power source. In his Manuscript E (circa 1513), da Vinci made the sketch picture in Figure 1.1 of an unidentified land bird traveling outside of Rome, Italy. The horizontal lines in the drawing represent the wind blowing from right to left, and a dark solid line shows the trajectory of a bird through the wind. The pose of the bird is drawn at several key points along its flight path. From the drawing, a clear pattern of flight emerges in four phases:

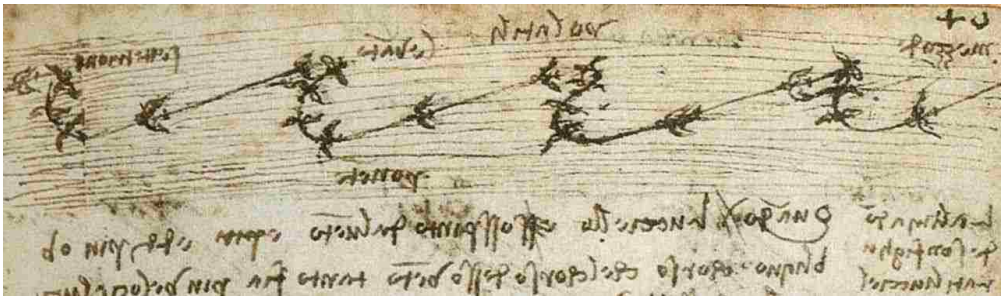


Figure 1.1 – A sketch of a bird performing a soaring maneuver in Leonardo da Vinci's Manuscript E (circa 1513)

1. The bird turns into the wind
2. The bird climbs in altitude
3. The bird turns sharply downwind
4. The dives and descends downwind. At a minimum altitude, the bird repeats the process

Leonardo da Vinci noted that this four-stage maneuver was continually repeated, which allowed the bird to travel over a distance without having to flap its wings. To him, it was obvious this form of flight was more energy efficient than when the bird is actively flapping, and so understanding this maneuver would finally enable him to achieve human-powered flight.

Unfortunately, da Vinci never did achieve his dream of flight, and it would be 300 years before his insight was re-discovered. In 1883, Nobel laureate and physicist John William Strutt, 3rd Baron Rayleigh was studying the flight of first pelicans [2] and then albatrosses [3], when he arrived at the same conclusion as da Vinci; their ability to stay aloft for long periods of time without flapping their wings was due to the cyclic climbing and diving maneuver. Strutt went a step further to postulate it was the differential in wind speed over the ocean surface that allowed for their longevity.

While neither da Vinci nor Strutt used the term, we now call this maneuver dynamic soaring.

1.1. Dynamic Soaring of Birds

Dynamic soaring (DS) is an aerobatic maneuver routinely performed by seabirds (and some land birds) like the albatross to extract energy (in the form of kinetic energy) from horizontal winds that vary in strength or direction. The gliding patterns of albatrosses have been particularly well-studied since Rayleigh first presented his ground-breaking work that explains the mechanics of dynamic soaring in detail. A soaring albatross is pictured in Figure 1.2. Cone Jr. modeled the albatross mathematically and showed with a set of equations that dynamic soaring provides them the capability to soar across oceans [4]. This model was simplified by Denny [5],

Figure 1.2 – A shy albatross performing a dynamic soaring maneuver. Its massive wingspan of up to 3.7m and weight of up to 5kg allow to harness the power of wind shear blowing over the ocean to gain kinetic energy without flapping its wings.

and verified experimentally with GPS trackers by Sachs *et al.* [6] [7]. Further investigations have identified the specific mechanisms by which albatrosses soar. Albatrosses themselves are structurally evolved to soar. Adams *et al.* showed that due to their large wingspan (up to 3.5 m, the largest of any living bird), wandering albatrosses have the lowest field metabolic rate recorded for free-ranging birds [8]. Pennycuick posited that the albatross' forward-pointing tubular nostrils serve as pitot tubes, while large nasal sensing organs monitor small variations in dynamic pressure, allowing them to detect wind gradients with great sensitivity [9]. Pennycuick also notes the albatross' wing is adapted to lock into a soaring position by way of a tendon sheath [10]. This enables the bird to hold a soaring configuration while expending very little energy; Sakamoto *et al.* compared the measured heart rate of a soaring versus resting black-browed albatross and found them to be comparable [11]. These adaptations make albatrosses well suited for environments like the ocean, with a constant supply of wind gradients that are strong enough to support dynamic soaring. Sachs found that a minimum wind speed of 5 m/s is necessary to support soaring in albatrosses [12], and Richardson found that wind shear soaring provides albatrosses with between 80% and 90% their total energy expenditure [13]. Even today, albatrosses continue to adapt to the oceans; Weimerskirch *et al.* found that increases in wind speeds due to global warming has resulted in larger albatrosses [14].

1.2. Applications of Dynamic Soaring

The unique and well-adapted nature of the albatross and other similarly evolved birds has lead researchers in aeronautics and robotics to consider how dynamic soaring can be used to extend the mission duration of unmanned aerial vehicles to last weeks,

months, or even perpetually. The idea of perpetual flight first found roots following the advent of electric engines and solar power. In 1974, Boucher proposed an aircraft that would climb to extremely high altitudes using solar power during the day, and glide during night to a reasonable altitude of 30,000 feet at dawn, essentially storing excess energy in the Earth's gravitational field [15]. With the advent of autonomous aircraft, researchers have looked into equipping UAVs with solar power as well. North *et al.* investigated using solar power for autonomous aircraft on Earth and on Mars [16]. Klesh and Kabamba considered energy-optimal path planning for solar powered aircraft [17]. One problem with solar-powered flight is that the design considerations are somewhat at odds; gliding aircraft aim to be as light as possible, while the addition of solar cells can add considerable weight. While solar technology remains in its infancy, they cannot generate enough power for flight on their own. Recent designs like the NASA Helios resemble a wing completely covered in solar cells, with little room left for payload. The Helios in particular met a grim fate when it disintegrated during flight, as a result of its sensitivity to atmospheric disturbances due to structural changes required for a long-duration flight demonstration [18].

By contrast, the aerodynamic factors that enable dynamic soaring actually synergize with aircraft design. The wandering albatross for example are not fragile birds; weighing anywhere from 5.9 to 12.7 kg, making them one of the heaviest flighted bird alive today. They use this heft to achieve a faster terminal velocity on the downward leg of the DS maneuver, allowing it to soar higher on the upward leg. What this means for soaring aircraft is that they don't have to be fragile in order to soar, making dynamic soaring a prime candidate for enabling perpetual flight. It has the added

benefit over solar that the maneuver can occur day or night as long as the appropriate wind conditions exist.

The migratory patterns of the albatross demonstrate that dynamic soaring conditions are prevalent close to the surface of the ocean, but Deittert *et al.* showed that the ability to fly very close to the water's surface is a key factor to dynamic soaring performance over water [19]. Indeed, as the albatross soars, it nearly skims its wing tips over the water's surface. Unfortunately, current and GPS and dynamic pressure sensors cannot position UAVs accurately enough to enable this kind of precise control. Fortunately, though, Sachs and da Costa showed that an alternative source of wind shear for dynamic soaring can be found in the shear wind regions near jet streams, which exist at extremely high altitudes of 10 – 20 km [20]. Grenestedt and Spletzer investigated soaring in this region and extracting energy to power turbines for electricity generation for avionics and control surfaces [21]. More recently, Xian-Zohn *et al.* showed that energy extracted during dynamic soaring at 10-20km altitude can compensate for between 30-50% of the energy consumed by drag on the downward leg, and between 20-40% on the upward leg [22].

The practical implications of this type of soaring are myriad. Dynamic soaring aircraft could use the jet stream as a kind of highway for global traversal. Alternatively, loiter-type dynamic soaring orbits could create a form of low-altitude satellite, which could monitor an area or act as a communications relay. We have even investigated the feasibility of using dynamic soaring aircraft in the eye of hurricanes for tracking and monitoring storms as they evolve [23].

1.3. Dissertation Objectives

In this paper we are interested in one specific formulation of the dynamic soaring problem – soaring in shear winds. Shear winds are the result of ground features which block prevailing winds, thus creating a wind gradient. Above the ridge, wind blows at a constant speed w_{max}, while below the ridge the air is still. Thus, a wind gradient exists in this region, which we parameterize by a height $\Delta z = z_f - z_0$. We are

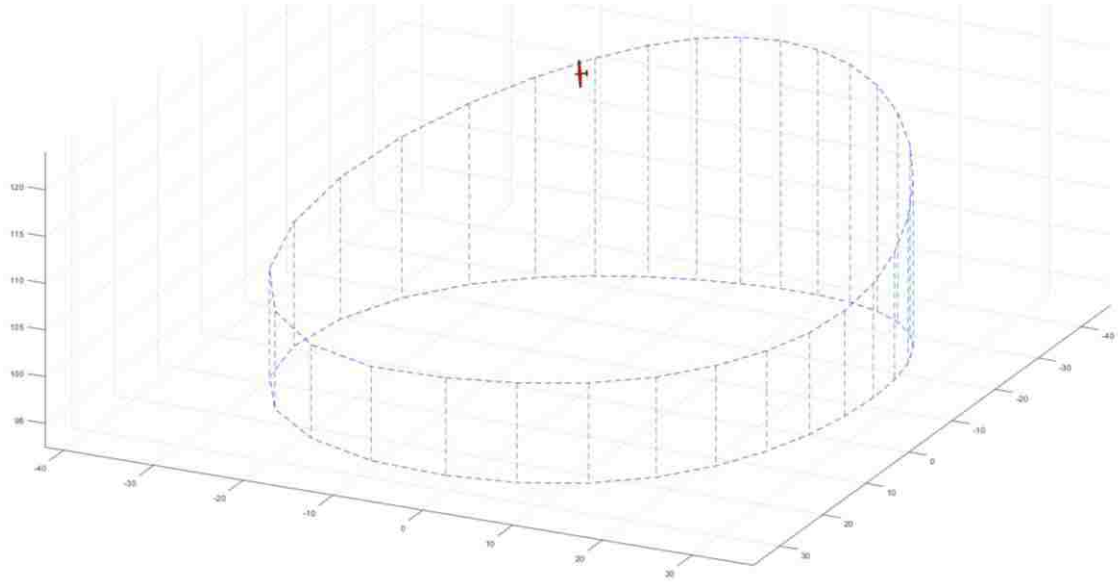


Figure 1.3 – A dynamic soaring loiter maneuver. A glider with 2.8m wingspan is pictured to scale in red. There is an implicit shear wind in this figure blowing from the back wall of the picture. Below 100 m altitude the wind is 0, and it increased to a value of w_{max} over a height of Δz.

interested in exploring loiter-type trajectories, an example of which is pictured in Figure 1.3, where the soaring aircraft returns to its starting position and orientation (but at a higher airspeed) after it executes one orbit of a DS maneuver, ready for a second cycle.

In this work we generate dynamic soaring trajectories offline, which represent a locally optimal solution to a nonlinear optimization problem. We use an iterative control strategy to follow these trajectories, the idea being that if the aircraft can track the trajectory well enough it will realize a net kinetic energy gain. However, wind gusts, tracking errors, and inaccurate sensor measurements can be devastating for an energy maximizing aircraft – too many inaccuracies will destroy the energy margin in the trajectory. Therefore, a control that properly accounts for these errors stands a greater chance to successfully completing a DS orbit. To this end, we explore an alternative control strategy based on reinforcement learning (RL), a machine learning paradigm which aims to learn a controller based on observed actions. Wharington specifically approaches several soaring problems, including thermal soaring, essing, and dolphin soaring using a reinforcement learning controller [24]. However, he stops short of RL for dynamic soaring, noting that the problem is probably of too high dimensionality for reinforcement learning techniques.

In this dissertation, we present all the components necessary in building an autonomous controller for a dynamic soaring task. First, we specify a global planner, which generates local optimal dynamic soaring trajectories. Then, we outline a local planner, which generates commands for the aircraft to execute. Finally, we detail the learning controller, which uses the local planner to gain experience about the dynamic soaring task.

1.4. Dissertation Contributions

In this work, we contribute several results to the field of robotics and aeronautics:

With respect to the gliding platform:

- Experimental results demonstrating the ability of this platform to withstand forces experienced during dynamic soaring.
- Experimental results verifying the autonomy platform is capable of controlling the glider for certain dynamic-soaring like orbits.

With respect to the software platform

- Experimental results demonstrating this system working in three different domains: a particle-based simulation, a Hi-fidelity simulator, and in real-world flight testing.

With respect to the reinforcement learning controller

- Experimental results that demonstrate the learning controller outperforms the teaching controller.
- Experimental results that demonstrate transfer of the learned controller from one domain (point-mass simulation) to another domain (hi-fidelity simulation that simulates aircraft dynamics)

1.5. Dissertation Outline

This dissertation is organized into 5 subsequent chapters:

Chapter 2 covers the necessary background in reinforcement learning, which is a necessary precursor to the work done in Chapter 5 on the reinforcement learning soaring controller

Chapter 3 details the airframe that is the target platform, the Fox, for the algorithms and methods described herein. We show each of the modifications made to the stock airframe to make it robust enough for a soaring task. We also detail the autonomy-enabling avionics added to the platform, and the software architecture that it supports. Finally, we show the results of several stress tests that validate the Fox as ready for dynamic soaring.

Chapter 4 covers the algorithms used to enable autonomous dynamic soaring in the Fox. We present a "global planner" that generates dynamic soaring trajectories for the Fox to follow, and a "local planner" that generates commands for the autopilot to execute the maneuver. We show simulated dynamic soaring results, as well as real world field tests on simpler elliptical trajectories.

Chapter 5 covers a learned approach to soaring. We present a two-stage learning process. First, a teaching controller (as presented in Chapter 4) is used to demonstrate correct soaring maneuvers to a learning controller. The learner observes the correct behavior and constructs a model that is used later to perform its own soaring task. This learned controller is not only computationally faster than the teaching controller, it outperforms the teaching controller in terms of energy gained per cycle.

We close the dissertation in Chapter 6 with a comparison of the two controllers, and a discussion on the steps required to field test the presented platform. We also provide several avenues for future research into the learned controller, as well as the more traditional sample-based controller.

Chapter 2

Background

The content presented in this section aims to map out the design space for reinforcement learning algorithms as applied to mobile robotics. We start by looking at reinforcement learning in general as method for problem-solving and examine each component of the learning framework. We then describe the various approaches to solving learning problems and pay special attention to the Q-Learning algorithm. Finally, we look at the challenges in implementing this algorithm in a mobile robot context. This analysis serves as a precursor for the presentation of reinforcement learning for dynamic soaring as presented in Chapter 5.

2.1. Reinforcement Learning Framework

According to Sutton and Barto, "Reinforcement learning is defined not by characterizing learning methods, but by characterizing a learning problem" [25]. Therefore, in this section we introduce reinforcement learning as a problem framework and define the various components of this problem.

The essential reinforcement learning model consists of the following elements [26]:

1. A set of environment states S
2. A set of actions A an agent can make
3. A set of scalar reinforcement signals (rewards)

Beyond these key components, most RL techniques share the following ideas

- **Optimality** – at its core, RL is an optimal control technique, so some notion of optimality needs to be defined. This can vary depending on the structure of a particular problem, and may take the following forms
  - **Cost to go** – the robot expects to act for a certain number of steps and receives a reward at each step.
  - **Receding horizon** – the robot acts infinitely and receives a reward for a certain fixed number of steps (the horizon). Thus, the agent is only interested in its actions over this horizon.
  - **Discounted infinite horizon** – the robot acts infinitely and receives a reward for its actions that stretch infinitely into the future, with actions closer in time receiving more weight and actions at infinite time receiving no weight (known as reward discounting). Most RL algorithms are based off this reward structure.
- **Exploration vs. Exploitation** – As we will see in the next section, once a robot learns how to act, it follows what is known as a *policy*. This policy tells the robot which actions to take given its current state to receive the best reward. This mode of operation is known as *exploitation*. However, it assumes that the policy the robot is following is globally optimal. But if the robot is following a merely locally

optimal policy it could miss out on even greater rewards. Thus, *exploration*, or taking sub-optimal actions in the hope of finding a globally optimal policy, is a key component of reinforcement learning.

- **Value Functions** – While the main objective of RL is to execute a policy, this policy is often difficult to derive directly. Instead, many RL algorithms keep track of what is called a *value function*, which indirectly encodes the policy. Value functions can be discrete or continuous and can be generalized from training to new scenarios.
- **Models** – While RL promises to be a model free framework for learning, as we shall see in the next section a model of the environment or agent is sometimes helpful or even required. Models can either be used to enable tractability, or even bootstrap the policy so the learning process can focus on important areas of the state space.
- **Observation** – After an RL agent acts on the environment, it must then observe the environment to ascertain the post-action state. This observation can come from sensors in the agent and can either fully or partially specify the state (i.e. the observation is noisy). If the state is partially specified, the agent must make some inference on what the true state is using by tracking the uncertainty in its state.

2.2. Approaches to Reinforcement Learning

We now survey several of the most fundamental approaches to reinforcement learning. These include dynamic programming methods, Monte Carlo methods, and temporal difference learning methods.

Dynamic Programming

Dynamic Programming was introduced by Bellman in 1957 to solve time-varying multi-stage decision processes [27]. In the context of reinforcement learning, dynamic programming is used to solve Markov Decision Processes (MDPs) meaning that the algorithm needs a full model of the MDP including all states, actions, and transition probabilities. Therefore, dynamic programming is a "model-based" approach to RL.

Unfortunately for many RL domains including mobile robotics, a full description of the MDP may not be readily available or impossible to construct. Thus, while dynamic programming methods are certainly used, they mostly serve as inspiration for the other methods in this section, which are more common in mobile robotics applications.

Monte Carlo Methods

These methods estimate the value function via a sampling procedure and have an advantage over dynamic programming methods in that they do not require a model of the environment. They work by taking the current policy and executing a sample of rollouts, and then observing the cumulative reward at the end of the rollout. The frequency of occurrence of the next states is recorded and used to estimate the value function [28].

Monte Carlo methods can also be "off-policy" – that is, they do not use the current policy to control the agent while simultaneously estimating it. In these methods, a second policy called the behavior policy is used to control the agent, while the true policy is estimated and improved [25].

While these methods have the noted advantage over dynamic programming methods, they suffer in that computational complexity can range from polynomial to exponential in the number of states, depending on the problem structure.

Temporal Difference Learning

Temporal Difference (TD) learning is very similar to Monte Carlo methods in that they are model free and sample the environment to test rewards. However, instead of looking at rewards for a full rollout, they look at rewards over a single time step. By calculating the error between the next state reward and the current state (the temporal difference) TD methods can update their estimate of the value function at each time step. We now look at an example of one of the most popular TD learning algorithms, Q-Learning, to gain an insight on how TD algorithms work.

Q-Learning

Introduced in 1989 by Watkins [29], Q-Learning is a TD algorithm where the learned action-value function Q approximates the optimal Q^* independent of the policy being followed (thus it is an "off-policy" algorithm). The introduction of Q-Learning was particularly important to reinforcement learning due to the convergence proof; Watkins showed that as long as all state-action pairs continue to be updated, the algorithm will converge to the optimal state-action function Q^* with probability 1 [25]. Thus Q-Learning is one of the few model-free RL algorithms with performance guarantees.

The basic algorithm proceeds as follows: the agent operates over a series of episodes, during which it transitions from state to state according to some policy (more on how

this policy is chosen later) in an attempt to reach a goal. At the start of the algorithm, the Q matrix is initialized arbitrarily. It can be random, filled with zeroes, or even bootstrapped from a previous run of the algorithm. The algorithm then begins iterating through episodes. At the beginning of each episode, the state of the agent is initialized randomly, which is important to maintaining the convergence property of the algorithm. The agent then chooses a next state based on a policy, which can be derived from Q_t (i.e. a greedy approach) but can also simply be random. If a greedy approach is taken, it is important to take random actions with some frequency in order to continue to visit all states.

On each action, the agent observes the reward in its new state, and updates Q_t according to the update rule (Equation 2.1). The agent continues taking actions and updating Q_t until it enters a predetermined goal state, at which point the episode is terminated and a new one begins. Episodes continue in this way until Q_t converges to some accuracy threshold.

There are many variations to Q-Learning, but the simplest form is the one-step update, using the following update rule:

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q_t(s_t, a_t)] \qquad \text{Equation 2.1}$$

Here, $Q_t(s_t, a_t)$ is the old value of the state-action function at the current state and the chosen action, α_t is the learning rate at the current time, R_{t+1} is the reward earned at the new state, γ is the future action discount factor, and $\max_a Q(s_{t+1}, a)$ is the best

state-action value over all actions at the next state. Pseudo-code for the Q-Learning algorithm is presented in Figure 2.1.

One interesting feature of Q-Learning is that the policy the agent follows can change over time, with more random actions taking place initially, and then switching to a greedy approach to take advantage of the information in Q_t.

Algorithm: Q - Learning

Initialize $Q(s, a)$ with arbitrary values

Choose discount factor $0 \leq \gamma \leq 1$

Choose learning rate schedule $0 \leq \alpha_t \leq 1 \; \forall t$

While *Q* has not converged

    Initialize *s*

    While *s* is not terminal

        Take action a_t

        Observe reward R_{t+1}

        Observe next state s_{t+1}

        Update *Q* :

        $Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q_t(s_t, a_t)]$

    End

End

Figure 2.1 – Q Learning algorithm pseudocode

While Q-Learning is quite popular in the RL community, it is not without its shortcomings. Foremost among these are the learning rate α, and the discount factor γ, which have a direct impact on the rate of convergence of Q_t. Currently, there is no well-defined method on how to choose these parameters to ensure an optimal convergence time. Choosing a learning rate of $\alpha = 1$ is optimal for a fully deterministic

environment, but many problems do not have this property. In the stochastic case, the problem has been proven to converge as long as α is decreased over time to zero [25].

2.3. Challenges for Mobile Robots

In this section we take a take a look at reinforcement learning from the perspective of mobile robotics research and explore some of the implications of the RL framework and algorithms presented in the previous section.

The Curse of Dimensionality

When Richard Bellman published his work on Dynamic Programming, he coined the phrase the "Curse of Dimensionality" [27], which is often used today as a way to express the dependence of machine learning performance on the availability of vast quantities of data. In robotics and RL in particular, we take this to mean that as the state and action space grows linearly, exponentially more data is needed to provide sample coverage of the entire space. For robotics applications, this is a tremendous problem. Mobile robots work in continuous spaces, often in many dimensions. Consider the state of a robot aircraft, which has six degrees of freedom in position and orientation (x, y, z, roll, pitch, yaw) each of which is continuous. Important in many applications are the dynamics of the aircraft as well, including its angular rates, its momentum, and its accelerations in all directions. Then consider its environment, which may have wind with components in all directions, each of which is continuous. We might also need to consider the actuators on the aircraft, of which there may be anywhere from one to a dozen.

Today, these problems are made tractable using hierarchical control systems [28]. For example, looking back to the robot aircraft example, we might be able to reduce the dimensionality of the state space by control angular rates of the airframe, delegating an autopilot to manage control surfaces. The key here is to reduce dimensionality of the space by clever representations or offloading them to subsystems. Unfortunately, there is no automatic way to perform this analysis.

Sampling Cost

Many RL algorithms contain a line of code that says to the effect "initialize *s* randomly". In particular, the convergence proof of Q-Learning requires that all states be visited continuously. Unfortunately, constantly exploring states or initializing using random states for a real-world system like a mobile robot is not a realistic task. In real environments, some states may not be accessible to the robot, or they could put it in a dangerous or unstable situation. Further, the environment itself may not present a certain state to the robot. Consider the robot aircraft again for a moment. If the wind is a component of the problem state, the robot may only get to experience fleeting wind conditions, and not at every desirable position. Furthermore, as the robot acts in the environment its performance changes – tire pressure changes with temperature, lift changes with altitude and air density, actuators wear over time.

To deal with some of the cost of generating samples, simulation is possible [28]. However, this requires a model of the robot's kinematics and dynamics (which reinforcement learning attempts to render moot) and does not guarantee the learned simulated model will generalize to the actual system [30]. Further, controlling the robot manually while allowing the learning process to observe the state and receive

rewards maintains safety and has proven to be an effective way to bootstrap the learning process under autonomous control [31].

Model Uncertainty

Robots are inherently uncertain machines, and mobile robots in particular must deal with their own inherent uncertainty as well as that of the environment. In the MDP literature, uncertainty is modeled using a Partially Observable MDP (POMDP). Typically, POMDPs track the current (assumed) state as well as a probability distribution that tracks the amount of uncertainty in that state observation [32].

Model uncertainty can also arise when trying to characterize the environment. Referring to the robot airplane example again, the wind is very difficult to model. It varies seasonally, by time of day, and even minute to minute with gusts that can potentially ruin learning performance.

RL is an attempt to remove modeling from the design of robot control, so it may seem counter-intuitive that modeling is detrimental to RL if we ultimately want to eliminate it. Nonetheless, some algorithms require a model before they can work, and even some basic modeling can solve issues of tractability.

2.4. Tractability

In this section, we provide some techniques robotics researchers use to apply RL algorithms to real world systems. These techniques address some of the problems outlined in the previous sections, including the curse of dimensionality and tractability concerns.

Discretization

Continuous spaces can be made tractable by bounding a region and discretizing it into a small number of cells. In this way, we can artificially limit the number of states in the system. This discretization can either be done at fixed intervals or change dynamically based on regions of interest. For instance, for a mobile robot navigation task, large open areas can have a coarse representation, while detailed areas (like the façade of a building) might have a finer grained discretization.

Meta-Actions

Meta-Actions are high level-commands given to a robot which are executed by a low-level controller. An example of a meta-action for a mobile robot navigation task would be to enter a designated room in a building. A traditional controller can control the low-level operations of the robot, such as obstacle avoidance, localization, local motion planning, etc. The robot is then free to explore its environment and try to learn the relationships between rooms in the building. The use of this underlying controller can significantly reduce the dimensionality of the problem. Even a grid discretization of an entire building results in an intractable number of states. However, a building with only a dozen or so rooms can be modeled compactly as an MDP.

Value Function Approximation

Value function approximation addresses issues of both scalability and generalizability. Consider a robot that operates in a discretized continuous space. If we discretize the space with different levels of coarseness and run an algorithm such as Q-Learning in each case, we will arrive with two different Q matrices, each with a

different number of elements but that resemble each other. This goes to illustrate that the information learned in a coarse representation can be generalized to a fine representation. There are various methods for doing this. One such method is a nearest neighbor approach, where an agent in a continuous space simply references the nearest or k-nearest state in a Q matrix.

Another often used technique is function approximation, where the Q matrix is converted from a discrete table to a continuous function. This can be accomplished using neural nets [33], where the training inputs are the discretized states and the outputs are the learned Q matrix values.

Finding Interesting Regions of the State Space

As part of the curse of dimensionality, the state space can be so large that interesting regions can be hard to find by an agent taking arbitrary actions. Depending on how the reward is structured, a robot may only receive a reward when it reaches a goal state, and thus in a large space will learn nothing until it happens to stumble upon that state.

For a mobile robot, this is particularly problematic since run time is limited, and continuous operation can have a wearing effect on motors and actuators. Therefore, directing the robot to search in a particular region of the state space can significantly reduce learning time.

One method for directing the robot to interesting regions of the space is by using intermediate rewards in a process known as “reward shaping”. Instead of a binary reward for reaching the goal, a robot may receive a partial reward for moving closer

to the goal, and the robot can perform a kind of gradient descent to arrive at a goal state [28].

Other methods include teaching the robot through demonstration [31], bootstrapping the policy with task-specific knowledge [28], or bootstrapping the value function with a previously learned value function.

Modeling

Despite being excluded by the general RL framework, modeling is important for enabling tractability of many RL algorithms in robotics. One of the most important functions of modeling is to help generate samples used to train algorithms before running them on actual hardware. The robot can even use a forward model of its kinematics to generate predicted states after taking a variety of actions before selecting one to execute.

One prominent use of modeling is to use a forward model to estimate gradients for update steps. Once the estimate is made, the policy is updated, and the action is carried out and evaluated to provide an update to the model [28].

2.5. Recent Trends in Robot Control

In 1991, Rodney Brooks wrote about two distinct engineering paradigms crystalizing in academic robotics research at the time [34]. He calls the first paradigm the "traditional" approach, in which robots are designed to demonstrate a theoretical system working. In this approach, delicate symbolic models of the world are created of a robot, allowing it to work under laboratory conditions, but nowhere else. Brooks

contrasts this with a "new" approach to robotics research, which involves tightly coupling sensing and actuation through a network of shallow yet broad computational units. This approach yields robots that can operate in a variety of conditions – instead of planning using a symbolic model, these new robots instead react to environmental stimuli directly. Loosely coupled components, each operating in this way, work in concert to produce emergent behaviors.

What Brooks described in 1991 seems to be one of the earliest applications of a neural networks to the problem of mobile robotics. Since then, we've largely dealt with uncertainty in robotics through probabilistic representations, *e.g.* modeling a robot pose with a probability density function of some mean μ and standard deviation σ [35]. For a long time, these approaches were more practical for dealing with uncertainty than learned approaches, due to the long times associated with training models and classifying data. Probabilistic approaches, by contrast, could run on commodity hardware that could fit in the trunk of an autonomous car.

By 2010, computing hardware, especially cheap GPU computing hardware, had reached the point where learned methods could compete with the probabilistic methods used prevalently. Learned controllers for helicopters, humanoids, and autonomous cars emerged taking advantage of these new resources [28]. Furthermore, wide availability of cheap, online storage gave birth to a renaissance of data analytics. The combination of copious amounts of data and the availability of performant hardware with which to process brought new life to the idea of convolutional neural networks and their application in "deep learning".

Deep Learning has demonstrated impressive results, starting first with advancements in object detection and recognition [36]. But now, we're starting to see more impressive results in control – from a reinforcement learning controller for classic video games [37], to a planner for the game Go that outperforms any other computer or human [38]. Still, for all its success, deep learning techniques rely fundamentally on a vast reservoir of data from which to learn, and a long time-frame over which to learn it. The Google AlphaGo controller, for instance, took months of real training time to achieve super-human performance [39].

Even in today's world of big data, some domains suffer from a dearth of viable data with which to train deep learning models. These domains are ones in which data collection is too risky, or too costly. In such domains, it would be advantageous to train adequate models with safety guarantees (these domains are dangerous after all) using as little training data as possible.

Chapter 3

Fox Soaring Platform

In this section, we outline the components of the end-to-end soaring platform that is the major contribution of this dissertation. The platform consists of 4 major components: a heavily modified off-the-shelf gliding airframe called the Fox, a custom chassis for mounting autonomy-enabling hardware, a ground station command center for monitoring and controlling flight missions, and a software platform for controlling the Fox platform as it soars. These four components work together to enable the Fox to perform our objective task of a dynamic soaring loiter pattern.

3.1. Related Work

Research into unmanned and manned dynamic soaring vehicles has been going arguably since Leonardo da Vinci hoped to harness the power of soaring for his flying machines in the 1500s, but interest has recently taken off as the effectiveness of dynamic soaring has been demonstrated in small-scale radio-controlled aircraft. In 2002, Boslough launched the first instrumented experimental flight test to collect data for dynamic soaring, which addressed the feasibility of developing a dynamic soaring flight controller [40]. Then, in 2006 Gordon investigated the potential for manned

dynamic soaring in a full-sized glider [41]. In 2010 dynamic soaring became somewhat of a hobbyist fad when Richardson demonstrated a radio-controlled glider traveling 487 mph while dynamic soaring off slopes at Weldon California [42]. Since 2011, the Lehigh Composites Laboratory has been manufacturing a carbon fiber airframe with a 6m wingspan intended to perform dynamic soaring maneuvers in the jet stream [43].

3.2. Chapter Contributions

The small-scale dynamic soaring airframes demonstrated in the literature are primarily remote controlled, meaning they lack the necessary equipment like computing hardware and autopilots which enable autonomy. The primary contribution of this work is the addition of the necessary computational hardware, and the resulting design considerations that follow from this addition. With the added computational resources comes the need for more battery power, and of course added weight. While additional weight is beneficial for a dynamic soaring application, it leads to additional complications in the structural design of the aircraft, which are systematically addressed in this chapter through the presentation of several stress tests. A second contribution of this chapter is the experimental verification of this platform in the field, which demonstrates the efficacy of the soaring controller and pushes the limits of the platform as far as we could short of actually soaring. Field test results are presented at the end of this Chapter and at the end of Chapter 4. The end result is a platform that has been field tested and modified over several iterations to arrive at a platform that is ready for dynamic soaring at low altitudes.

3.3. Design Considerations

Before we go into detail on each of the four platform components, we will first take some time to discuss the design parameters that informed the resulting system. Our primary objective is the creation of a platform capable of performing a dynamic soaring loiter maneuver in a shear wind at low altitudes (~400 m above sea level). This means the airframe needs to have a wide enough wingspan to support the maneuver at target windspeeds of 8 – 12 m/s, and the wings need to be strong enough to sustain the large coefficient of lift endured during the dive phase of the dynamic soaring maneuver. While many soaring platforms aim to be as light as possible, having a vehicle with a hefty mass is actually beneficial to our application. Some parameters that support dynamic soaring include [43]:

- High lift to drag ratio (implies a large wing area)
- Ability to perform high-g turns
- Energy extraction increases with speed
- High weight in order to increase terminal velocity on the downward phase of the maneuver
- Low parasitic drag to reduce energy loss

On a practical level, we expect to endure several crashes during field tests, so the airframes need to be easy to repair and cheap to purchase. While the airframe will be cheap, we need to protect the sensitive avionics and computer systems inside of the airframe in the event of a crash. Furthermore, the regulatory environment we expect to encounter for testing is uncertain. The components selected should have safety

features built in, such as geo-fencing, a dead-man switch on the pilot console, and the ability to monitor telemetry from the ground and abort dangerous missions.

Figure 3.1 – Fox dynamic soaring platform balanced on a center of mass calibration device. The Fox has a wingspan of 2.8 m and an empty mass of 2.90 kg.

3.4. Airframe

In this section we detail the airframe used as a development platform throughout this dissertation. The parameters laid out here are used as the basis for trajectory generation and motion planning as discussed in Chapter 4, as well as the reinforcement learning controller in Chapter 5.

Development Airframe

Our development airframe for low-altitude soaring is pictured in Figure 3.2. This airframe is a heavily modified EScale Fox commercial off-the-shelf radio-controlled sailplane.

It features a large fuselage capable of housing the autonomy-enabling computer hardware, as well as batteries for powering the avionics and aircraft motor. The aircraft has a wingspan of 2.8 meters, a wing area of 0.634 m^2, a fuselage length of 1.359 m, and an empty mass of 2.90 kg.

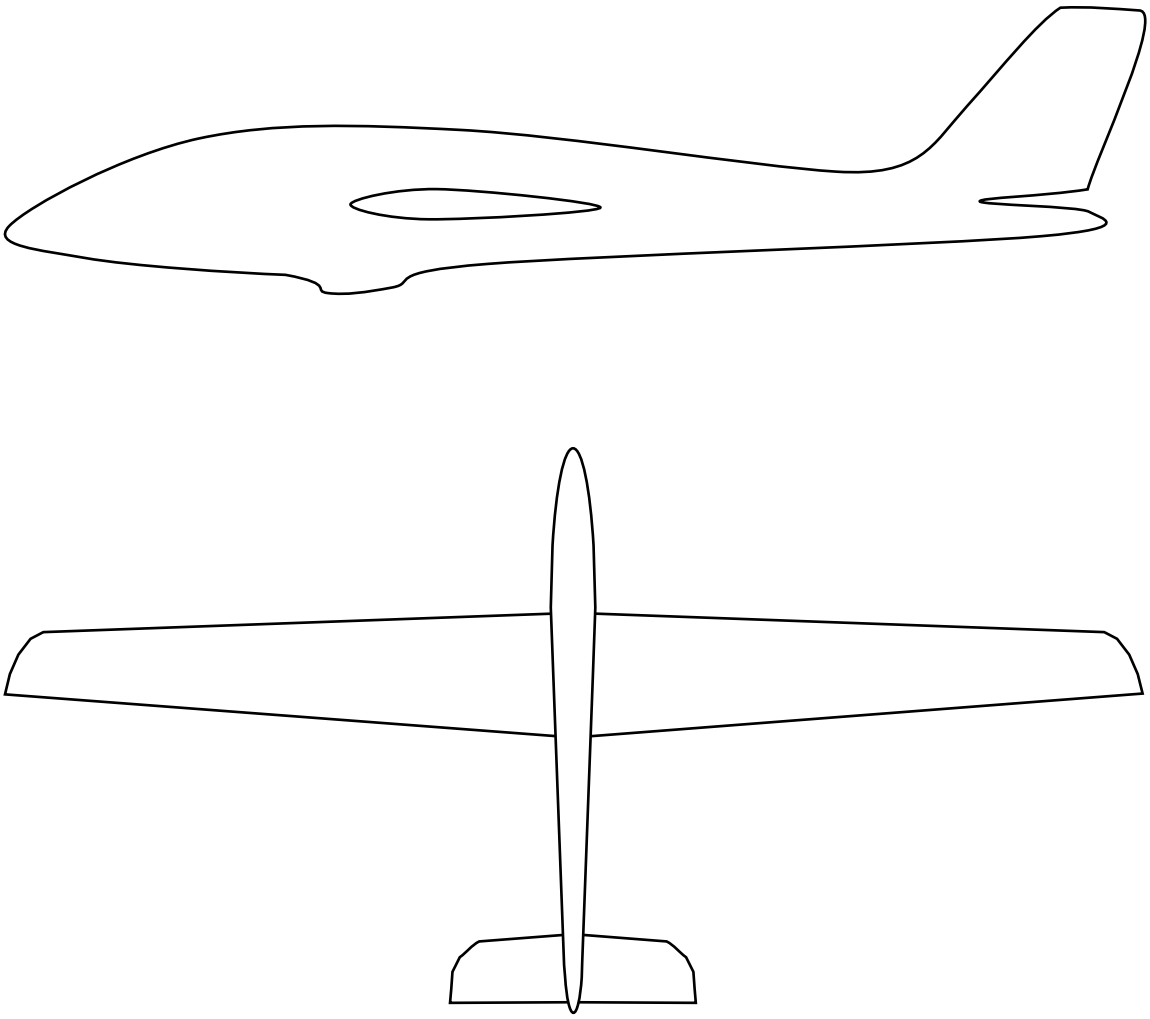


Figure 3.2 – A schematic of the Fox soaring airframe. It features ailerons on each wing, an elevator, and a rudder as control surfaces. A pitot tube is mounted on the right wing. Static pressure sensors are installed behind the wings on either side. A single brushless motor at the nose of the glider powers the aircraft for takeoff and landing but is not operational during dynamic soaring maneuvers.

Each wing is constructed of foam and balsa wood and is equipped with a long servo-powered aileron. The wings also have optional mountings for air brakes, which were installed for testing, but would be left out in a dynamic soaring scenario to reduce drag. A quarter inch thick steel rod acts as wing spar, joining the wings to the fuselage. They are secured together via a spring, and we add additional security with a plastic security tie.

The rudder and elevator are deflected by push rods, powered by servos located near the center of mass. The push rods are constructed of a bendable nylon and are very flexible. Due to the high forces exerted on the rudder and elevator during a soaring maneuver, we reinforced the push rods with aluminum tubing that runs from the servo and extends to their respective control surface. The push rod runs through the tubing, which prevents any flexing that would occur due to dynamic soaring forces.

3.5. Hardware Architecture

Autopilot

The Cloud Cap Piccolo SL autopilot [44] is the centerpiece of the Fox soaring platform, providing a variety of telemetry data that enables our autonomous motion planning. Through GPS it provides 3D pose information; through an inertial measurement unit it provides 3D heading information, as well as rates on all axes; through a pitot tube system it provides IAS. It also powers and controls the ailerons, elevator, and rudder. Most importantly though, it provides a link between the Fox platform and a ground station, which enables commands to be sent from the ground to the Fox as it is in mid-flight though the Cloud Cap Piccolo Command Center software suite.

Figure 3.3 – A custom steel chassis for avionics. It is installable in the fuselage of the glide. Pictured here is the Piccolo SL autopilot mounted on rubber standoffs to isolate it from vibrations. The two plastic tubes connect to the static and dynamic portions of the pitot sensing system. The antenna pictured here was a preliminary design that was abandoned in favor of a whip antenna that extends along the fuselage into the vertical stabilizer.

The Piccolo is pictured here mounted on a custom steel chassis that mounts within the Fox fuselage. The chassis is secured via three screws (one at the back, one in the middle, and one in the font) to wooden blocks glued to the inside of the fuselage. The Piccolo itself is secured to the steel bar via four rubber mounts to isolate the autopilot from vibrations. The efficacy of this mounting solution is explored in Section 4.8.4.

The antenna pictured here was a preliminary design that was replaced with a longer whip antenna (pictured in Section 4.8.3) when signal strength issues presented during flight tests.

Control Surface Calibration

Each control service needed to be calibrated for the Piccolo autopilot to properly control it. The process began with securing the Fox airframe in a level position on a table, and with all control surfaces set at zero deflection. In this case, the position was determined such that the pitch of the frame as reported by the autopilot IMS was zero with respect to the yaw. The calibration process for the ailerons and elevator was as follows: the surface was set to zero deflection and the orientation of the surface was measured with an electronic level. This recording was used a tare for the level measure. Then a pulse width signal was sent to the control surface via the Cloud Cap software. This resulted in a deflection of the surface, which was measured with the electronic level and recorded. This process was repeated several times to cover the entire operational range of the servo. The values were inputted into Cloud Cap, which performs a fit on the data points to arrive at a calibration for the control surface.

The process for the rudder was similar, but since it is a vertical oriented surface the electronic level could not be used. Instead, we used a protractor resting on the horizontal stabilizer, and noted the deflection of the rudder.

Pitot Tube Mounting

The pitot tube provides the Piccolo autopilot with an IAS measurement. This sensor consists of two parts. The first part of the pitot system registers dynamic pressure. It is a small steel tube mounted on the wing on the Fox. We originally used an aluminum tube, but this proved to be fragile during hard landings. The steel tube is embedded

in the wing facing toward the nose of the plane. Within the wing, a rubber tube runs toward the fuselage along the same track as the aileron servo control line.

The second part of the pitot system detects static pressure. These are two small rivets installed behind the wing on either side of the fuselage. They each are connected to plastic tubing inside of the fuselage and are joined together by a T connector that feeds into the Piccolo. Together, these work to report the airspeed of the Fox as it flies. Combined with the GPS, which provides ground speed, the Piccolo is able to determine the wind speed as well.

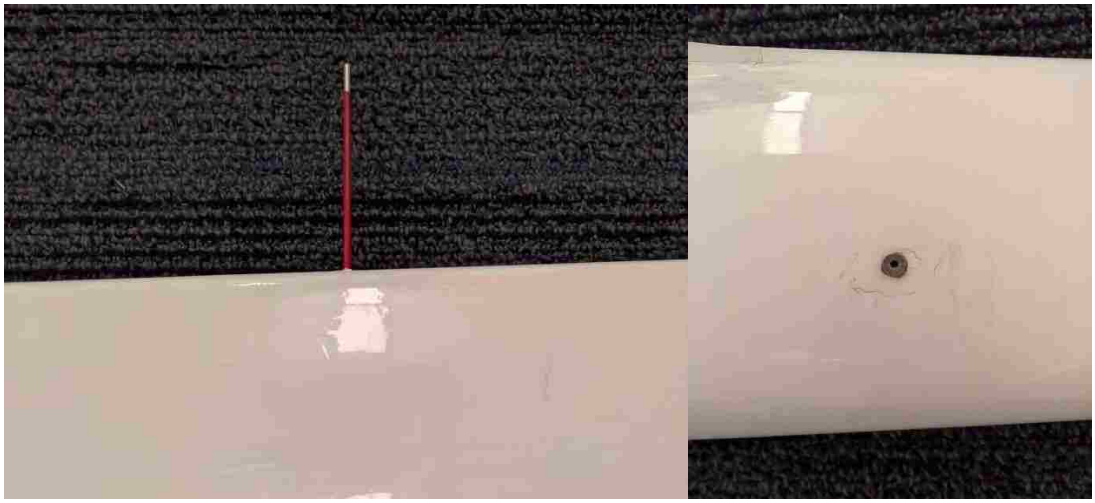


Figure 3.4 – The pitot sensing system. (right) A steel tube is mounted on the wing and painted red for contrast against the ground. This steel tube is connected to a rubber tube that extends through the wing into the fuselage. (left) static pressure sensors are mounted behind the wings on either side of the fuselage.

Autonomy Subsystem

While the Piccolo autopilot is capable of providing autonomy out of the box, it does not accept custom controllers that would enable a control scheme capable of dynamic soaring. However, it does accept control packets from a secondary

source in the form of autopilot control loop packets, which override the stock autopilot controller with roll and pitch angle targets. These packets can be provided from the ground station, but for a highly dynamic task like soaring, the latency between the ground and air is too much to overcome. Therefore, we equipped the Fox soaring platform with an on-board computer (OBC). The OBC is an Advantech PCM-3363 PC/104 single board computer with a dual core Intel Atom 32-bit processor. It runs the Linux operating system, and the MATLAB suite of software is installed as the programming language used for all control tasks.

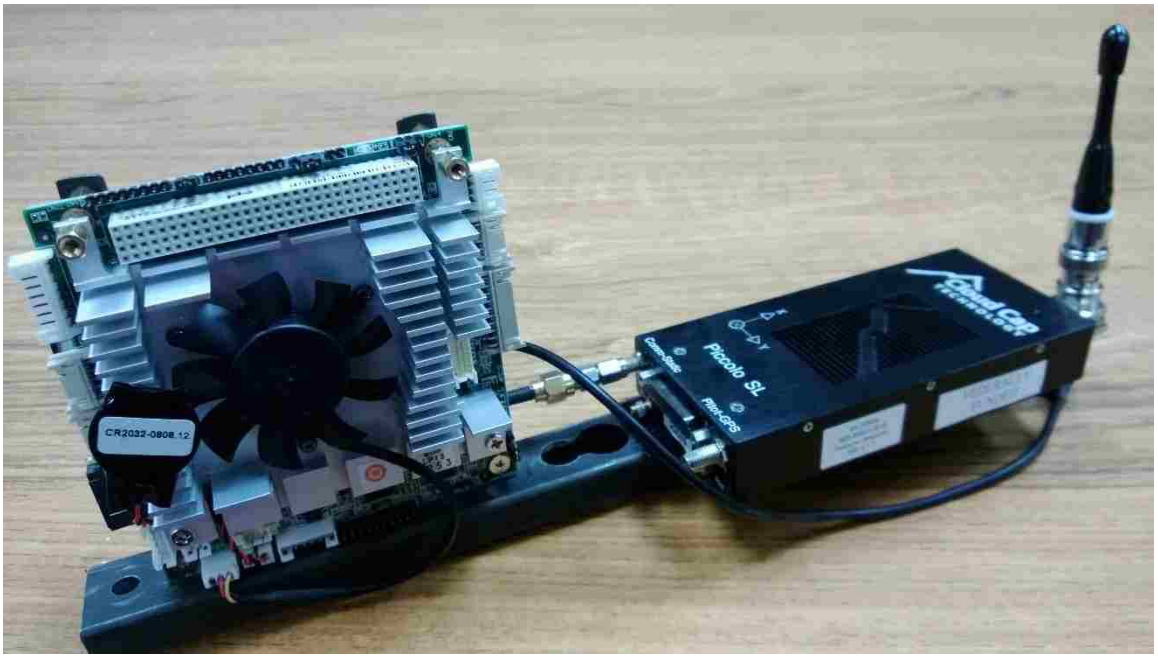


Figure 3.5 – The full autonomy subsystem. Pictured here, the On-Boar-Computer (OBC) is mounted on the steel chassis. The OBC is an Advantech PCM-3363 PC/104 single board computer running an Ubuntu operating system and MATLAB computer programming language.

The OBC is mounted to the steel chassis next to the Piccolo autopilot. The proximity of the OBC to the Piccolo autopilot allows for a direct serial connection between the two (pictured below).

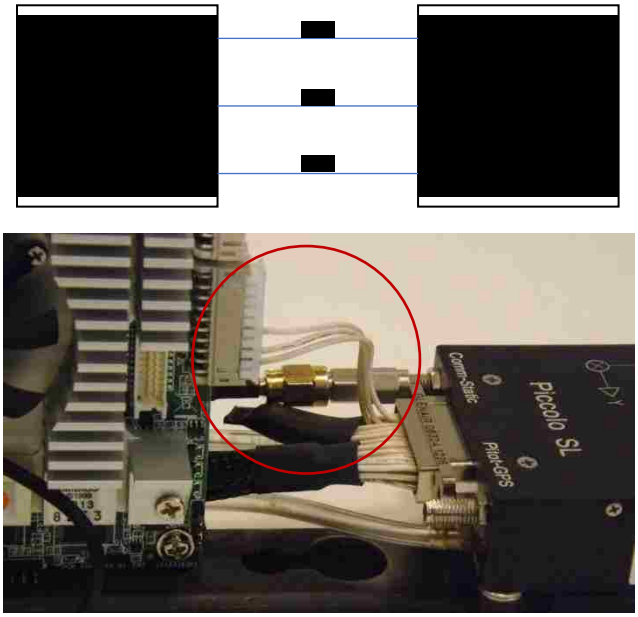


Figure 3.6 – (top) logical connections between the Piccolo SL autopilot and on-board-computer OBC. This is a serial interface. (bottom) the physical connection between the autopilot and OBC.

Power Subsystem

Beyond the normal power requirements for servo and motors, the Fox soaring platform requires power for the on-board autonomy equipment. The Piccolo SL is powered at a nominal 12 V and draws typical 0.3 A, with a maximum draw of 1.5 A. The OBC is powered at a nominal 5V and draws a typical 1.85 A at a max of 2.7 A during startup. The motor is powered at 12 V and draws a maximum of 45 A, but this draw is reduced to 0 A during a DS maneuver. However, the motor is still needed for

take-off and landing. Servos are powered at 5V and draw a typical 1A, but can become loaded during a DS maneuver and draw up to 2.8 A.

We targeted a powered flight time of approximately 7 minutes, which allows for the demonstration of powered maneuvers with some leeway for takeoff and landing. To supply the required power, we equipped the Fox platform with three batteries: one 3300mAh that is dedicated to powering the motor and servos; one 2200mAh that is dedicated to powering the OBC; and one 1350mAh that is dedicated to powering the Piccolo autopilot.

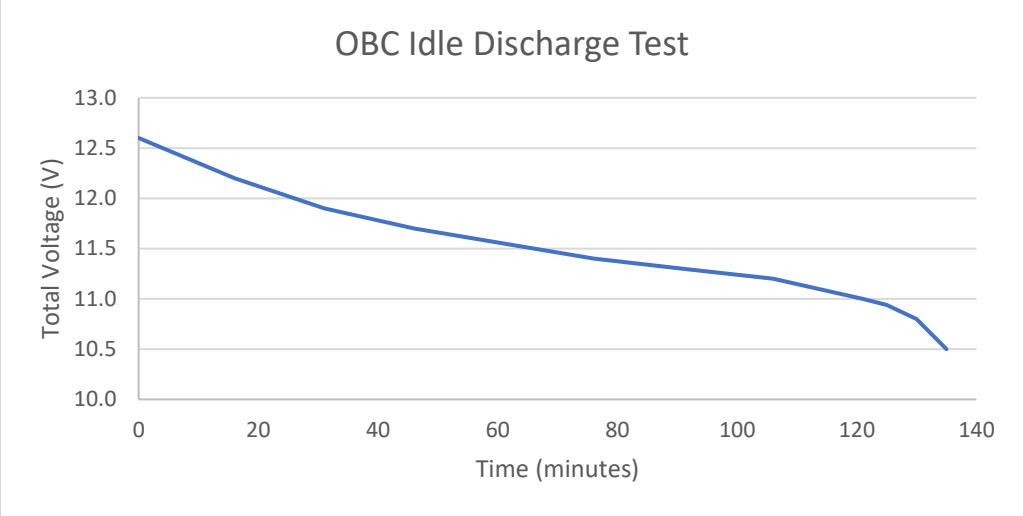


Figure 3.7 – Power discharge of OBC as it idles after boot. This represents the maximum running time of the OBC. Nominal voltage of the battery is 12V with a capacity of 2200mAh. The battery was charged to 12.60 V and the OBC was powered until the battery discharged. The final voltage was 10.50V. Total running time was 135 minutes.

In this scheme, the autopilot can run for a maximum of 206 minutes, the propulsion and actuation can run for 7 minutes, and the OBC can run for 145 minutes. This makes the limiting factor in flight time the propulsion system. For a soaring task, the limiting factor would be the OBC run time, which at over 2 hours is more than adequate.

Wiring diagrams for the power system are shown below. Since all batteries run at a nominal 12V, voltage regulators were used to regulate the voltage for the OBC and the servos. Additionally, for safety we included two fuses, one 0.75A fuse between the Piccolo and its power source, and one 5A fuse between the OBC's power source and the switching regulator.

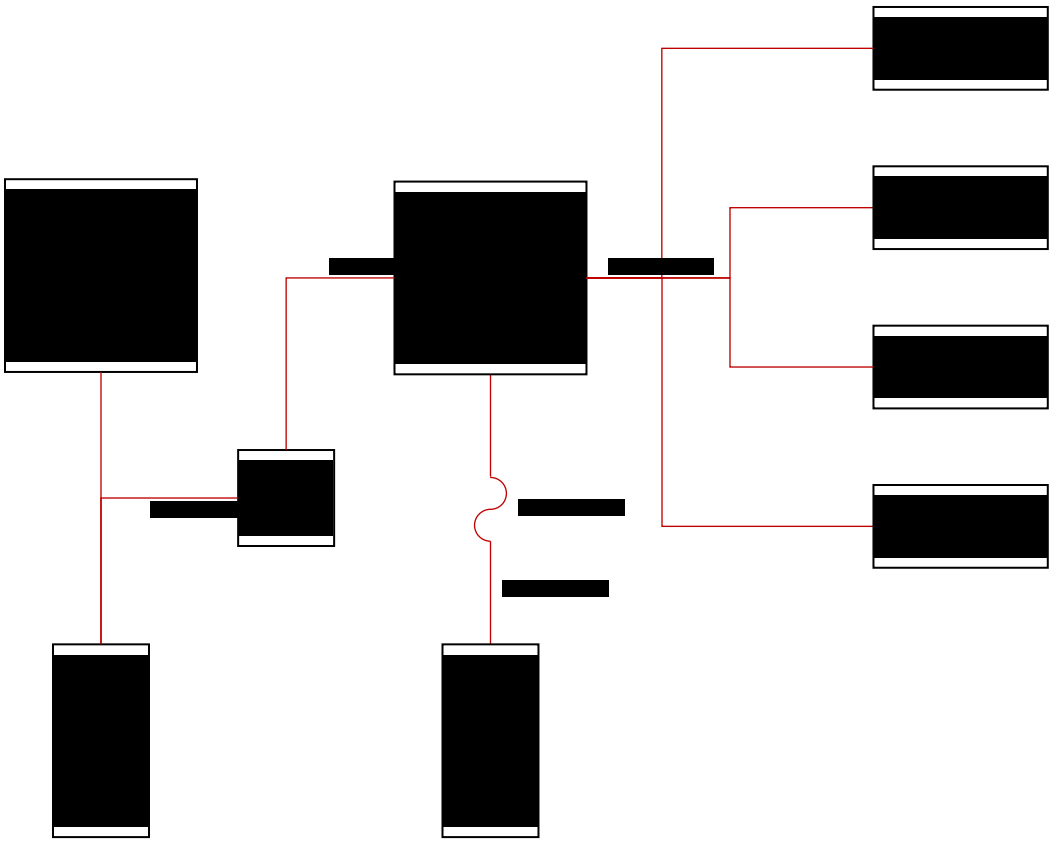


Figure 3.8 – First part of the power subsystem of the Fox soaring platform. A single 3300mAh battery powers the motor and control surface servos. The power to the servos is passed through the Piccolo SL autopilot. Physically, the 3300mAh battery is connected to the switching regulator, but power is passed directly to the motor. A 1350mAh battery powers the Piccolo SL directly, which is protected by a 0.75A fuse.

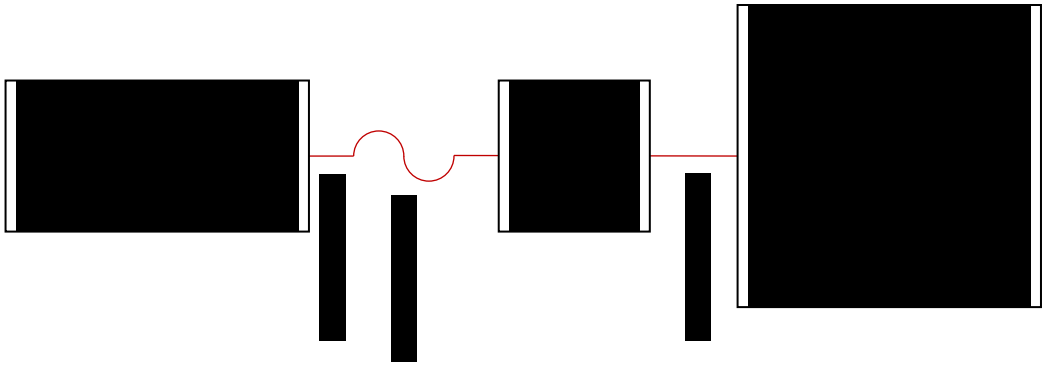


Figure 3.9 – Second part of the power subsystem of the Fox soaring platform. A single 2200mAh battery powers the Advantech PC/104 single board computer. Voltage is regulated from 12V to 5V through a switching regulator. The computer and regulator are protected by a 5A fuse on the battery side.

Figure 3.10 – Batteries and autonomy subsystem installed into the Fox fuselage. Pictured here are two 3300mAh batteries, one for powering the motor and actuators, and the other for powering the on-board computer; and a 1350mAh battery for powering the Piccolo autopilot. In alternative configurations, the 1350mAh battery is mounted behind the center of mass.

3.6. Portable Ground Station

The portable ground station is shown in Figure 3.11. Via radio link, it sends command packets to the Piccolo to control its behavior and listens for telemetry packets. A laptop computer is connected to the ground station via a serial link. It is used to visualize the state of the aircraft as it is in flight and provides an interface for issuing autopilot loop control packets, as well as uploading waypoints and mission boundaries. All telemetry is recorded on the laptop's hard drive for later playback and analysis.

Figure 3.11 – The portable ground station. The ruggedized case contains a radio transceiver that is used to receive telemetry packets from the Piccolo autopilot and send autopilot loop control packets upstream. Telemetry information is monitored on the laptop computer, which connects via serial port to the ground station. The pilot console (pictured) connects to physically to the ground station via a serial link and allows a pilot to take over manual control of the aircraft, bypassing the autopilot.

A pilot console connects to the ground station via a direct physical interface. The pilot console is configured to override the piccolo controller when a trigger is depressed,

allowing the pilot to take over when he or she notices trouble in the behavior of the aircraft.

3.7. Software Architecture

The software of the dynamic soaring platform is detailed in this section. The major components of the system are depicted in Figure 3.12 as a graph, with computational units shown as blocks, connected by messaging channels (blue arrows). In this section, we detail the functionality of each computational unit (which we call nodes), as well as the structure of the messages sent between them.

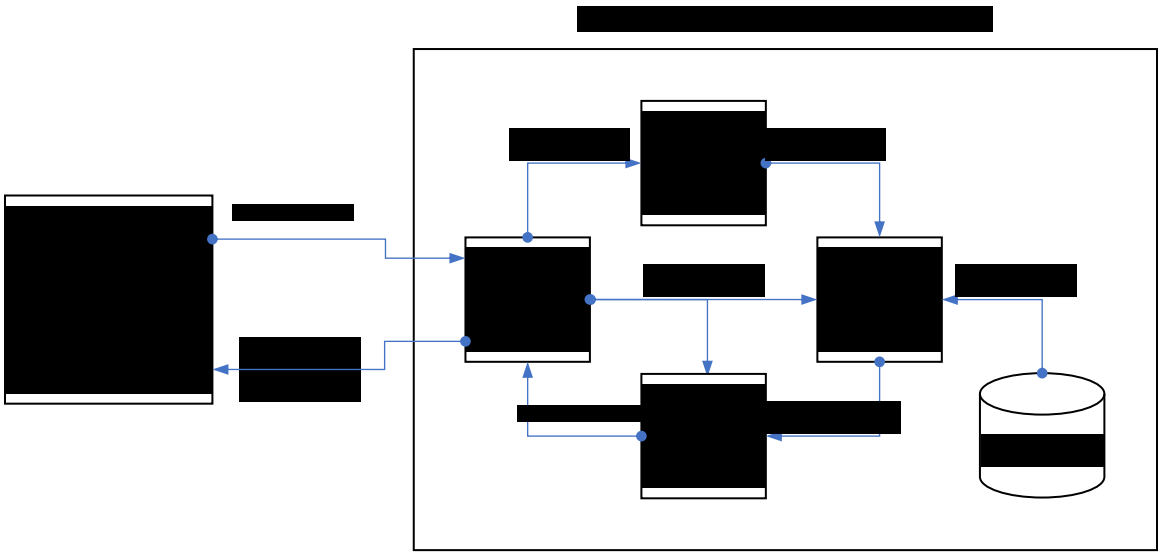


Figure 3.12 – The logical architecture of the dynamic soaring platform software system. The system is divided into two main components: (left) the Cloud Cap Piccolo SL autopilot. This autopilot is regarded functionally as a black box. It generates telemetry packets at 25 Hz and accepts incoming commands. (right) the Advantech PC/104 On Board Computer consumes telemetry from the autopilot and issues commands to enable following of dynamic soaring trajectories. The architecture of this system is a loosely connected graph of nodes that pass messages.

The loosely connected nature of this architecture allows the system to be robust and reconfigurable. If a component dies dues to software error, the system can detect this and restart any submodule. This system also allows us to run critical components in parallel. For example, the local planner node can run in duplicate, so if one dies, the other will simple take over. Finally, this architecture allows us to plug in different controllers easily by conforming to the message protocol.

Autopilot Packet

Autopilot packets are generated at a frequency of 25 Hz by the Piccolo SL autopilot. These packets contain telemetry information relating to the pose of the aircraft, true airspeed, as well as environmental details including temperature, air pressure, and wind speed. Upon generation, autopilot packets are sent over the serial link to the on-board computer and sent over radio link to the ground station computer. This allows the ground station to visualize the aircraft.

Waypoint Packet

Waypoint packets contain a linked list of waypoints. Each waypoint has a location in latitude, longitude, and altitude, as well as a pointer to the next waypoint in the list. The entire list of waypoints must be self-closing – that is, the last waypoint in the list must point to the first. Waypoint packets are sent from the OBC to the Piccolo autopilot over the serial connection. When a packet arrives at the autopilot, it is sent to the ground station computer over the radio link. Each waypoint in the packet is individually verified to be within the mission boundary. If any waypoint in the packet is invalid, the entire waypoint list is marked as invalid.

Autopilot Loop Control Packet

The autopilot loop control packet is the primary interface for controlling the DS platform. It consists of a target roll angle and a target pitch angle. These angles are generated by the DS planner on the OBC and sent to the Piccolo autopilot. The autopilot then attempts to reach these angles while obeying the angle rate limits set by the system configuration.

Telemetry Packet

The telemetry packet is derived from the Piccolo autopilot packet. It contains only the information from the autopilot packet that is necessary in powering the DS controller, namely: the 6-D aircraft pose, airspeed, and measured wind speed.

Wind Profile Packet

The wind profile packet is generated by the wind mapper node and denotes a map of the wind. It contains the maximum wind speed, as well as the direction it is blowing.

Target Trajectory Packet

The target trajectory packet is generated by the global planner node and denotes the trajectory to be followed by the local planning node. The target trajectory is a list of 32 waypoints with a 3D position in UTM coordinates, a 3D attitude in degrees, and a velocity. The packet also contains information about the environment, including pressure, temperature, the maximum wind speed, and the altitude and height of the boundary layer.

Communications Node

The communications node (comm node) runs on the OBC and is responsible for communicating with the Piccolo Autopilot. It runs at a frequency of 25 Hz and receives incoming autopilot packets from the Piccolo. These packets are transformed into telemetry packets, which is published for the wind mapper, the global planner, and the local planner. The comm node also subscribes to target trajectory packets from the global planner, which it transforms into waypoint packets for the autopilot. Finally, the comm node subscribes to autopilot control loop packets and forwards them to the Piccolo autopilot.

Local Planner Node

The local planner runs at a nominal frequency of 25 Hz. This node subscribes to target trajectory messages and telemetry messages. It uses information from these two sources to generate a command to be issued to the autopilot. The algorithm of the local planner is generic and can accept a controller that is specific for the particular mission objectives. Options include: a circle planner, a feed forward LQR controller [45], a sample-based controller (detailed in Chapter 4) and a reinforcement learning controller (detailed in Chapter 5).

Global Planner Node

The global planner subscribes to wind map and telemetry messages. At the beginning of each DS loop, the global planner selects an appropriate dynamic soaring trajectory from the local storage as informed by the wind map and current airspeed indicated by the telemetry message. Trajectories are parameterized by the maximum wind at the

top of the boundary layer, the height of the boundary layer, and airspeed. The selected trajectory is published for consumption by the local planner and comm node, which sends the trajectory as a waypoint message to the autopilot.

Wind Mapper Node

The wind mapper runs at a nominal rate of 25 Hz and subscribes to telemetry messages and listens specifically for wind speed and altitude data on those messages. The wind mapper uses these data to construct a map of the wind over an altitude. The map is summarized as a prevailing heading, and the maximum wind speed in that direction. This summary data is published for consumption by the global planner to inform its selection of the next dynamic soaring trajectory. This node is beyond the scope of this dissertation, but the method of the node can be found in [46].

Local Storage

Since the generation of dynamic soaring trajectories is expensive in terms of time, we pre-compute a large set of trajectories ahead of time and store them in an in-memory database for quick lookup. The trajectories are indexed by maximum wind speed, the height of the boundary layer, and the airspeed of the aircraft at the bottom of the lowest altitude in the trajectory. The database is queried once per DS cycle, and the closest matching trajectory is selected for publication to the local planner and the comm node.

3.8. Platform Validation

In this section we present various tests used to validate the design of the Fox airframe for soaring purposes. Some of the tests presented here were prescribed by the Piccolo SL autopilot integration guide, to ensure the proper operation of the autopilot during flight. Others were done to make sure the Fox could withstand the forces inherent in dynamic soaring.

3.8.1. Wing Load

A feature of dynamic soaring loiter maneuvers is a high-load turn on the downward dive segment of the cycle. We tested the wings of the Fox soaring platform to ensure it could withstand this load before deploying the Fox into the field. We set a target limit of +7g for this test, considering limits on the autopilot and air speed constraints of a maximum of 40 m/s in 9 m/s wind.

Experimental Setup

The fuselage of the Fox was secured to a saw horse using ratchet straps. The wings were attached using the stock steel spar and spring, and were allowed to freely deflect, as pictured in Figure 3.13. A total of 20 plastic garbage bags were each filled with 1kg of pea gravel. The weighted bags were then placed on the wings one by one, starting at the fuselage and extending out toward the wing tips. When all the bags were placed on the wings, we allowed 5 minutes to elapse, simulating 5 continuous minutes of a high turn maneuver at +7g. Because the wings are symmetric in the x-y plane, we assume that the +7g test is valid for -7g, although for a counter-clockwise loiter loop there would be no negative-g turns.

Figure 3.13 – The Fox platform during the wing load test. One-kilogram bags of pea gravel were placed on the wing to simulate a high load turn. The load test was performed for 5 minutes, and then the bags were removed, and the structure was inspected for damage. Preliminary testing revealed damage to the structure of the wings and wing spar.

Results

The first time we conducted this test, it revealed several problems with the structure of Fox. First, we found cracking at the base of the wings where the eye hooks are embedded. Second, the spring was stretched to a point where the connecting hooks were deformed.

Conclusions

In light of these test results, we made two adjustments to the Fox design, pictured in Figure 3.14. These changes include:

- More robust eye hooks. The original Fox wing design was a small eye hook screwed into the wing. We re-designed this with a nylon standoff and a larger eye hook. The standoff and eye hook were then super-glued to the wing.

- Plastic security tie. Because the spring hooks were deformed, we added a plastic security tie as a failsafe incase the spring became undone during flight, preserving the wings to the body and allowing a pilot to manually land the vehicle.

After these changes were made, the load test was performed again, specifically to test the efficacy of the reinforced eye-hooks. This second subsequent test revealed no damage to the integrity of the wings.

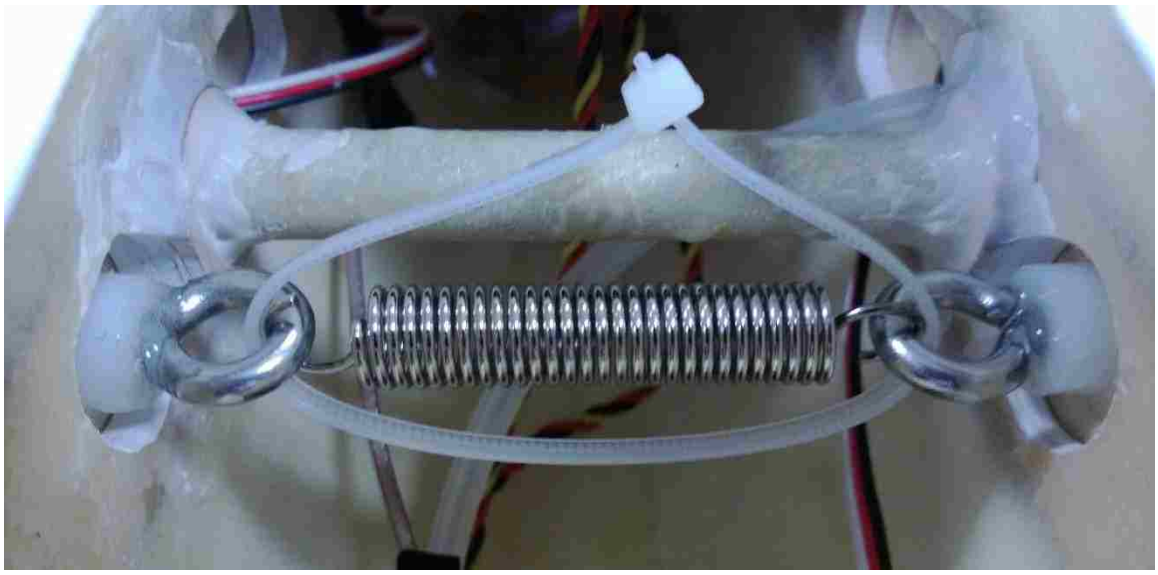


Figure 3.14 – An inside look at the wing joint inside of the Fox fuselage. The stock joint is two eye hooks connected by a spring. The wing spar runs left to right through the tube in the center of the image. Results of the test led us to reinforce this join with more robust eye hooks, and a plastic security tie.

3.8.2. Temperature

The Piccolo SL Autopilot is rated to operate under a temperature range of -40C to 80C. The Piccolo SL typically dissipates about 4W of power while running. In our application, the Piccolo is mounted next to a single board computer, which typically dissipates nominally 10W of power while running. Since both pieces of equipment operate in the same enclosed area, they work together to increase the ambient

temperature of the cabin. This test was designed to determine if the maximum steady state internal CPU temperature of the Piccolo SL exceeds its operational limits when running in conjunction with the SBC.

Experimental Setup

Both the Piccolo SL and Advantech PCM-3363D single board computer were mounted to the custom steel rig, which was then installed and secured in the aircraft fuselage, complete with wings attached. Power was applied to the Piccolo at a nominal voltage of 12V. A connection was established with the Portable Ground Station, and telemetry was set to log at 25Hz. Power was then applied to the SBC at 5V through a voltage regulator, which was powered by a 12V battery. When all systems were running, the canopy was attached, and the cabin was sealed with tape.

Results

Figure 3.15 depicts the result of the temperature test. Here, the internal CPU temperature of the Cabin is reported in the blue line. The initial temperature of the CPU was reported at 36C, while the initial ambient temperature was reported as 23C. The final steady state ambient cabin temperature reached 36C, while the internal Piccolo CPU temperature reached 50C, as depicted in the figure.

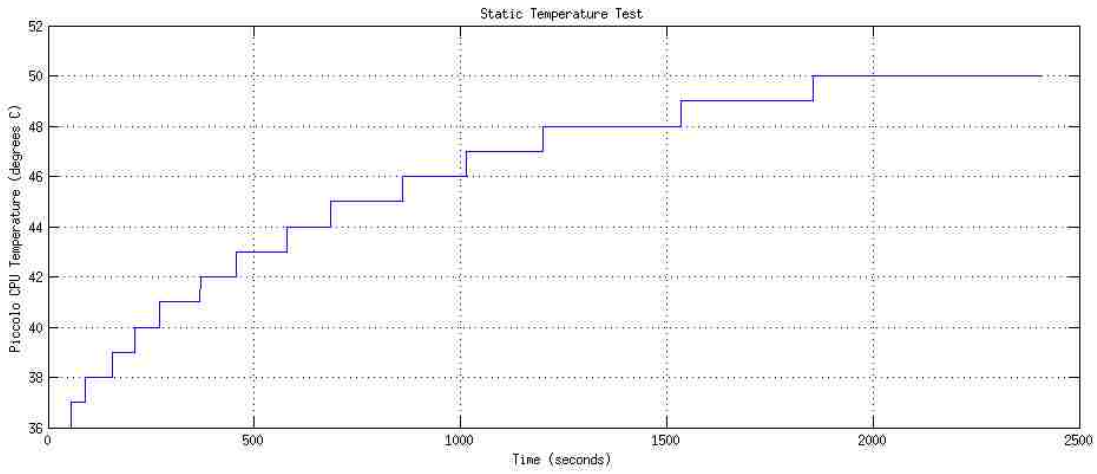


Figure 3.15 – Results of the temperature test. A steady state internal temperature was achieved at 50C over roughly 30 minutes. This is within the operational range of the CPU and Piccolo SL at 80C.

Test Conclusions

At 36C and 50C respectively, both the steady state ambient temperature and internal CPU temperature were well below the maximum 80C operational limit suggested by Cloud Cap Technology's Piccolo SL specifications. Further, since this was a static test, it represents a worst-case-scenario, as during normal operation air will flow through the cabin, lowering the ambient temperature and cooling both the Piccolo SL and SBC. Thus, we can conclude that exceeding the temperature limitations of the Piccolo SL Autopilot will not be an issue while the plane in operating.

3.8.3. Communication Link Signal Strength

In response to signal strength issues encountered during field tests, a new antenna was designed and installed in the U-Fox UAV. The antenna (Fig 3.16) is a whip antenna with approximately .6m of shielded coaxial cable, terminating an exposed radiator approximately 1.2m in length. The antenna is installed in the airframe such that exposed element is positioned vertically within the UAV's vertical stabilizer, and the coaxial portion runs in the fuselage from the tail to autopilot mount point between the wings.

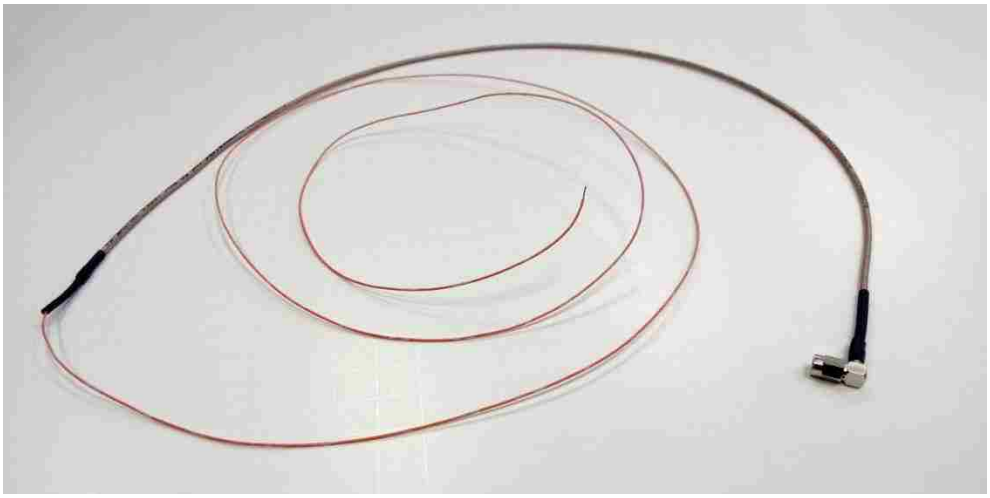


Figure 3.16 – The whip antenna design replaced an earlier design after signal strength issues were encountered. This antenna is approximately 2 m long, with .6 m of shielded coaxial cable, and 1.2m of exposed radiator. The antenna runs through the length of the fuselage, starting in the vertical stabilizer and terminating near the cabin for connection to the Piccolo SL autopilot.

Experimental Setup

A whip antenna of the design described in the previous section was installed in the U-Fox airframe. The radiating element was installed vertically in the U-Fox's vertical stabilizer and trimmed to approximately 26cm. All electronics (SBC, Piccolo SL),

batteries, GPS, and servos were installed in the airframe to best represent a flight-ready configuration.

The Piccolo Portable Ground Station was positioned in a level field, with its antenna and GPS mounted on a sub-compact automobile, and a communication link was established with the UAV directly next to the antenna. Radiating power was set to 1.0W on both the ground station and the autopilot.

North	0 deg
East	0 deg
South	0 deg
West	0 deg
North	-60 deg
East	-60 deg
South	-60 deg
West	-60 deg

Table 3.1 – A summary of the 8 directions and angles of the signal strength test. Each direction and angle of the aircraft with respect to the ground was tested at 200m. The -60-degree roll test was chosen to represent the maximum angle of roll we expect to see during dynamic soaring maneuvers.

The UAV was walked to a distance of approximately 200m from the ground station. At this distance, the airframe was oriented in the following directions/bank combinations to test the robustness of the antenna.

After this test was completed, the UAV was manually walked back to the ground station, and the test was repeated with the transmitting strength of the Piccolo SL reduced to 0.1W.

Results

We measured two variables to characterize the performance of our communication link: the packet acknowledgment ratio (AckRatio) and the Received Signal Strength Indication (RSSI). According to Cloud Cap's documentation, "The acknowledgement ratio is the ratio of acknowledged communications frames to polled communications frames. It is a measure of the communications link performance to these avionics. 100 is best and 0 is worst." Likewise, RSSI measures the signal strength of the data link in units of dBm, and ranges from a -50 (best link) to typically -115 (worst link).

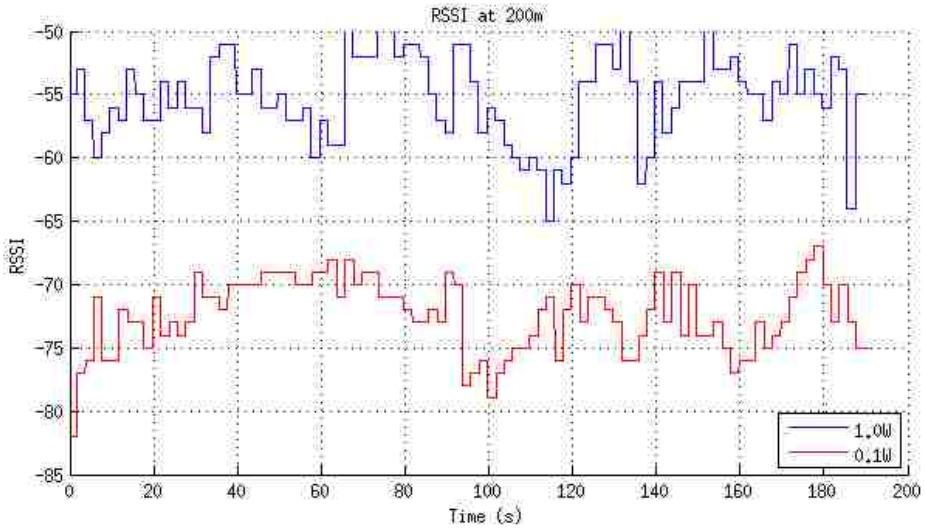


Figure 3.17 – The received signal strength test. (top) performed at 1.0W (bottom) performed at 0.1W. This test shows even at the lower power setting, the received signal strength never drops below the recommended -85 at 200m range. At the higher power setting, which is the target test setting, the RSS never drops below -65.

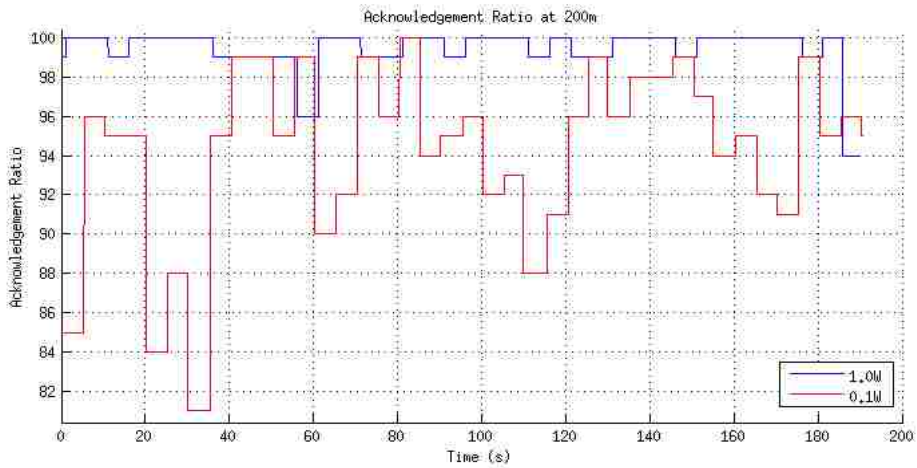


Figure 3.18 – The acknowledgement ratio at 200m for 1.0 W (top) and 0.1W (bottom). This measure represents the percentage of packets sent from the ground station that are acknowledged by the autopilot as received. At full power, this ratio never drops below 94%, with a mean acknowledgement of 99.42%. At the lower power, this ratio never drops below 81%, with a mean acknowledgement of 94.18%.

	Test 1 (1.0W)		Test 2 (0.1W)	
	AckRatio	**RSSI**	**AckRatio**	**RSSI**
Max	100	-50	100	-67
Min	94	-65	81	-82
Mean	99.42	-55.26	94.18	-72.42
Standard Deviation	1.10	3.34	4.50	2.84

Table 3.2 – Summary statistics for the signal strength test.

Test Conclusions

Per Cloud Cap's radio guidelines, their PCC software is configured to report a "Signal Strength" error when either the AckRatio drops below 85, or the RSSI drops below

100. In Table 3.2, we can see that even at 0.1W transmission power, the average AckRatio is 94% and the average RSSI is -72.

3.8.4. Vibration

One concern in properly integrating an autopilot system is whether aircraft induced vibrations, principally those emanating from the motor, cause substantial orientation drift or noise in gyroscopic sensor measurements. To test this on our aircraft platform, we followed the procedure outlined by the Vehicle Integration Guide published by Cloud Cap Technology [47]. The Vibration test procedure is described as follows:

1. Establish a wireless link between the Piccolo Autopilot and Portable Ground Station
2. Enable Fast Telemetry at 25Hz
3. Slowly run the engine through its full RPM range, noting the 3 orientation angles and 3 gyroscopic rate outputs
4. Acceptable rates for the gyroscope rates are below double digits
5. Acceptable drift for the orientations is less than a few degrees over the entire operational RPM range

Experimental Setup

The aircraft was fixed to a table to isolate motor vibrations as depicted in Figure 3.19. The aircraft was mounted on a compressible surface and tied down with straps. Each

wing was also strapped to a supporting structure to prevent roll induced by gravity, and the wing roots were taped to the fuselage.

Figure 3.19 – The experimental setup of the vibration test. The fuselage was leveled under two compressible rolls and secured to a table using ratcheting ties. The motor was then run through its operational range and gyroscope data was recorded via the Cloud Cap Command Center logging facility.

Internally, the Piccolo SL autopilot was mounted on our custom steel rig with hardware recommended by Cloud Cap Technology. The rig was installed in the aircraft and securely tightened down. All interface cables were securely mounted and tightened with the appropriate hardware to prevent any unnecessary vibrations.

Throttle was controlled via an RF receiver mounted in the cabin with Velcro. The Piccolo was powered by a 12V, 1350mAh battery, while the motor was powered by a

12V 3300mAh battery. The RF receiver was powered by a voltage regulator spliced into the 3300mAh battery.

When all connections were established, and power supplied, the cabin was covered with a canopy and sealed with tape. And the test commenced.

With a radio transmitter paired to the RF received, 3 levels of throttle were supplied. Each throttle level was sustained for 30 seconds before moving to the next.

Results

Figure 3.20 – 3.21 depicts the results from the vibration test. The top figure depicts orientation drift, while the bottom figure depicts gyroscope rate readings. The data were convolved with a 5x1 normalized filter.

The bottom figure clearly depicts three different RPM levels. The entire test lasted about 180 seconds. Thereafter, the throttle was reduced to zero and the aircraft was returned to a steady state.

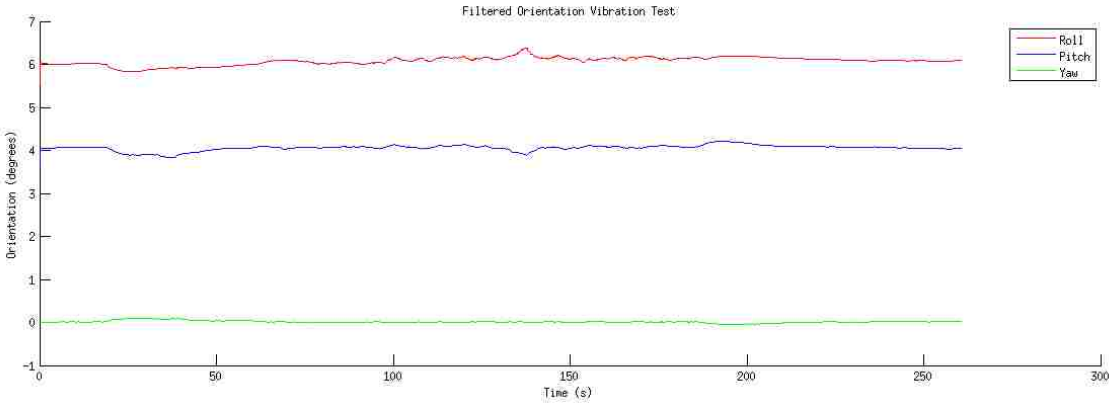


Figure 3.20 – Orientation results for the vibration test. The mean absolute deviation from the mean orientation across all axes stayed below 0.7 degrees. Since the airframe did not move during the test, a very small deviation was expected.

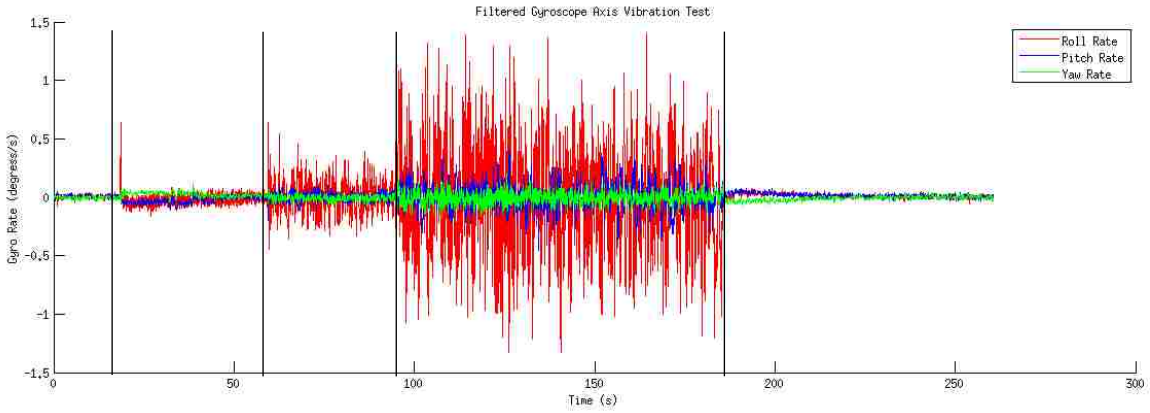


Figure 3.21 – Gyroscope results for the vibration test. The three stages of the test are indicated by vertical lines. The first stage beings at ~20 seconds, where a spike in roll rate is observed as the motor is started. The second stage begins at ~60 seconds, when the RPM of the motor is increased. The final stage begins at ~100 seconds, when the motor is set to its highest setting. The motor is turned off at ~180 seconds.

	Max absolute deviation from mean	Mean absolute deviation from mean	Standard deviation of absolute deviation from mean
Roll Rate	1.39 deg/s	0.15 deg/s	0.22 deg/s
Pitch Rate	0.45 deg/s	0.04 deg/s	0.06 deg/s
Yaw Rate	0.19 deg/s	0.03 deg/s	0.03 deg/s
Roll	0.31 deg	0.07 deg	0.06 deg
Pitch	0.23 deg	0.04 deg	0.05 deg
Yaw	0.78 deg	0.01 deg	0.02 deg

Table 3.3 – Summary statistics for the result of the vibration test.

Several statistics were evaluated from this data to evaluate the effectiveness of the mounting solution in the vibrational test. These are summarized in Table 3.3. For a given sensor reading X, these values were calculated as follows:

$$\text{mean} = \text{mean}(X);$$

$$\text{max\_absolute\_deviation\_from\_mean} = \max(|X - \text{mean}|);$$

$$\text{mean\_absolute\_deviation\_from\_mean} = \text{mean}(|X - \text{mean}|);$$

$$\text{standard\_deviation\_absolute\_deviation\_from\_mean} = \text{std}(|X - \text{mean}|);$$

Conclusions

Referring to the vehicle integration guide, acceptable values for rates were below double digits, and acceptable values for orientation drift were less than a few degrees over the entire operational RPM range. Here, we see just that for all sensor values. Hence, we conclude that the mounting solution for the Piccolo SL Autopilot

provides sufficient isolation from motor vibration.

3.8.5. Wind Mapping

One of the primary objectives of this work was to demonstrate real-time wind mapping that would support flight on an actual aircraft. In order to characterize the performance of the proposed approach, "ground truth" data were needed. While these were readily available for simulated wind fields, estimating wind velocities in proximity to an actual aircraft during flight was significantly more challenging. To address this requirement, we employed a vision-based approach for ground truth wind field estimation.

Experimental Setup

During flight testing, brightly colored balloons containing an air-helium mixture were released serially from the ground, so their trajectories would carry them in the vicinity of the aircraft flight path. The balloons were then tracked over time using what amounted to a wide baseline (e.g., 50-70 meter) stereo vision system using a pair of Point Grey Chameleon 1280x960 video camera systems that logged images at a rate of 2 Hz. Point correspondences between the two sets of camera images were then recovered manually for each balloon track during a post-processing phase. Using these correspondences in conjunction with a three-dimensional reconstruction approach based upon Hartley's method [48], the relative position and orientation of both camera systems, as well as the positions of the tracked balloons, were recovered to a scale factor. The scale factor was obtained by measuring the camera baseline. The balloon position estimates were then transformed to an earth-centered coordinate frame using GPS position estimates for the cameras, as well as measurements of camera azimuth

and elevation from a compass and inclinometer, respectively. With the balloon positions known, their velocities – and as a consequence the Northing/Easting components of the wind field – were estimated using a finite difference approach vs. time with temporal smoothing to help mitigate high frequency noise.

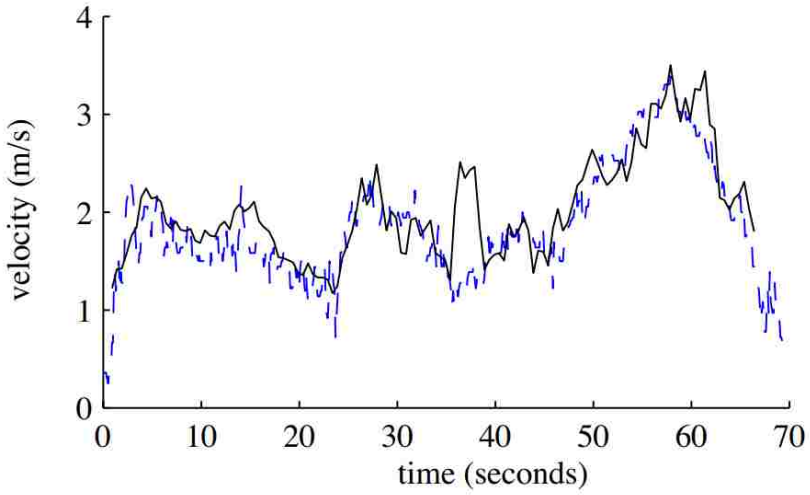


Figure 3.22 – Balloon horizontal velocity magnitude estimates vs. time for the vision system (solid black line) and GPS (dashed blue line). In this trial, the mean absolute deviation vs. time between the two approaches was 0.24 m/s.

To validate the vision-based approach, initial experiments involved tracking a tethered balloon rig carrying an EagleTree eLogger V4 with a 10 Hz WAAS enabled GPS module as payload [49]. The motivation was to use the logged GPS velocities for benchmarking the vision system's tracking performance. A total of 6 launches were conducted from different initial positions at standoff distances of ≈110 − 180 meters from the camera systems. The balloon rig was released at ground level and allowed to rise with minimal resistance while attached to a 125-meter-long, 0.15 mm diameter tether. Each trial was considered completed once the end of the tether was reached. Results from these experiments showed that the mean absolute deviation between the

two velocity estimates vs. time for all trials was 0.35 m/s (minimum 0.22 m/s, maximum 0.55 m/s). Results from a single launch are shown at Figure 3.22. Note that these error levels represent the compounding of both the vision tracker and GPS velocity errors. These results indicate that the vision-based tracking system provides an effective means for estimating the wind velocity at standoff distances in excess of 200 meters.

Results

To assess the performance of wind field estimation, a flight test was conducted using the aircraft pictured in Figure 3.23. The planform is based upon an inverse Zimmerman design. It is made of two half ellipses, both having the same minor axis but the forward ellipse having a major axis that is three times that of the rear ellipse.

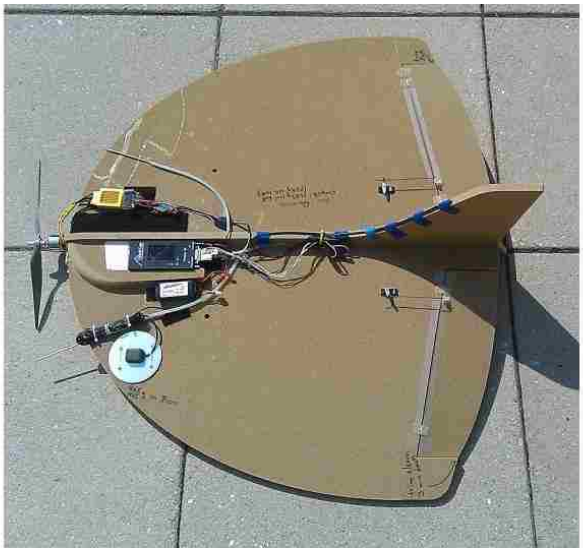


Figure 3.23 – Testing platform used in this work. During testing, the aircraft was flown manually while telemetry data were logged via the on-board Piccolo SL autopilot system.

A fixed fin was used in conjunction with two elevons. The design has very benign stall characteristics, is capable of operating safely in turbulent air, and can glide down

steeply to land when required. Full stall landings at almost zero airspeed are easy with this aircraft. The platform was equipped with the Piccolo SL autopilot and pitot tubes for the purpose of logging GPS and wind data but was flown manually.

The aircraft was flown manually (radio controlled) while telemetry data were logged via the on-board Piccolo SL autopilot system. During the approximately 6.5-minute flight, 9 balloons were launched and tracked with the vision system to estimate the wind field. Figure 3.24 shows the aircraft flight path, as well as the balloon tracks.

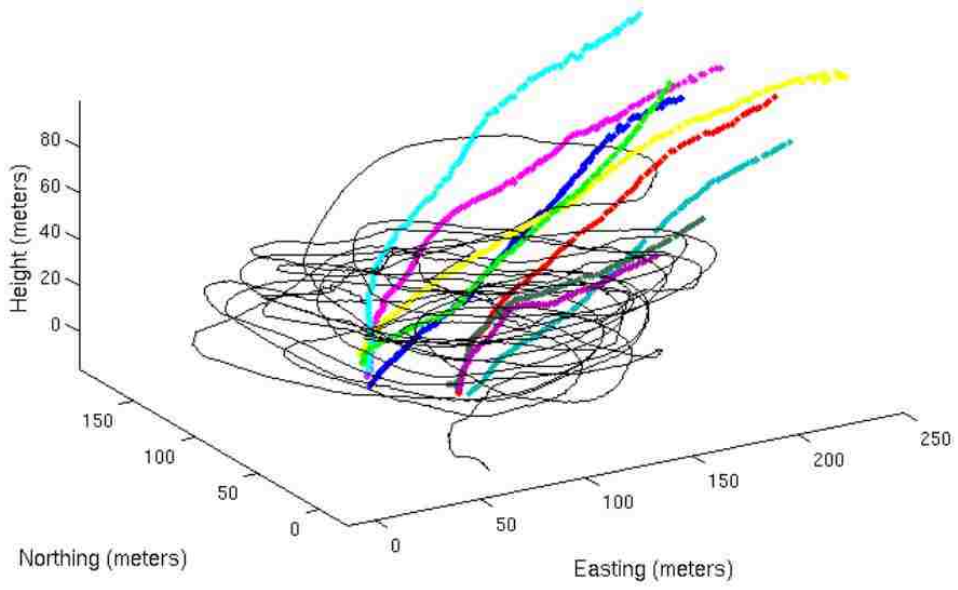


Figure 3.24 – Aircraft flight path (black solid line) and balloon tracks (dotted color lines) from flight testing. The latter were recovered by the vision tracking system

When post-processing the aircraft telemetry data, we observed GPS velocity drops during significant portions of the flight. These drops compromised the aircraft velocity estimates, and as a result the algorithm's ability to estimate wind velocity. As a result, we constrained our analysis to an 80 second window where GPS velocity estimates

were *mostly* available. Analysis of the data using the wind mapper algorithm was performed by out collaborators at Penn State University and is presented in [50].

Conclusions

The analysis of the collected data by Penn State found that while performance of the wind estimator is good in the linear wind shear case examined here (both the gradient and the wind speed at a reference altitude were well-estimated), some tuning of parameters in the Kalman filter should further improve performance. Results from this test encouraged the use of splines to parameterize the wind field, which is detailed in [46].

3.9. Flight Tests

Figure 3.25 – The Fox gliding platform landing in a snow-covered field after a flight test. This particular test was conducted under manual control.

We conducted 17 field missions consisting of a total of 46 flights of the Fox platform between 2013 and 2014. Almost 4.5 hours of flight time were recorded on the platform. Early testing revealed deficiencies in the design of the platform including problems with the communication signal strength, and the center of gravity location. When these issues were resolved, we conducted tests to verify the model parameters

embedded in the auto pilot configuration. After tuning the auto pilot, we conducted tests to verify our various controls, including the Sample Based Controller outlined in Chapter 4. Over a series of tests designed to test the dynamic soaring capabilities of the platform, we discovered issues relating to weak engine power, as well as an insufficient BEC, which warranted upgrades of both components.

3.10. Summary

In this section we presented the target airframe for our dynamic soaring system. Each component presented in this chapter plays an important role in the overall operation of the dynamic soaring mission. What's left in the remainder of this dissertation is a discussion of the details of the software components of the system. In each of the subsequent chapters, the system is designed to work on the parameters of this platform specifically. While the software and the hardware aspects are not directly tied together, if a different platform is used, then the parameters must be updated accordingly.

Chapter 4

Trajectory Generation and Following

In this chapter we present a trajectory following controller for the Fox dynamic soaring airframe. First, we detail a method to generate loiter-style dynamic soaring trajectories that orbit a fixed-point. This method is integrated into a planner that selects the most appropriate trajectory given the environment and aircraft state. This planner is referred to as the Global Planner. Then we detail a method of following the dynamic soaring trajectories that samples from a point-mass motion model for the Fox platform.

4.1. Related Work

The technique for dynamic soaring trajectory generation presented in this chapter is based on the work of Zhao, who formulates dynamic soaring as a nonlinear optimal control problem [51]. The technique he presents uses a nonlinear solver to find a feasible trajectory given a set of linear and nonlinear constraints, as well as an objective function that is maximized. Depending on the number of parameters in optimization problem, the solver can take a considerable amount of time to converge (on the order of 30 seconds on state-of-the-art circa 2015 desktop-class PC hardware). The PC-104 single board computer presented in Chapter 3 for use as the on-board

computer takes several minutes to arrive at a solution. With soaring cycle times on the order of 10 seconds, this solution time is prohibitive for *in situ* trajectory generation. Several methods have been proposed to generate dynamic soaring trajectories quicker. Akhtar *et. al* propose mapping the optimization problem into the output space, which results in a solution that takes 8 seconds to converge for a loop time of 176 seconds on circa 2008 hardware [52]. For our purposes, this convergence time is still too long for use during flight. Another technique presented by Ariff and Go use so-called Dubin's Curves and relaxed constraints to reduce the degree of computation by half compared to other methods, but the constraints relaxed in this work (loitering trajectories) are important to ours [53]. More recently, Mir *et. al* specifically look at small unmanned aerial vehicles with severe limitations to on-board processing, and propose a technique called GPOPS to achieve convergence times of less than 5 seconds [54].

There have been a variety of methods proposed for controlling dynamic soaring aircraft. Barate *et al.* proposed a set of hand-crafted rules based on the phases of dynamic soaring flight listed in Chapter 1, which are optimized using an evolutionary algorithm [55]. This yields good performance but requires a substantial training time that must be re-incurred to modify the controller. Lawrance and Sukkarieh propose a method similar to the one presented in this chapter, in that it uses the motion model of the aircraft to generate candidate trajectories which are scored based on an optimality criterion [56]. The best candidate trajectory is selected, and the state of the aircraft is advanced using that control. The process is repeated until a time window elapses. This is a form of a rapidly expanding tree planner, and only considers a few

candidate actions due to resource constraints. Flanzer proposed a repetitive control process that uses errors from previous cycles to optimize energy gain on future cycles [57]. This helps the aircraft in a steady state dynamic soaring loop, but what's sometimes more fraught is reaching a steady state. Before the steady state is reached, repetitive control cannot help so much because each orbit is very different from the last. Most recently, Gao *et al.* proposed breaking an optimized trajectory into pieces congruent with the phases of soaring flight, and then approximating each piece as a parameterized equation to be followed with a 4-stage controller – customized for each of the stages of DS [58]. Their solution manages to soar, but the results presented do not closely resemble reference trajectories. Park presents a tracking controller that uses a linear quadratic regulator to follow a reference trajectory [59]. We explored the use of this controller for dynamic soaring in [45] and found good performance for a variety of dynamic soaring scenarios, but better tracking performance was achieved with the work presented in this chapter.

4.2. Chapter Contributions

We claim two contributions in this chapter. First, is a global planner for generating a plan for reaching a steady-state dynamic soaring loiter pattern. The related planners in the literature mainly focus on the generation of a single soaring trajectory in a hypothetical context. The global planner presented herein is for the specific context of achieving a loiter pattern from a level orbit. This involves first, mapping the wind field, and then following a series of dynamic soaring trajectories. They must be followed one after the other and provided to the controller at the control frequency of 10 Hz. None of the methods presented in the literature generate dynamic soaring

trajectories at such a rate. The proposed planner cannot either, but we trade time for space by pre-computing trajectories and storing them in a searchable database.

The second contribution of this chapter is the adaptation of a controller for a real platform. Most soaring controllers presented in the literature are demonstrated in a simulated environment, which side-steps some of the complications of real hardware, such as latency, and real-time constraints. The controller presented herein is built for the Fox soaring platform and Piccolo SL autopilot and contains several adaptations that were found necessary through experimentation on real hardware. Experiments are presented to demonstrate the efficacy of this controller in simulation and on the actual hardware in field tests.

4.3. Aircraft Motion Model

As a prerequisite to generate dynamic soaring trajectories, we need a model for our Fox soaring aircraft presented in Chapter 3. We model the airframe as a point mass subject to drag and lift forces. State variables are the aircraft's position in R^3, $[x, y, z]$; and the aircraft's yaw ψ and pitch γ. Control inputs are assumed to be the roll μ and coefficient of lift C_L. We can write the track relative to the air along the unit vector [60]:

$$e^v = cos(\gamma)sin(\psi)\hat{\imath} + cos(\gamma)\hat{\jmath} + sin(\gamma)\hat{k} \qquad \text{Equation 4.1}$$

Where $\hat{\imath}$ points East, $\hat{\jmath}$ points North, and $\hat{k}$ is normal to the Earth's surface; and γ is the pitch of the aircraft relative to the air, while ψ is the heading angle. If we take the

derivative, we get the velocity of the aircraft, $\dot{r}$. If we assume the wind only has a component in the $\hat{\imath}$ direction, then the velocity of the aircraft plus the wind drift is

$$\dot{r} = (Vcos(\gamma)sin(\psi) + W_x)\hat{\imath} + Vcos(\gamma)cos(\psi)\hat{\jmath} + Vsin(\gamma)\hat{k} \quad \text{Equation 4.2}$$

We introduce drag, D and lift L:

$$D = \frac{1.5\rho V^2}{2}\frac{3w_a}{(3.46\log_{10}R_w - 5.6)^2} + \frac{0.9\pi f_d f_l}{\left(3.46\log_{10}R_f - 5.6\right)^2} + \frac{L^2 2}{\rho V^2 w_s^2 \pi\varepsilon} \quad \text{Equation 4.3}$$

$$L = \frac{\rho V^2}{2} w_a C_L \quad \text{Equation 4.4}$$

Where w_a is the wing area, $R_w = c_w V\rho/\mu_{air}$, $R_f = f_l\rho/\mu_{air}$, w_a is the wing area, w_s is the wing span, $c_w = w_a/w_s$ is the chord width, and ε is the Oswald's span efficiency factor.

The final equations of motion are:

$$\dot{V} = \frac{-D}{m} - gsin(\gamma) - \dot{W}_x cos(\gamma)sin(\psi) \quad \text{Equation 4.5}$$

$$\dot{\psi} = \frac{1}{mVcos(\gamma)}\left(Lsin(\mu) - m\dot{W}_x cos(\psi)\right)$$

$$\dot{\gamma} = \frac{1}{mV}\left(Lcos(\mu) - mgcos(\gamma) + m\dot{W}_x sin(\gamma)sin(\psi)\right)$$

Fox Model Parameters

The Fox has a 2.8m wingspan and a loaded weight of 4.3kg. Beyond a wingspan, the point mass model requires a wing area, and fuselage length and diameter for the

calculation of drag. These parameters specific to the Fox airframe are presented in Table 4.1:

Loaded mass	4.3 kg
Fuselage length	1.359 m
Fuselage diameter	0.1 m
Wing area	0.634 m
Wing span	2.8 m
Oswald's efficiency factor	0.8

Table 4.1 – Summary of aircraft parameters used in the point-mass aircraft model.

4.4. Atmosphere Model

The point mass simulation models drag and lift on the aircraft using supplied environmental parameters

R	287.1
g	9.80665 m/s^2
h_s	11000
L	0.0065
T_0	288.16
ρ_0	1.225
μ_{air}	1.43e-5

Table 4.2 – Summary of atmosphere parameters used in the point-mass aircraft model.

Furthermore, we modeled the wind field as a sigmoid function, uniform in x and y, but varying with altitude from zero wind, to a maximum windspeed w_{max}, blowing only in the W_x direction. The wind is further parameterized by z_0 and z_f, indicating

the beginning and end of the wind gradient transition region in meters. The wind field and its gradient can be calculated by:

$$W_x(z) = \frac{w_{max}}{1 + e^{b(a-z)}} \qquad \text{Equation 4.6}$$

$$\frac{\partial W_x(z)}{\partial z} = \frac{b w_{max} e^{b(a-z)}}{(e^{ab} + e^{bz})^2}$$

Where $a = (z_0 + z_f)/2$ is the center of the transition region, and $b = 14/(z_f - z_0)$ is a scale factor. An example wind profile is shown in Figure 4.1.

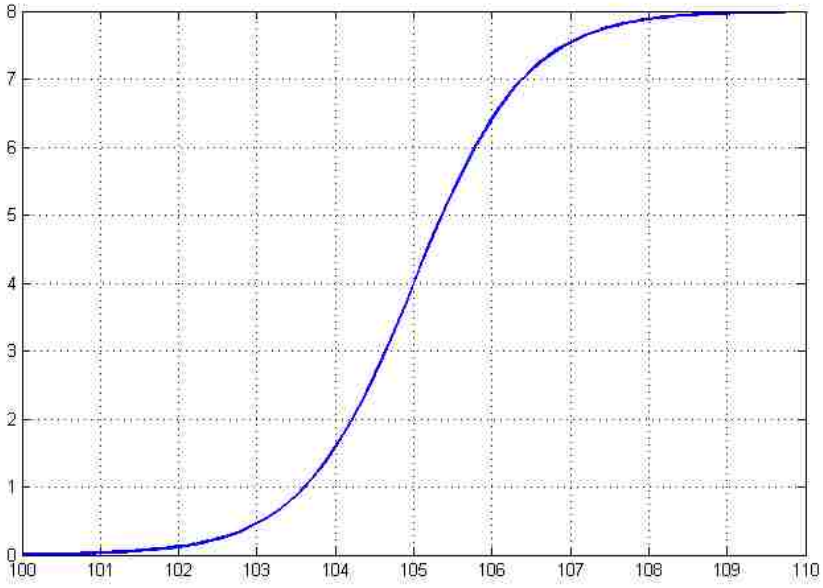


Figure 4.1 – An example wind profile. The parameters for this profile are $w_{max} = 8\text{m/s}$ and $\Delta z = 10\text{m}$. The boundary layer is between 100 and 110 m altitude. Below 100 m the air is still ($w = 0\text{m/s}$) and above 110 m the wind blows at w_{max}. The aircraft gains kinetic energy as it transitions through the boundary layer.

4.5. Global Planning

The purpose of the global planner is to find valid dynamic soaring trajectories. A valid trajectory is specified by a series of waypoints, indicating transitions between states that when executed, allow the aircraft to gain airspeed. One such trajectory is illustrated below, with various states of the aircraft noted along the trajectory.

We can find such trajectories using the collocation method for estimating the solution to differential equations, proposed by Zhao [51]. We break the trajectory into discrete points (colocation points) at equal time intervals along the track. Each colocation point represents the full state of the aircraft at that time in the trajectory.

$$s_t = [v_t, x_t, y_t, z_t, \psi_t, \gamma_t \mu_t, C_{Lt}] \qquad \text{Equation 4.7}$$

Each state $s_t$is also associated with an action a_t, that when executed produces state s_{t+1}.

$$a_t = [\gamma_{t+1}, \mu_{t+1}] \qquad \text{Equation 4.8}$$

Therefore, the equations of motion governing the movement of the aircraft, $\dot{s} = f(s, a)$ must be obeyed at every colocation point. Since we have discretized the trajectory, we assume the equations of motion behave quadratically between colocation points. We know:

$$s_{t+1} = s_t + \int_t^{t+1} f(s, a)dt \qquad \text{Equation 4.9}$$

$$s_{t+1} = s_t + \frac{dt}{6}\left(f_{t+1} + 4f_{m,t} + f_t\right)$$

Where $f_{m,t}$ is the midpoint between the two colocation points. Since we assumed f behaves cubically between colocation points, we can find $s_{m,t}$

$$s_{m,t} = \frac{1}{2}(s_{t+1} + s_t) + \frac{dt}{8}(f_t - f_{t+1}) \qquad \text{Equation 4.10}$$

Optimization problem

With the dynamics of the trajectory defined, we can use a nonlinear solver to enforce these dynamics. With the appropriate objective function, the solver will produce trajectories that enable various patterns of dynamic soaring. For this work, we are interested in the loiter pattern, where the trajectory resembles an orbit around a fixed point. The requirements for this pattern are that the flight is *periodic*, that is, the aircraft post at the beginning and end of the trajectory is the same, but with an increased airspeed. We put requirements on the periodicity of the trajectory with the following linear equality constraints

$\mu_f = \mu_i$	Final roll must equal starting roll
$\gamma_f = \gamma_i$	Final pitch must equal starting pitch
$\psi_f = \psi_i + 2\pi$	Final yaw must equal starting yaw plus one revolution
$x_f = x_i$	Final x must be starting x
$y_f = y_i$	Final y must be starting z
$z_f = z_i$	Final z must be starting z

Table 4.3 – Linear equality constraints of the dynamic soaring trajectory solver

These constraints ensure the trajectory makes a full orbit and the aircraft arrives exactly from where it left. For a dynamic soaring task, our objective is to gain the most energy after one complete orbit. By constraining the solution to start and stop at the same altitude, we only need to consider kinetic energy gains, which will result from an increased airspeed. Thus, our objective function is

$$max \frac{1}{2} m\left(V_f^2 - V_i^2\right) \qquad \text{Equation 4.11}$$

Furthermore, additional nonlinear inequality constraints were added to increase the smoothness of the solution. These rates were designed to mirror the rate limits on the Piccolo autopilot, which if violated can cause the tracking filters in the autopilot to diverge. We found that without these constraints, the nonlinear solver exploits freedom in these dimensions to arrive at solutions that appear to soar but are ultimately not feasible on real hardware.

$\lvert\dot{\mu}\rvert < \dot{\mu}_{max}$	Limit roll rate
$\lvert\ddot{\mu}\rvert < \ddot{\mu}_{max}$	Limit roll acceleration
$\lvert\dot{\gamma}\rvert < \dot{\gamma}_{max}$	Limit pitch rate
$\lvert\ddot{\gamma}\rvert < \ddot{\gamma}_{max}$	Limit pitch acceleration
$\lvert\dot{C_L}\rvert < \dot{C}_{L_{max}}$	Limit C_L rate
$\lvert\ddot{C_L}\rvert < \ddot{C}_{L_{max}}$	Limit C_L acceleration

Table 4.4 – Nonlinear inequality constraints of the dynamic soaring trajectory solver

Initialization

Possible solutions can be found by initializing each colocation point to a random value. However, a reasonable guess for the starting value of each point leads to considerably

faster convergence times. Here, we present the initial point used in our tests. This point approximates a valid dynamic soaring trajectory, which helps the solver converge.

$$v_t = 5cos(2\pi t + 5 + v_{min})$$

$$\psi_t = -2\pi/(t-1) - 2\pi$$

$$\gamma_t = \frac{25\pi}{180} sin(2\pi t)$$

$$C_{L,t} = .1$$

$$z_t = \frac{(z_f - z_0)}{2}\big(1 - cos(2\pi t)\big) + z_0$$

$$x_t = -20\big(1 - sin(2\pi t)\big)$$

$$y_t = -20\big(1 - cos(2\pi t)\big)$$

This seed trajectory is shown in Figure 4.2:

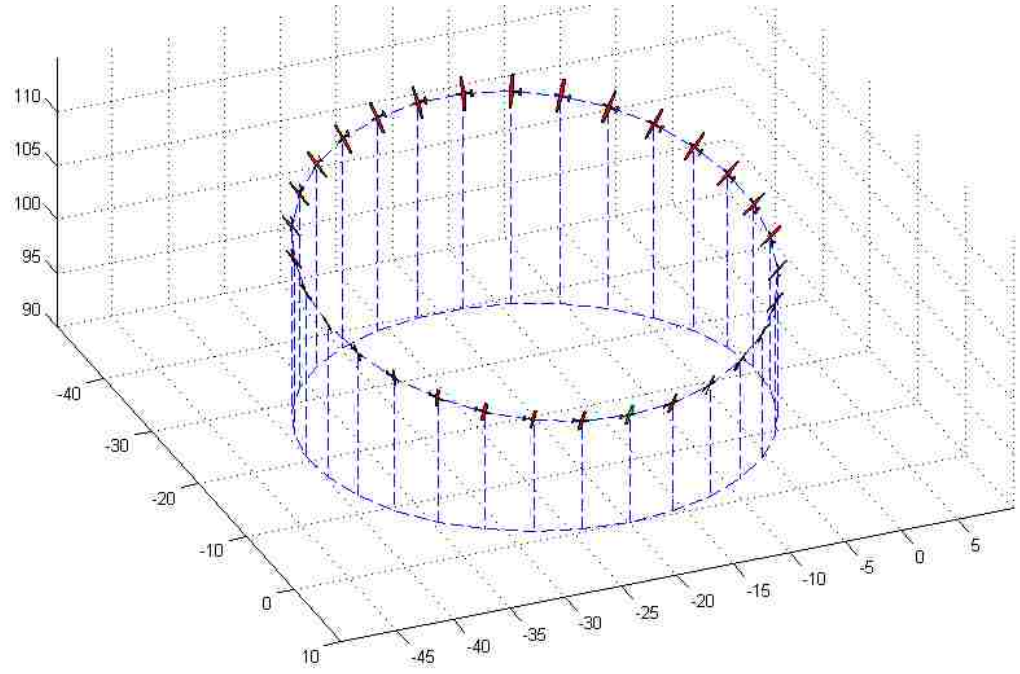


Figure 4.2 – The seed trajectory used to initialize the planner. This trajectory is just a banked orbit with a constant roll. Using a seed trajectory of this form speeds up the solution process significantly.

Solution

An example of the solver output is shown below in Figure 4.3, along with several states of the glider visualized. The first state is shown in green, at the orbit's minimum altitude. It continues along in a counter-clockwise direction, crossing the boundary

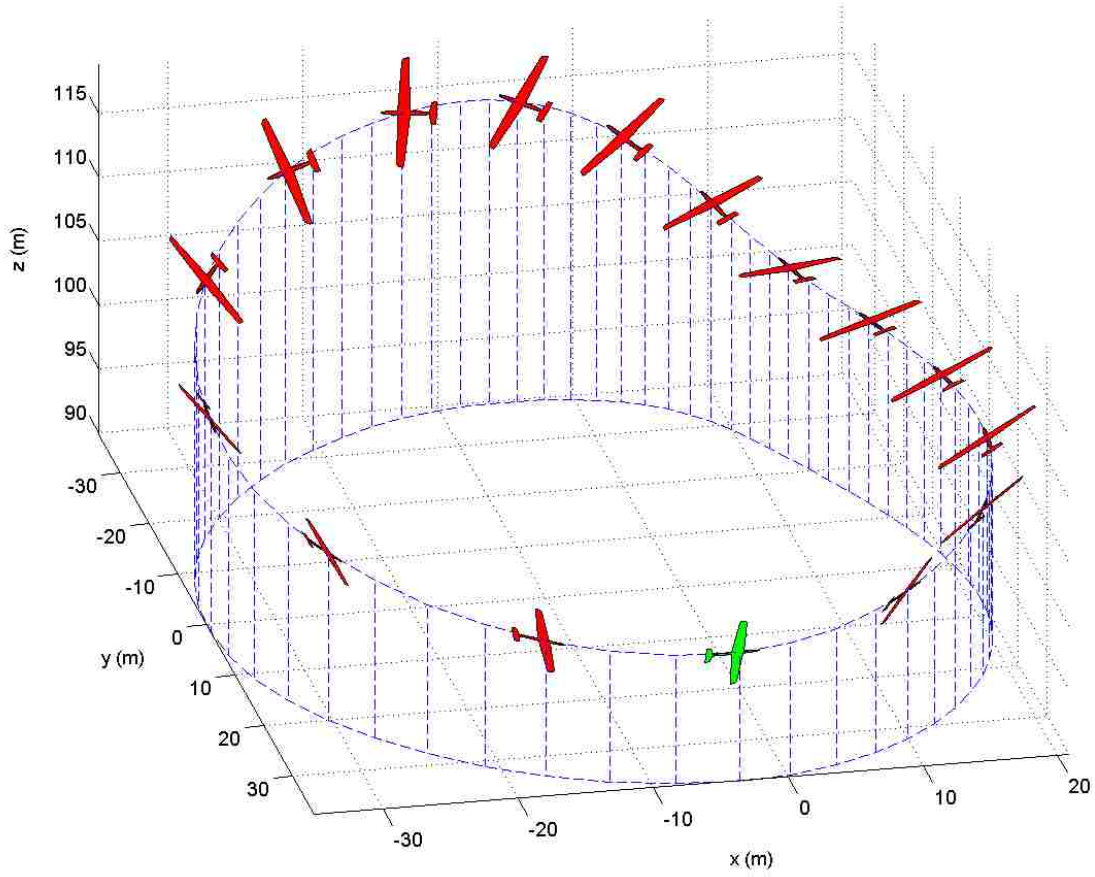
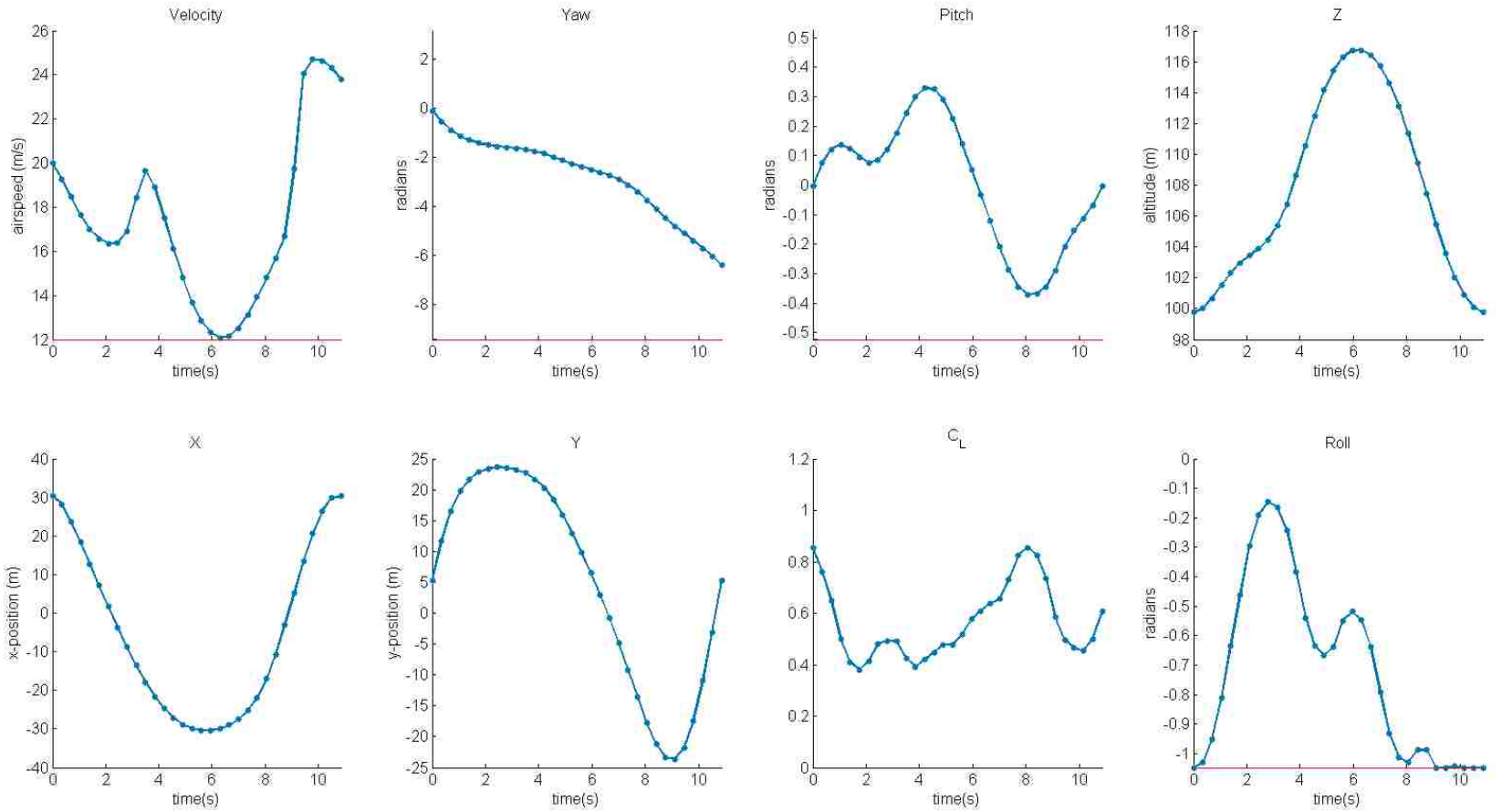


Figure 4.3 – An example loitering dynamic soaring trajectory. The aircraft, drawn to scale, is visualized at several points around the trajectory. The starting point of the aircraft is marked in green. A wind gradient is blowing in the positive y direction at 8m/s at 110 m altitude, and 0 m/s at 100 m altitude.

layer at 100m through 110m. The wind in this simulation is blowing from the -y direction. Unlike the seed, the solution does not have a constant bank. We can see the glider roll angle nearly levels out as it climbs through the boundary layer. After it

reaches 110m, it continues to climb until it nears the stall speed of 12 m/s. Then it banks harder and begins to dive through the boundary layer. It gains airspeed by converting potential to kinetic energy through the dive and gains additional airspeed as it moves through the boundary layer. This is the mechanism by which energy is extracted from the wind. The glider maintains its roll through the boundary layer to arrive at its original starting location. When it arrives, it has the same position and attitude as when it began, but the airspeed has increased, netting the aircraft an energy gain.

On the next page, each dimension of the state of the aircraft is visualized over the course of one orbit, with colocation points depicted as dots. The blue lines connecting the dots are for illustrative purposes only:


Velocity
airspeed (m/s)
time(s)
Yaw
radians
time(s)
Pitch
radians
time(s)
Z
altitude (m)
time(s)
X
x-position (m)
time(s)
Y
y-position (m)
time(s)
C_L
time(s)
Roll
radians
time(s)

Generating Trajectory Sets

Even on state-of-the-art desktop hardware, a dynamic soaring trajectory can take upwards of a minute to solve starting from a generic seed. Even from a bootstrap trajectory, a solution can take several seconds to converge. A soaring aircraft waiting on a trajectory to finish converging will not be able to react to in-situ re-planning of the global plan every cycle with this kind of latency. Therefore, we trade time for space by precomputing trajectories for a range of expected wind shears and air speeds. These trajectories are stored in an in-memory database indexed by maximum wind speed, height of the boundary layer, and speed of the aircraft at the lowest point in the trajectory. In the Figure 4.5 below, two trajectory sets are shown. The red trajectory is the same in each. The set on the left varies over a range of initial velocities and holds the other indices constant. The set on the right varies over maximum wind strength and holds the other indices constant.

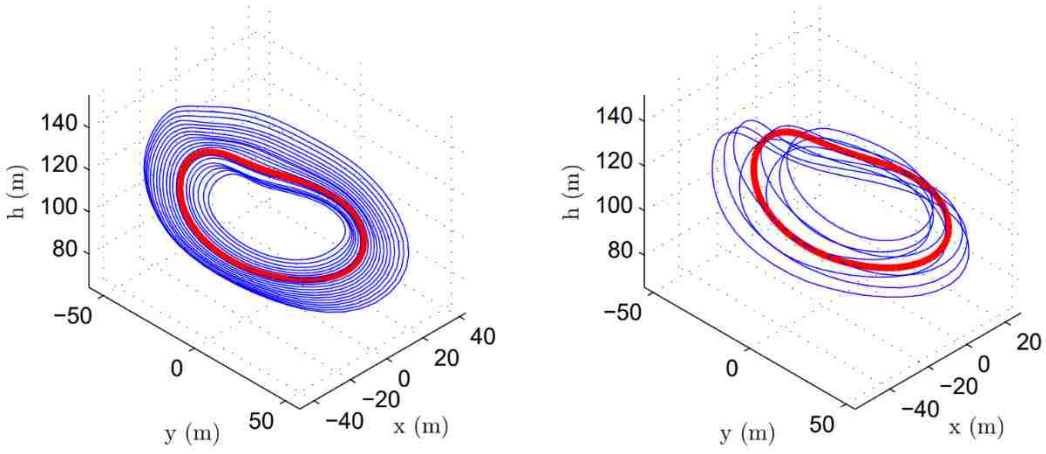


Figure 4.5 – Two sets of similar dynamic soaring trajectories. (left) a set of trajectories that hold w_{max} and the height of the boundary layer constant, while varying the initial airspeed of the aircraft. (right) a set of trajectories that hold the initial airspeed of the aircraft and the height of the boundary layer constant, while varying w_{max}. The red trajectory in each of the figures has the same parameters of 15m/s initial airspeed and 8 m/s w_{max}.

Global Planner

The global planner is responsible for selecting the next best trajectory for the soaring aircraft. As the glider completes each loop, its kinetic energy increases, making the previous global plan unsuited for the next cycle. If the same plan were followed, the aircraft would not extract as much energy as it could with a newly generated plan. Moreover, a low-energy trajectory cannot be followed by an aircraft at a higher energy state without violating constraints on lift, roll rate, and pitch rate. The ideal planner would generate a new plan from scratch to match the specific situation of the aircraft and environment. But this planning processes takes too long, and the aircraft cannot wait for it to finish. We've found that discretizing the index space at 1 m/s intervals for windspeed, 1 m over a 10 m range for boundary layer height, and .1 m/s intervals for airspeed allows for a set of global plans that covers the expected conditions and dynamics encountered by the aircraft during a DS mission.

While we initially tried planning for the next cycle at the bottom of the DS cycle, we found this to be problematic. While the cycles are planned as closed loops, this is not how the aircraft actually performs; it is easier for the aircraft to transition to a new plan when the aircraft is in the diving phase of the cycle. This way, it can regulate its turn to slowly merge into the new trajectory. This results in an outward spiral motion as the glider converges to a steady state and is depicted in the experiments section in Figures 4.8 – 4.13.

4.6. Local Planning

Trajectory Rollout [61], is a sample-based algorithm on the robot's control space, which allows us to not only easily encode complex goals into the robot's performance,

but to guarantee the robot operates within a specified performance envelope as well. The algorithm works by integrating sampled controls forward in time using the robot's motion model, derived in Section 4.3, resulting in a set of of k candidate trajectories. The trajectory with the lowest cost is chosen and its associated control is executed.

Overview

Trajectory Rollout takes as input the current pose of the robot, s_0, and a cost map M. The map can take the form of an occupancy grid, where each cell is marked as either occupied (if it contains an obstruction), or scored according to a cost function (more on this later).

The algorithm begins by sampling K control inputs from the control space of the robot

$$U = \{u_1, u_2, \ldots, u_K\} \qquad \text{Equation 4.13}$$

where u_k is the k^{th} sampled control. These can be chosen according to the specific robot domain. One opportunity presented here is to only sample control inputs that fall within the performance constarints of the robot. For instance, a robot car should obey the speed limit. Sampling velocities that do not exceed the posted limit ensures the robot will obey this restriction.

Next, for each control we integrate the state forward. Each sampled control is held constant as we integrate the initial robot pose forward in time using the robot's motion model, ending at some specified time horizon t_f. This results in K trajectories emanating from the current pose, indicating how the robot would move if the associated control input were applied over the time horizon. The rollouts in Eq. 4.14 are depicted schematically in Figure 4.6.

$$T_{u_k} = \left\{s_0, s_{t+1,u_k}, s_{t+1,u_k}, \dots s_{t_f,u_k}\right\} \qquad \text{Equation 4.14}$$

where s_{t+1,u_k} is the integrated pose of the robot at time $t+1$ under the control u_k. Once we have our candidate trajectories, we score each one using a cost function that evaluates how well a trajectory conforms to stated mission objectives. For instance, a cost function might assign a prohibitive cost to a trajectory that veers too close to an obstacle, and a low cost to one that moves the robot closer to a goal location. This function is evaluated in the context of a map M, which encodes the robot's objectives.

$$C\left(T_{u_i}, M\right) = w_1 C_1 + w_2 C_2 + \cdots + w_n C_n \qquad \text{Equation 4.15}$$

In this case the cost function is a linear combination of the various weighted costs, but there is flexibility on the part of the algorithm implementer in choosing exactly how these costs are assigned. For the dynamic soaring problem, the lowest cost trajectory

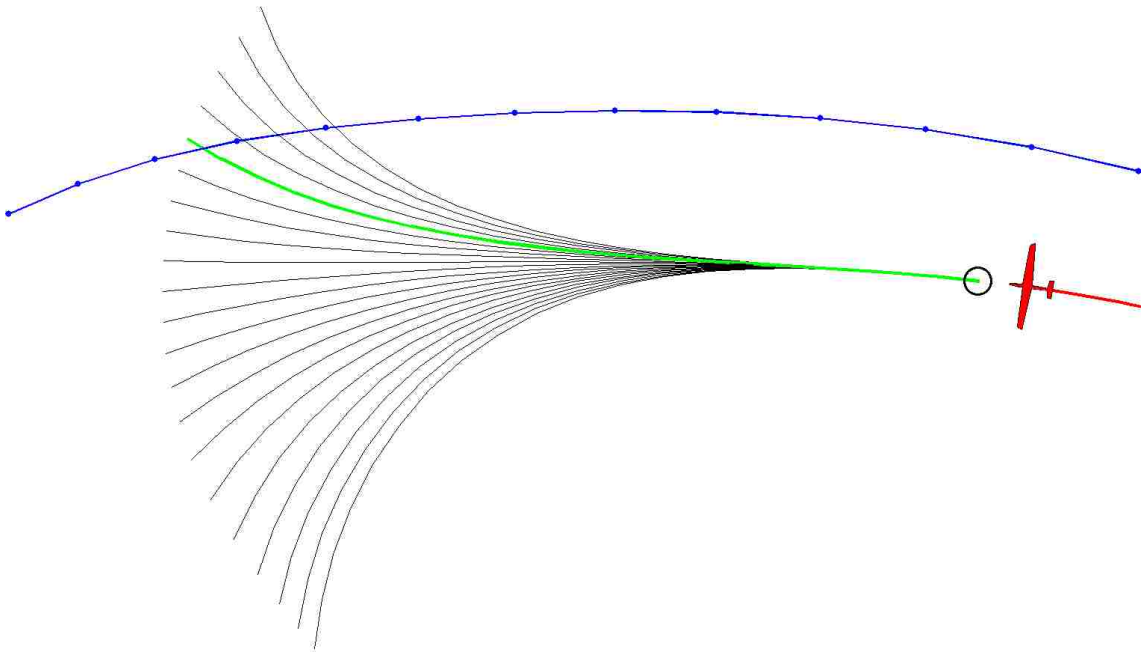


Figure 4.6 – A schematic of the local planning algorithm in action. Several trajectories are shown in black, with the lowest-cost trajectory highlighted in green, since the position is closest to the global plan, shown in blue.

is the one where the endpoint is closest to the global plan, where the distance is defined as the Euclidean distance of the pose of the aircraft with the pose of the closest waypoint.

Finally, we select the trajectory with the lowest cost (the green trajectory in Fig. 4.6) and use its associated control as input to the robot plant. The algorithm then repeats, where the next pose is taken as the input to the next round of Trajectory Rollout.

It is important to note that the selected control input is not implemented for the entire time horizon, but for only one cycle of the algorithm (*e.g.* if Trajectory Rollout runs at 10Hz, the selected control is held for 1/10 seconds), so the robot does not necessarily follow any one of the trajectories generated. By continually re-evaluating feasible trajectories, the robot can perform more complex behaviors than any single trajectory would suggest. The full trajectory rollout algorithm is presented below:

Figure 4.7 – Trajectory Rollout Algorithm

Trajectory Rollout

Input: s_{t-1}, M, t_f, K

Output: u_t

sample control space: $U = \{u_1, u_2, \ldots, u_K\}$

for each $k \in K$

for each $t = 0 \ldots t_f$

$s_t = g(s_{t-1}, u_k)$

$T_{u_k} = T_{u_k} \cup s_i$

$c_k = C(T_{u_k}, M) \; \forall k \in K$

$j = argmin(c_k)1$

return $u_t = U_j$

4.7. Experiments

In this section we present results of experiments of the dynamic soaring system across several different domains. The first domain is a point-mass simulation of the Fox platform. It models lift and drag and ballistic motion but does not model vehicle dynamics. The second is a high-fidelity simulator provided by Cloud Cap Technology that does model vehicle dynamics. Finally, we present result from real-world field testing. These last results are not of the dynamic soaring task, but of a simplified tilted orbit following task.

4.7.1. Point Mass Simulations

Point mass simulations were conducted of the dynamic soaring system. Trajectories were generated for w_{max} = 10m/s, Δz = 10m, and v_0=15-40m/s. This trajectory database was provided to the global planner. The simulated aircraft was placed at the bottom of the boundary layer facing the positive x-direction at 20m/s airspeed, and the simulation was started. The boundary layer was placed between 465m and 475m altitude. The (x, y) position of the aircraft in Figures 4.8 – 4.10 are depicted in UTM coordinates located at State College, Pennsylvania. The first trajectory selected by the planner was ($w_{max,}$ Δz, v_0) = (10m/s,10m,20m/s), followed by airspeeds of 22m/s, 24m/s, 26m/s, and 27m/s. The aircraft reached a steady state at this trajectory, and achieved a maximum airspeed of 34m/s. The path of twenty cycles of the aircraft over this mission is depicted in Figures 4.8 – 4.10. Airspeed is shown in Figure 4.11, and altitude is shown in Figure 4.12. Total energy (kinetic + potential) from the frame of reference of the aircraft is shown in Figure 4.13.

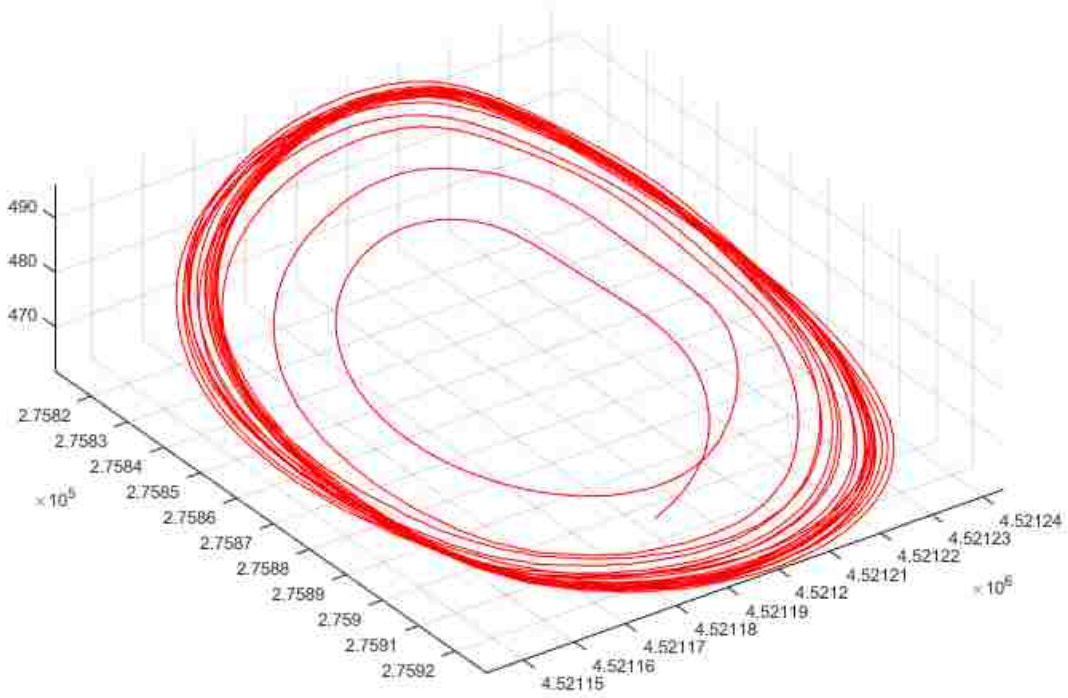


Figure 4.8 – Point mass simulation over 20 DS cycles. Wind in this figure blows from the negative Y direction.

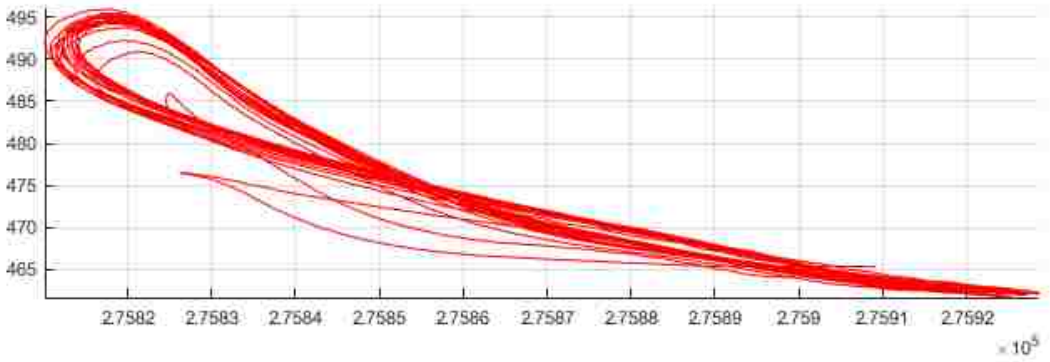


Figure 4.9 – A side view of the path of the aircraft, looking toward the YZ plane. Wind is blowing from left to right in this figure.

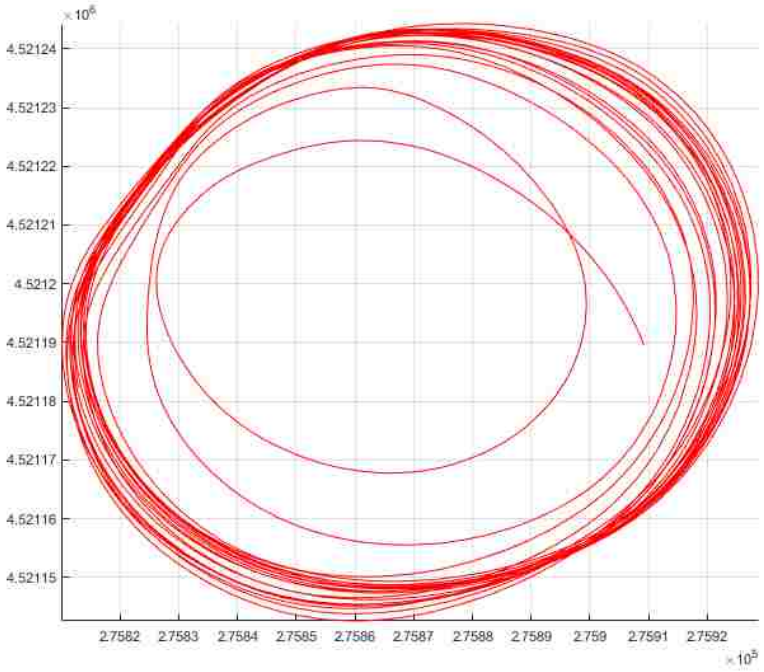


Figure 4.10 – A top view of the point mass simulation, looking at the XY plane. Wind is blowing from left to right in this figure.

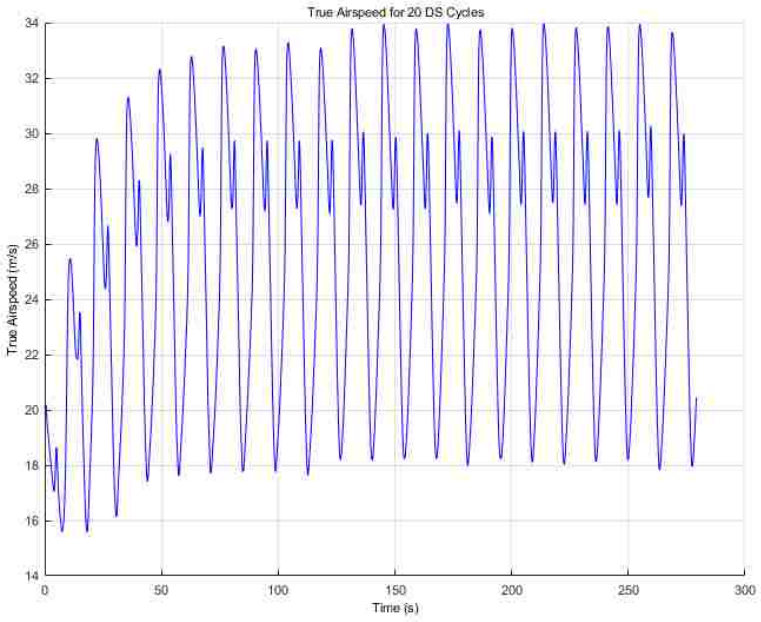


Figure 4.11 – True airspeed of the glider over 20 cycles. The maximum steady state airspeed reached is 34 m/s. The initial aircraft airspeed is 20 m/s.

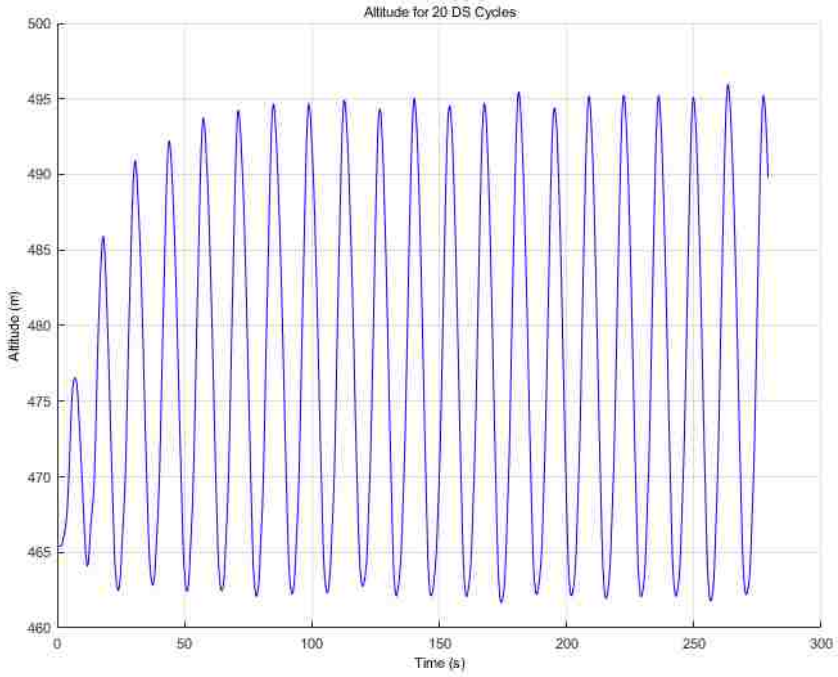


Figure 4.12– Altitude of the glider over 20 simulated loops

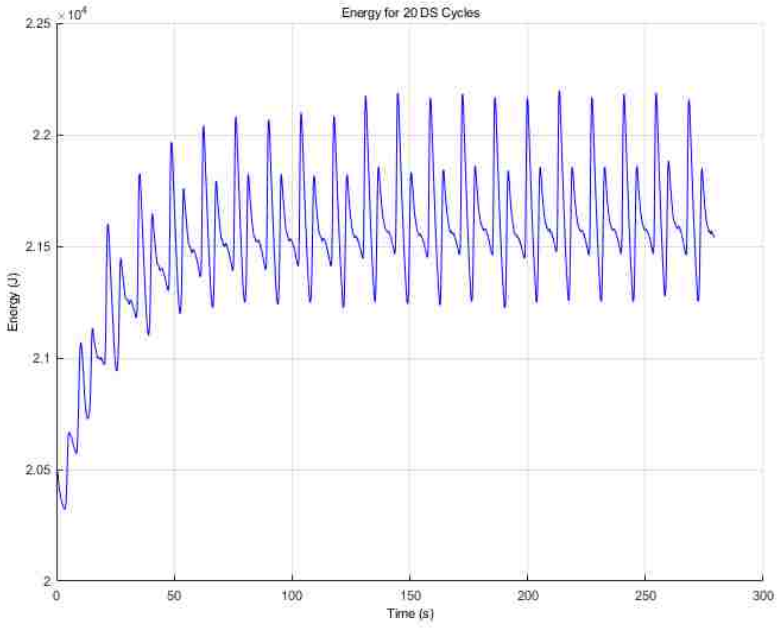


Figure 4.13 – Total energy of the glider, defined as kinetic energy plus potential energy due to gravity, over 20 simulated loops.

4.7.2. Hardware-In-Loop Simulations

Hardware-in-loop simulations were conducted with the Piccolo SL autopilot and Cloud Cap Technology's simulation software through the Piccolo Control Center. This simulator uses a high-fidelity model of the aircraft that models aircraft dynamics, momentum, and other elements beyond what the point mass model simulates. The simulator works through the autopilot and uses the actual hardware in the simulation loop. Control signals are generated by the dynamic soaring controller and are sent to the autopilot, which generates rate targets that are sent to the simulator. The simulator executes those targets on the aircraft model and generates simulated telemetry, which is sent to the dynamic soaring controller, closing the control loop. The simulated model uses the autopilot gains that the actual aircraft uses during field testing, to try and make the simulation as close to an actual flight.

For this test, the aircraft was directed to fly a level orbit below a w_{max} = 10m/s, Δz = 10m boundary layer. Stead state results are shown in Figures 4.14 – 4.15. While the aircraft was able to reach a steady state, it did not track the target trajectory well at the lower end of the cycle, soaring wider than the target trajectory prescribed. This was a result of latency in executing commands, as well as the autopilot not baking as aggressively as in the point mass model. Aggressive banking caused divergence in the autopilot's AHRS filter, so we used more leisurely rates to avoid this scenario. Despite these tracking errors, a steady state dynamic soaring orbit was reached.

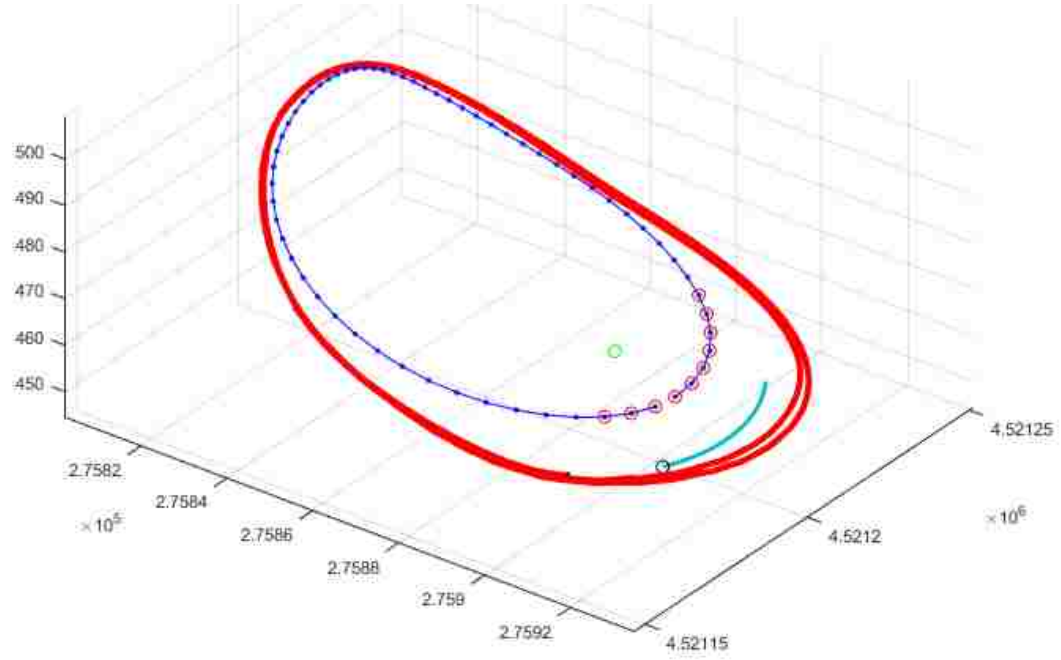


Figure 4.14 – Hardware in loop simulation results at a steady state. The blue line represents the target trajectory, the red line represents the flown steady state trajectory, and the green line shows a planner rollout.

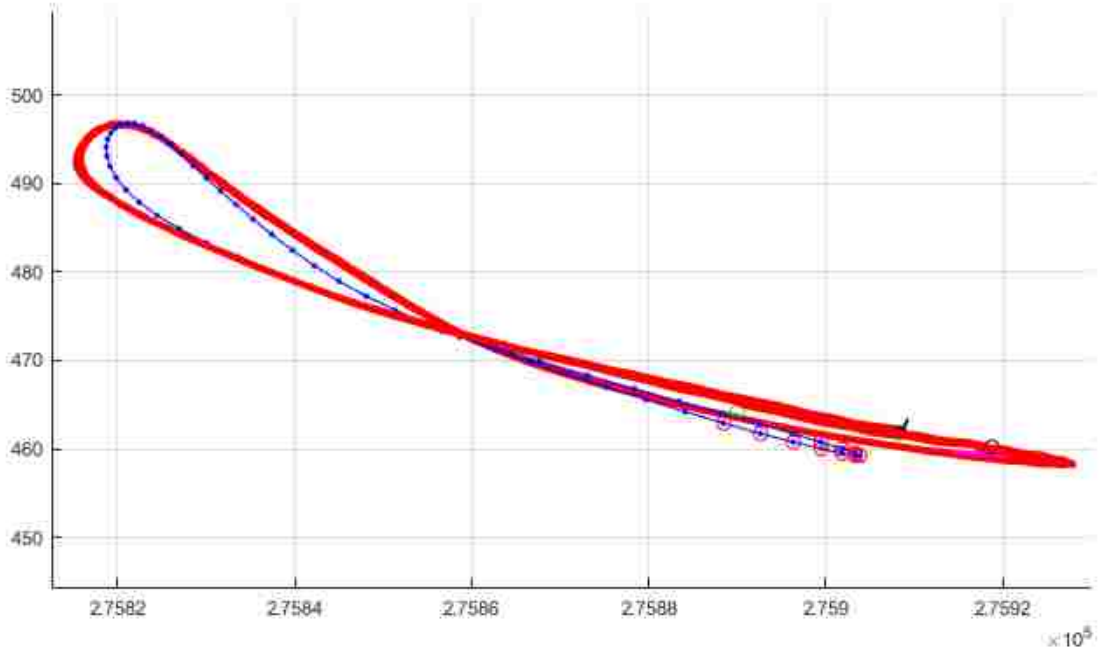


Figure 4.15 – A side view of the hardware in loop simulation. The glider reaches the target maximum altitude and minimum altitude, but glides wide on the low-end turn.

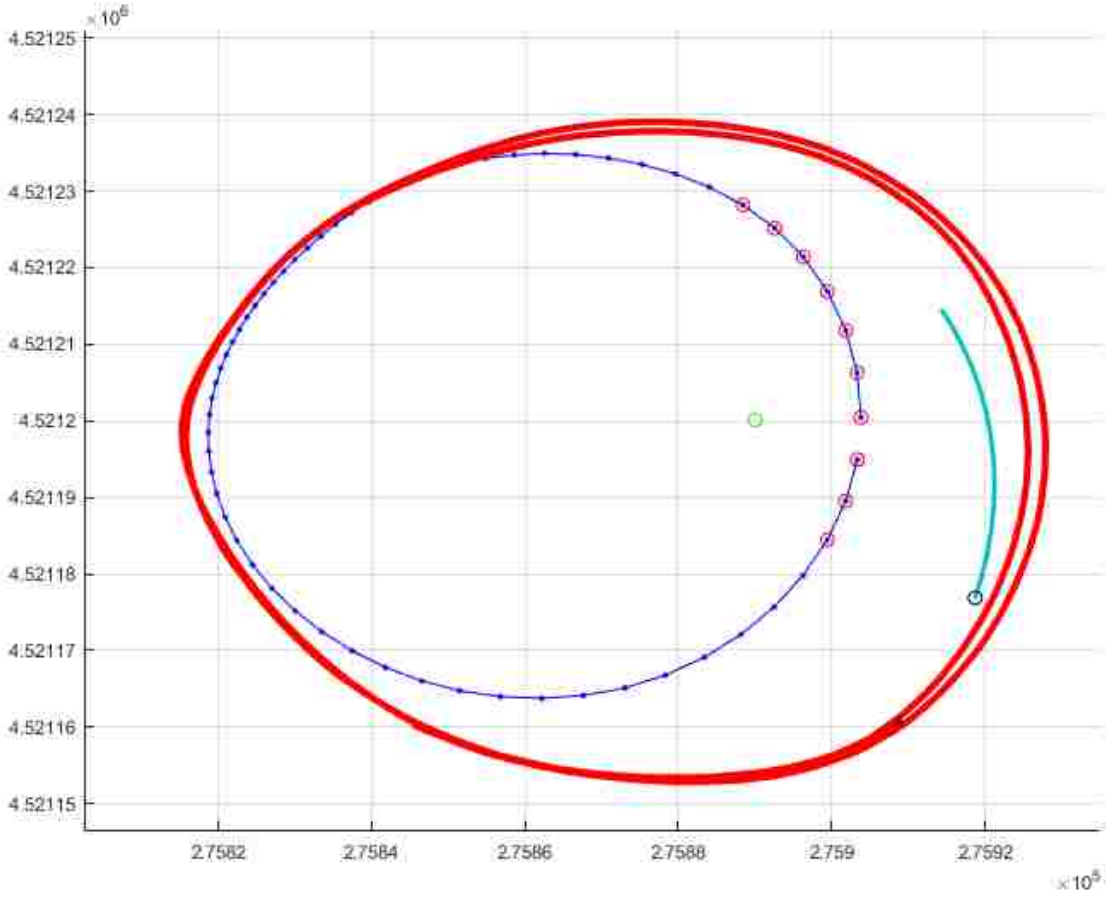


Figure 4.16 – A top view of the hardware in loop simulation.

4.7.3. Field Tests

Field tests were performed using the Fox soaring platform described in Chapter 3. The purpose of these tests was to assess the ability of the sample-based planner to follow an orbit, and for the global planner to select the next best trajectory. This field test did not perform a dynamic soaring maneuver, but instead followed level orbits. A second test more closely approximates a dynamic soaring maneuver by following tilted

orbits. However, without the presence of a wind gradient, which was not available at the time of the tests, the dynamic soaring maneuver could not be performed.

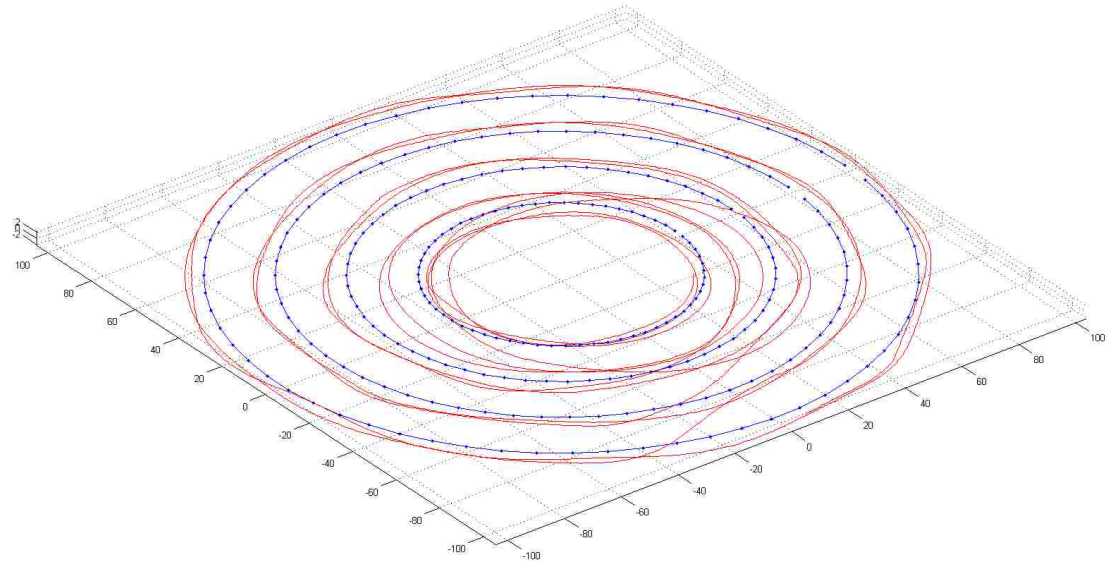


Figure 4.17 – Results for the concentric level orbit following. Blue lines represent the target trajectory, red line represent the flown path. The glider was commanded to follow 4 concentric orbits of varying radii from 40m to 100m. Transitions between orbits can be seen in the lower portion of the figure.

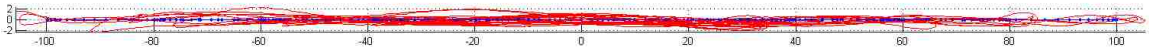


Figure 4.18 – A horizontal view of the orbit following results, looking at the YZ plane. Deviation from the level orbits did not exceed 2 meters in either direction.

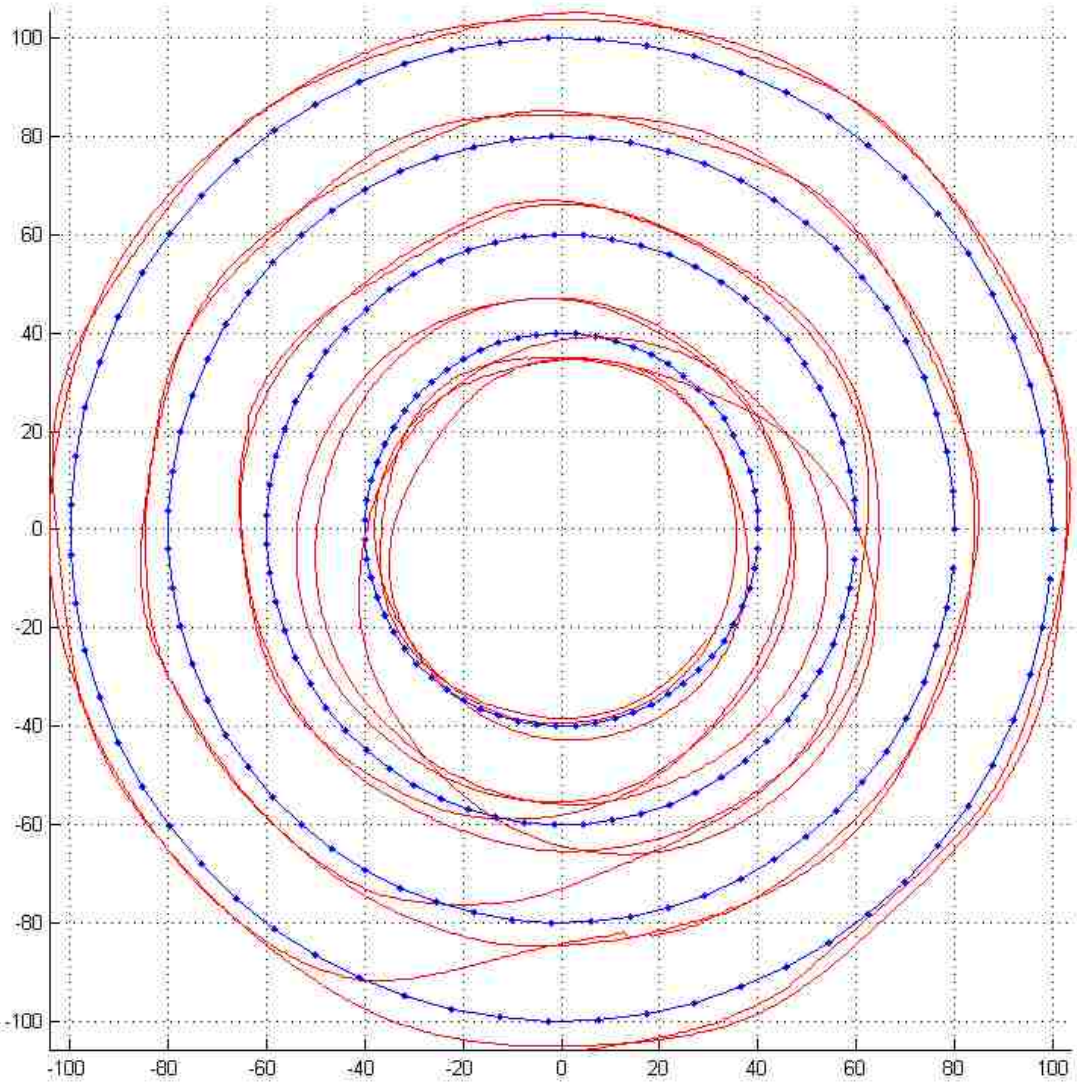


Figure 4.19 – A top view of the orbit following results, looking at the XY plane. A constant offset from the target track is noted, due to the lookahead setting on the sample-based controller.

A second test was carried out to more closely resemble a loitering dynamic soaring task. In this experiment, the aircraft was commanded to follow a level circular orbit of 80 m radius. After completing an orbit, the target trajectory was changed to a tilted orbit of 40 m radius. The tilt began at 0 degrees and was increased in 5-degree increments until a maximum tilt of 15 degrees was reached.

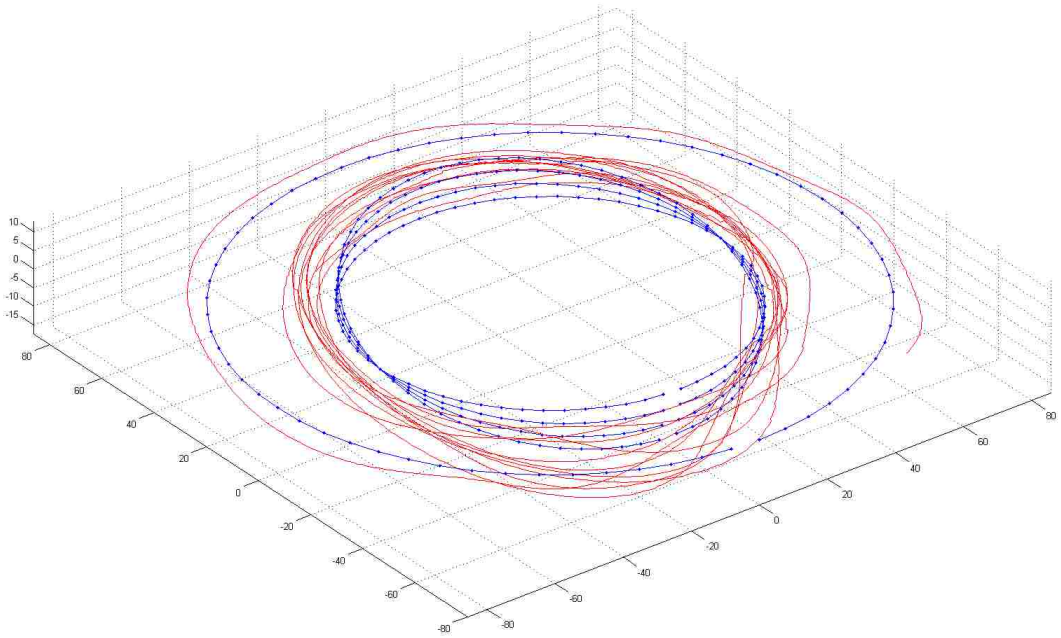


Figure 4.20 – Results of the field test for tilted orbit following.

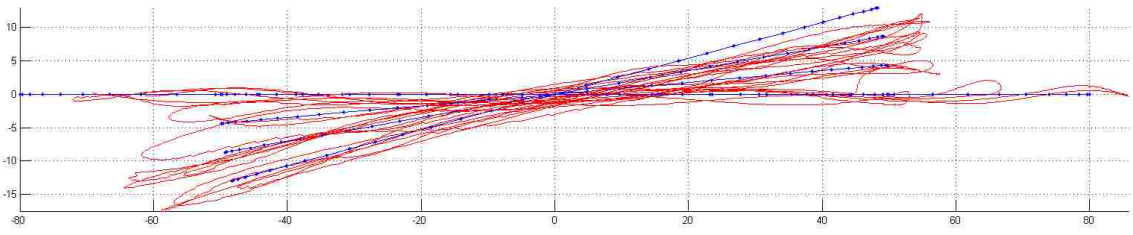


Figure 4.21 – Side view of the tilted orbit test, looking at the YZ plane. The glider did not quite reach the maximum altitude on the highest tilted orbit. This was due to a lack of power in the motor, which was fixed in a subsequent platform revision.

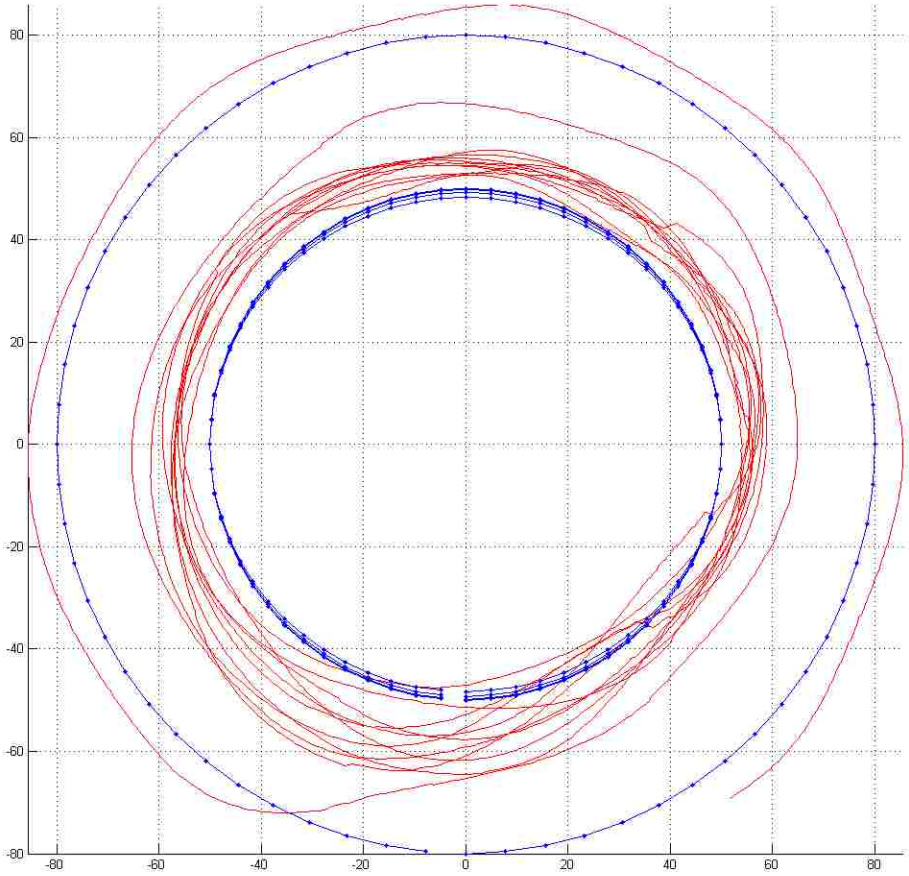


Figure 4.22 – Top view of the tilted orbit test, looking at the XY plane.

4.8. Summary and Discussion

In this section we presented a controller for a dynamic soaring task. The controller consisted of a global planner that generates dynamic soaring trajectories that are followed by a local controller. The results of several tests were presented to assess the efficacy of this system. Point mass model simulations showed the sample-based controller can follow dynamic soaring trajectories to a steady state and soar indefinitely. Hardware-in-loop simulations showed that the controller was sufficient

for executing a dynamic soaring task on the simulated Fox model. However, tracking errors were observed due to latency in issuing commands to the hardware. This was partially accounted for using a latency queue in issued commands, but some tracking error still persisted. To compensate for these errors, we began work on adaptive tuning of controller parameters such as lookahead distance and control sample size, but we soon began to realize our time at adaptive control might be better spent directed at a completely different control paradigm. This led to the ideas presented in Chapter 5.

Chapter 5

Learned Controller

Sutton *et al.* recognized two categories of control problems: 1) tracking problems in which a reference trajectory is followed and 2) optimal control problems in which an objective function is maximized [62]. They classified reinforcement learning as being direct (in that the system rather than a model is controlled) adaptive (in that the controller responds to changes in the environment) optimal controllers. With this distinction in mind, we take a moment now to step back and consider the dynamic soaring problem anew. The literature on dynamic soaring control and the controller presented in Chapter 4 treat dynamic soaring as a tracking control problem. The reason for this formulation is that because solving for dynamic soaring trajectories takes a great amount of computational power (and time), we solve them ahead of time and use a tracking controller to follow one. Thus, if the controller tracks the DS trajectory well enough, it should perform a DS maneuver.

But this method indirectly solves the problem of DS control. Dynamic soaring is more directly expressed as an optimal control problem; we don't actually care to follow any specific trajectory as long as the executed maneuver nets an energy gain after one cycle. In this section we re-formulate the dynamic soaring problem using a

reinforcement learning framework. As we will show, formulating the control problem this way greatly simplifies the architecture of the software control layer, and even out-performs the controller presented in Chapter 4 in energy-gained per cycle.

5.1. Related Work

Reinforcement Learning in general is covered in Chapter 2. The literature regarding soaring and reinforcement learning dates back to 1998, when Wharington presented his Ph.D. thesis that studied a variety of soaring problems in the context of reinforcement learning [24]. He looked at dolphin soaring, thermal soaring, open-loop dynamic soaring, but stopped short of closed-loop dynamic soaring. Lawrance and Sukkarieh investigated reinforcement learning for thermal soaring, looking at both the exploration-exploitation tradeoff [63], and path planning in dynamic wind fields while exploring them [64]. Chung *et al.* explored the use of Gaussian processes for estimating the Q-Function in Q-Learning for a variety of problems, including soaring [65], and further showed that their choice of objective function resulted in an agent that learned to switch between exploration and exploitation of thermal wind depending on its current energy reserves [66]. Woodbury *et al.* used reinforcement learning to control bank angles for thermal soaring [67]. Reddy *et al.* combined numerical methods with RL to learn strategies that cope with soaring in turbulent winds [68]. Lecarpentier *et al.* demonstrate Q-Learning for thermal soaring in much the same way as the presentation in this chapter, focusing on the elements of the framework that need to be adapted for the thermal soaring domain [69]. The most complete and recent treatment of reinforcement learning for soaring was done by Reddy *et al.* in which they deploy a learning thermal soaring glider in the field [70].

Finally, deep reinforcement learning was used in a 2D gliding and perching simulation by Novati *et al.* [71].

5.2. Chapter Contributions

In this chapter we use reinforcement learning as a tool to approach the dynamic soaring control problem as an *optimal* control problem rather than a tracking problem. The literature largely concerns reinforcement learning applied to bank control and exploration for thermal soaring. By contrast, we formulate reinforcement learning for a dynamic soaring loiter mission. Our formulation results in three contributions. First, we analytically identify a reduced state-action space using trajectories generated using the method in Chapter 4. Next, we identify a method for function approximation under the constraints of on-line learning with limited computational resources of the soaring platform and lack of training samples for the soaring problem domain; and handle maintenance of that method considering potentially long-term mission durations. Finally, we present experimental results that demonstrate the optimal control formulation of the dynamic soaring problem out-performs the tracking control formulation presented in Chapter 4.

5.3. Problem Formulation

The learned planner is built in two phases, as described in [72]. In the first phase, the learning planner passively observes the local planner follow a DS trajectory. This is called the "bootstrap" phase, where the learned planner builds as basis on which to act in the future, provided by the local planner.

In the second phase, the local planner is discarded, and the learned planner can execute the learned policy. In this section, we'll detail the learned planner.

As an inherently high dimensional, continuous state space problem, DS is not a drop-in candidate for Q-Learning; there are various implementation details needed to fit Q-Learning for our domain, which were inspired by [73]. First, we discuss our state space formulation. Then, we detail how we used a function approximator to adapt Q-Learning for a continuous state space. Finally, we detail next state selection and our reward function.

5.3.1. State / Action Space

An intuitive space to use for the learner is the position and attitude of the aircraft

$$s_t = [v_t, x_t, y_t, z_t, \psi_t, \gamma_t \mu_t, w_{max}] \qquad \text{Equation 5.1}$$

$$a_t = [\gamma_{t+1}, \mu_{t+1}] \qquad \text{Equation 5.2}$$

Where v_t is the airspeed; x_t, y_t and z_t are Cartesian coordinates relative to the loiter point; and ψ_t, γ_t, and μ_t are the yaw, pitch and roll. The action space consists of the commanded pitch and roll angles for the next state. However, this means the Q-function will have 9 dimensions, which is undesirable. Using a conventional controller (described in Chapter 4) we collected the states and actions for an entire trajectory to perform a statistical analysis on the data to try and reduce the dimensionality of the state space.

The most obvious dimensions to eliminate from the state space are γ_t and μ_t; since our time step is small (0.1 s) these dimensions are highly correlated to the action space

dimensions γ_{t+1} and μ_{t+1} ($\rho > 0.99$ for each). We can further reduce the dimensionality by performing a principal component analysis (PCA) on the remaining state space dimensions. We find that the first principal component accounts for 93% of the variability in the data and is aligned along the ψ_t axis (with a coefficient value of 0.98). Next in order are x_t (5.5%),y_t (0.9%), v_t (0.2%), and lastly z_t (0.1%). Thus, by removing v_t and z_t we maintain 99.7% of the variability in the state space, for a final reduced-dimension state/action space:

$$x = [s_t, a_t]^\top = [x_t, y_t, \psi_t, \gamma_{t+1}, \mu_{t+1}]^\top \qquad \text{Equation 5.3}$$

5.3.2. Function Approximation

In continuous state spaces, as in the domain at hand, function approximation techniques such as neural networks are used to represent the Q-function instead of a matrix representation [33]. Unfortunately, the immense training time needed to create neural networks means the Q-function can only be learned offline after a robot has collected a large amount of experience. Further, since many neural networks cannot adapt to new data, the robot might not be able to learn incrementally. This is the heart of why a deep neural network is not appropriate for this problem – the ability to retrain the network in real time as experience is gained is crucial to this approach enabling dynamic soaring in the real world.

We use an approach with minimal training time that can be incrementally updated, with the trade-off of requiring a larger memory footprint. Our approach is inspired by generalized regression neural networks [73], which calculate the distances from an input vector to a set of training vectors to arrive at a classification. We implement our

function approximator as follows: Consider a query point x_q, which is a state/action vector. We want to determine its Q-value based on a set of N training vectors, X, and their associated Q-values Y. We first calculate the Euclidean distance d from x_q to each training vector $x \in X$. Then the K nearest neighbors are selected, and their distances are weighted using a radial basis function (RBF) of the form

$$w_k = e^{-(d_k/h)^2} \quad k \in K \qquad \text{Equation 5.4}$$

where the parameter h is the RBF bandwidth. We then multiply these weights by the Q-values corresponding to the K selected vectors and calculate the predicted Q-value

$$q_p = \frac{\sum_k^K Y_k w_k}{\sum_k^K w_k} \times \max_k w_k \qquad \text{Equation 5.5}$$

We multiply by the maximum weight to ensure that poorly supported vectors have a diminished Q-value. Without this, it is possible to extrapolate a q_p that is not properly justified by observed data. We further reduce this possibility by choosing a threshold \tau, such that if an observation's maximum weight is less than τ, q_p is set to zero. In our experiments, we chose $\tau = 0.6$.

Through using this technique, we are trading computation time for space. Therefore, as the number of neurons in the network grow, searching for the K-nearest neighbors will become computationally prohibitive. Therefore, we store X in a k-d tree [74] for fast searching and rebalance the tree when enough new neurons are added.

Searching still eventually becomes a bottleneck, as X grows unbounded. Therefore, we remove redundant points (*i.e.* closely clustered points have the same predictive power as a single point) using a leave-one-out optimization technique that aims to minimize

the predictive impact of the removed observation. Again, since this is an expensive procedure it is not performed every iteration, but only when X grows to a set capacity.

5.3.3. Selecting the Next Best Q

The maximization involved in Equation 5.5, while straight forward in the matrix case, is much more complicated in the continuous case – finding the next best action typically involves an optimization procedure that can be time-intensive. Instead, we sample controls around the current roll μ and pitch γ, since again we note that the time step between states is small, and thus our next state (after executing the selected roll and pitch command) will closely resemble the UAV's current attitude. We sample 20 values each from the ±5-degree regions surrounding the current μ and γ. We take the combination of these samples and the current μ and γ for a total of 441 action pairs. These actions are then concatenated with a vector of the current state and fed into the function approximator described in the previous section. We then simply take the maximum of the resulting predicted Q-values for use in the Q-function update. Accuracy can be increased by increasing the sample resolution.

5.3.4. Rewards

For loitering trajectories, the UAV earns zero reward at every state except the one where it completes a loop. The reward is the energy gained by the aircraft over that loop.

$$R_{t_f} = \frac{1}{2}m\left(v_f^2 - v_i^2\right) + mg\left(z_f - z_i\right) \qquad \text{Equation 5.6}$$

where v_f and v_i are the final and initial airspeeds of the UAV, z_f and z_i are the final and initial altitudes, m is the UAV mass, and g is the acceleration due to gravity.

Due to the sparse nature of this reward function, we increase the rate of learning recording the states of the aircraft over the trajectory. When a reward is granted at the end of the cycle, we replay these states in reverse order and update the Q-value with Equation 2.1 until the reward propagates to the start of the trajectory, as suggested by [75].

5.4. Experiments

We considered three experiments to test the performance of this controller. First, we evaluated the learned controller on a single dynamic soaring orbit. Then, we extended performance to multiple orbits that arrive at a steady state. Finally, we compared the performance of the learning controller against the sample-based controller under conditions of changing wind. We present each of these experiments here in turn.

5.4.1. Single Orbit Learning

Our first set of experiments employed a simulator based on the same point mass model used to generate DS trajectories. While this model considers basic aerodynamic forces, it does not simulate more complex dynamics. Regardless, it offers three key benefits that are not available in our alternative simulation environment: First it can be configured to run in faster-than-real-time, which accelerates the learning process. Second, the aircraft can be initialized to any state, which allows us to demonstrate arbitrary experiences to the RL controller. Finally, it provides us with baseline performance for the teaching controller, since the controller model is identical to the simulation environment model. For this test we generated a DS trajectory with maximum shear strength w_{max} = 9m/s, shear region height Δz = 10m, and initial airspeed v_0 = 16 m/s. For the teaching phase, we chose the initial location of the glider

as the first collocation point on the DS trajectory. We then demonstrated a series of 10 flights to the RL algorithm using the teaching controller. Since the teaching controller is deterministic, at the start of each episode we corrupted the glider's starting position, attitude, and airspeed with random Gaussian noise to ensure a variety of trajectories. The mean energy gain for these 10 flights was 352.62J, with a maximum energy gain of 359.39J; while the mean RMS path error was 0.93 m. A controller which perfectly tracks this particular trajectory should gain 364.96J of energy. Thus, the difference between this value and the mean energy gain is the result of controller tracking error.

When these 10 flights concluded, authority was then granted to the RL controller. Figure 5.1 depicts the energy gain of the RL controller as it learned new trajectories. Learning ceased when energy gain between episodes was below a target threshold. In this case, it happened also to be 10 episodes. An interesting feature of this result is

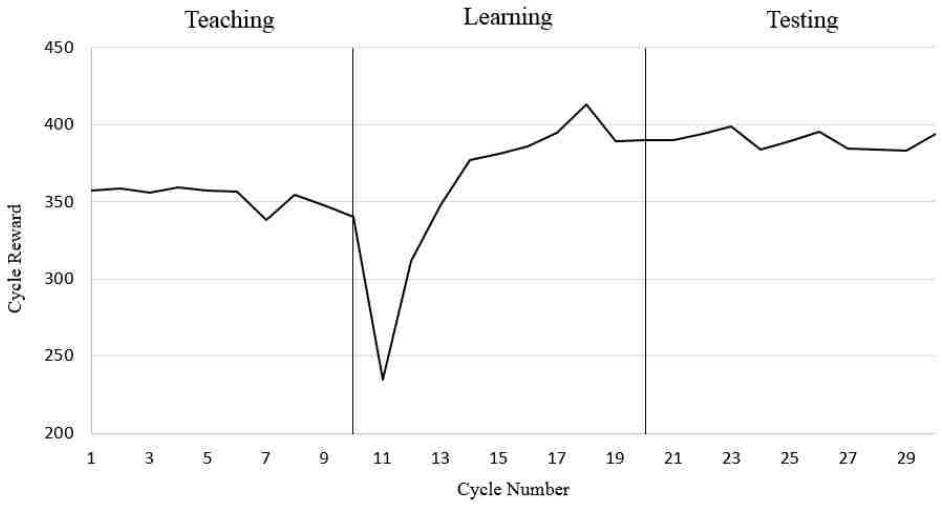


Figure 5.1 – The training process for one DS cycle. The teaching controller performs 10 examples of following the DS trajectory. Then, the learning controller assume agency over the aircraft, initially experiencing a decrease in performance, but soon surpasses the teaching controller. The learner is then held constant and tested at various starting locations to follow the DS trajectory.

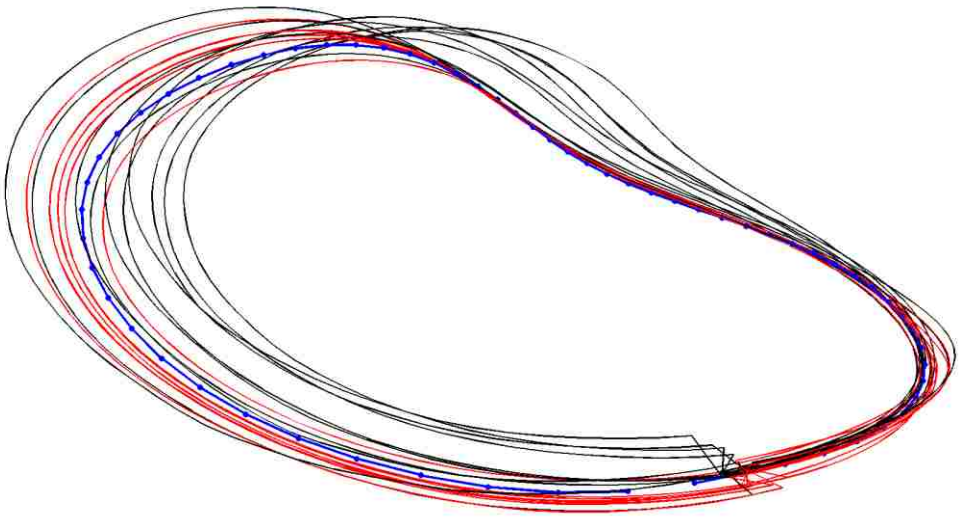


Figure 5.2 – Ten training trajectories (red) and ten learned trajectories (black). The training trajectories aim to track the blue trajectory. The black trajectories aim to maximized energy gained after one cycle.

the decrease in performance seen on the RL controller's initial flight. This phenomenon mirrors that reported by [72]. Once learning ended, we fixed the Q-function and ran 10 more episodes with the RL controller, each with starting conditions corresponding to the first 10 flights on the teaching controller. We saw a mean cycle energy gain of 389.96 J with a maximum of 398.98 J; while the mean RMS path error was 4.63 m. The foregoing statistics are summarized in Table 5.1. From these results we can see that by teaching the RL controller example trajectories, we were not simply encoding the actions of the teaching controller for the RL algorithm to replay. Instead, the RL controller used that experience to find distinct trajectories that were more efficient than any of the demonstrated trajectories.

	Teaching	Learning	Percent Gain
Mean Energy Gain (J)	352.62	389.96	11%
Maximum Energy Gain (J)	359.39	398.98	11%
Mean RMS Path Error (m)	0.93	4.64	-76%

Table 5.1 – Results for the point mass model simulation after learning converged. Means are computed for 10 episodes for each controller.

The paths taken during these flights are shown in Figure 5.2. The blue line shows the target trajectory, the red lines show training paths, and the black lines show learned paths. While the red paths track the blue trajectory closely, the black learned paths deviate from the path in a way that gains more energy compared to the red paths.

Transferring Experience

Figure 5.3 depicts two hardware-in-loop (HiL) soaring trajectories and the same dynamic soaring trajectory used in the previous simulations. The green trajectory employed the teaching controller, which experienced an energy gain of 238.09J, a percent decrease of 32.48% compared to the mean energy gain in the point mass simulator. There are three primary reasons for this: First, latency in responding to commands. Second, the integrated states are based on an incomplete model. Finally, the planning horizon was increased from 1s to 2s. Using the 1s horizon, the controller exhibits overshoot and never converges to the trajectory. Increasing the horizon to 2s eliminates this behavior, but also serves to "round out" the glider path, making it more ellipsoidal instead of kidney bean shaped. The red trajectory was followed by the

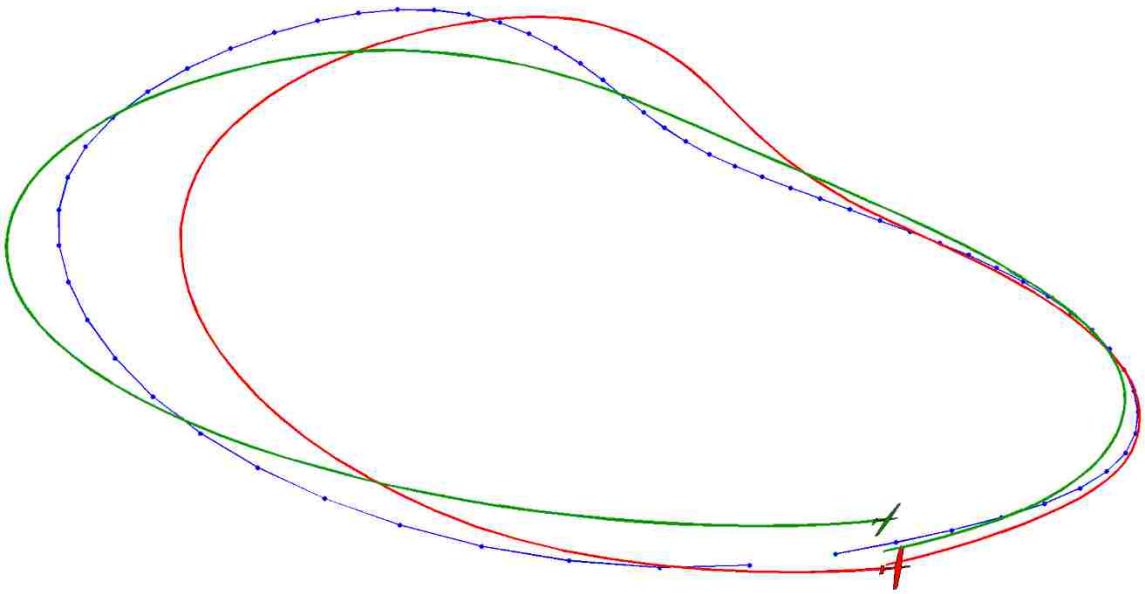


Figure 5.3 – Soaring results for the HiL simulation. The target DS trajectory is depicted in blue, with the RL controller path in red, and the sample-based controller path in green. The RL Controller does not track the blue trajectory but creates similar path from learned experience.

learning controller, and experienced an energy gain of 354.34 J, a percent decrease of 9.13% compared to the point mass simulator. Compared to the HiL sample-based controller, the RL controller was able to extract 49% more energy. The glider did not start on the planned trajectory, but instead of tracking to it and losing energy, the glider used its learned Q-function to find another path which closely resembles the planned one. An interesting result here is the learning controller was not re-trained in the high fidelity simulator model; the Q-function learned from the point mass model generalized to the higher fidelity model with minimal parameter adjustments. We expect a learning controller trained in the HiL environment would perform even better than the results presented here.

5.4.2. Multiple Orbit Learning

The learning method was extended to multiple orbits in simulation. The training process was the same as for one loop but was re-run at orbits of increasingly higher initial velocity. Starting with v_0 = 15m/s and ending with v_0 = 30m/s. This provided the learning controller with experience at different energy levels. Results for this process are shown in Figure 5.4, where the first colocation point for each orbit was selected both the learned and teaching controllers were set to perform one orbit.

To get the glider to soar for subsequent orbits, the training process required a new step; instead of granting the learning controller full authority at this point, we devised a "hand-holding" training process where the learning controller would ask the teaching controller for a control choice if it didn't have sufficient experience to make

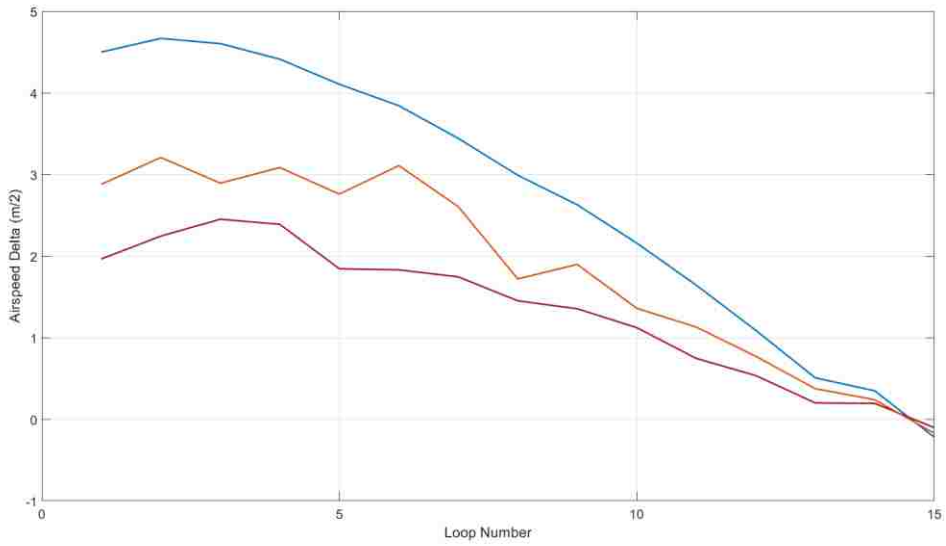


Figure 5.4 – Results from multiple learning loops shown above. The top line (blue) is the energy gain per cycle of a set of dynamic soaring trajectories generated by the global planner. The bottom line (red) is the energy gain per cycle of the sample-based controller following those trajectories. The middle line (orange) is the energy gain of the learned controller following those trajectories after training was finished.

a control decision within the threshold specified in Equation 5.5. Without this hand-holding phase, the learning controller frequently entered states that were not supported by observed experience. This necessitated running both controllers and the global planner simultaneously.

After the training process finished, the hand-holding scenario was turned off and the learning controller was granted full authority over the glider. Results of a multiple cycle dynamic soaring maneuver are depicted in Figure 5.5, were the aircraft started at 15m/s in a w_{max} = 8m/s and Δz = 10m shear wind and cycled to a maximum steady state airspeed of 27m/s after 4 cycles. A trace of the aircraft state over this flight is shown in Figure 5.6.

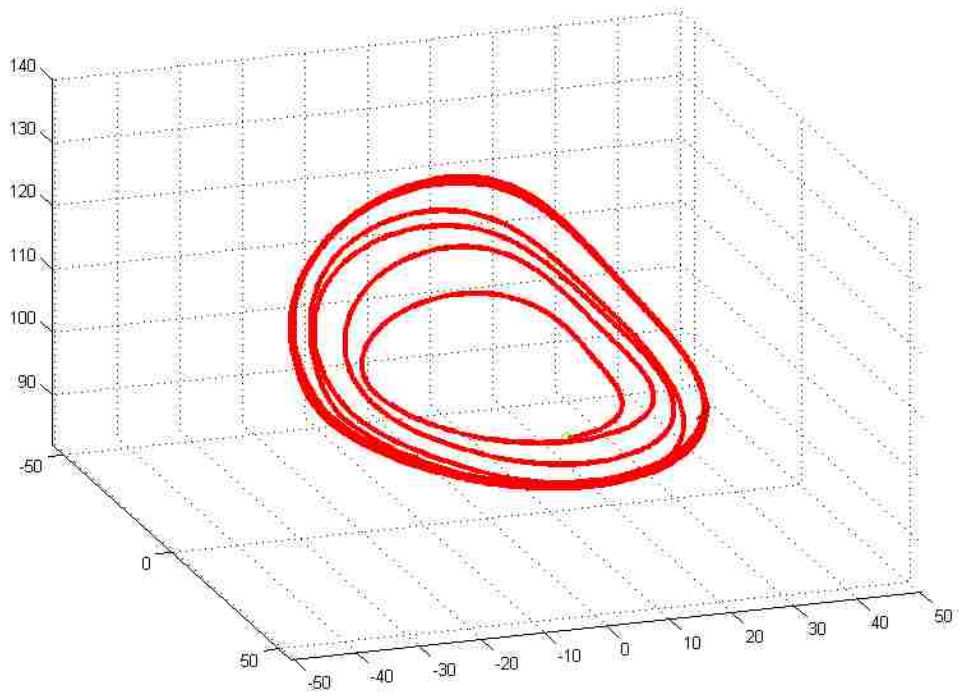


Figure 5.5 – Path of the learning controller followed over multiple DS cycles to a steady state. No global planning or wind mapping is necessary for this controller to function. A steady state is reached after about 4 cycles. Wind is blowing at w_{max} = 8 m/s from the negative y direction. The boundary layer exists between 100 and 110 m altitude.

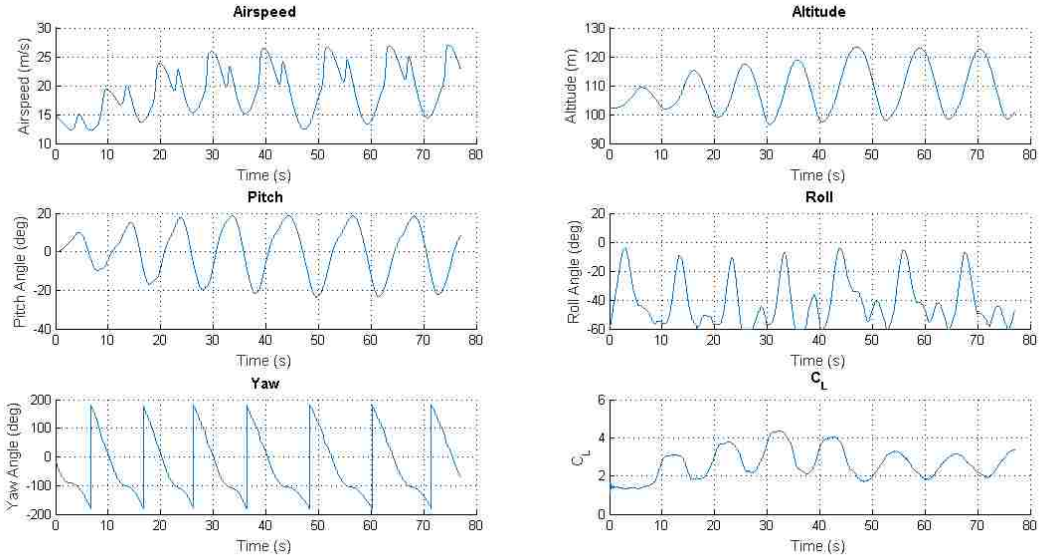


Figure 5.6 – A trace of the aircraft state over the course of several cycles. The initial airspeed of the glider was 15 m/s, and the maximum achieved airspeed was ~27 m/s.

Loop	SBC Energy Gain (J)	RL Energy Gain (J)
1	322.53	432.24
2	304.81	333.03
3	183.27	290.61
4	69.84	205.91
5	33.74	-33.08
6	14.1	
7	-21.95	

Table 5.2 – The dynamic soaring sample-based controller performance compared to the reinforcement learning controller. The sample-based controller takes 6 cycles to reach a stead state, while the learning controller takes only 4.

5.4.3. Controller Robustness Comparison

Our premise for developing the reinforcement learning controller is that because its purpose is not to track a target trajectory, it will be able to perform better in situations when the trajectory or wind field map do not match detected field conditions. The sample-based controller described in Chapter 4 requires launching the aircraft and performing a mapping maneuver, where the aircraft is directed to follow level orbits at several altitudes. This allows the wind map to sample the wind at several points, giving us a good estimate of its profile, direction, and maximum wind speed. This procedure necessarily takes some time, which drains the on-board batteries. Our procedure for this has been to land the aircraft, swap out batteries for freshly charged ones, and then launching the aircraft again to perform a dynamic soaring maneuver. What happens though, if the wind changes between launches? In this section, we compare the performance of both controllers in this scenario.

Experimental Setup

First, we assume the aircraft has landed after mapping the wind field, providing us with an estimate of the wind. For this experiment we assume the detected wind field has parameters w_{max} = 8 m/s, Δz = 10 m, and the wind is blowing from the -y direction. However, after the plane lands, the wind picks up to w_{max} = 10 m/s with the same height of the boundary layer. This extra windspeed should be beneficial for dynamic soaring, but when we launch, we will be planning with a trajectory suited for 8 m/s. Because the sample-based controller is a tracking controller, it will attempt to follow a dynamic soaring trajectory that leaves energy on the table. In situations like this, we've observed that the sample-based controller will eventually destabilize, leading

to violations of our flight conditions (either airspeed falls below the threshold of 11 m/s, the altitude falls below our minimum threshold of 95m, or the coefficient of lift exceeds the limit of +7g). This situation is usually observed when the planned dynamic soaring trajectory does not match the airspeed of the aircraft and is what makes the global planner necessary. For the reinforcement learning controller, we expect it to take advantage of the fact that it is gaining energy at a rate faster than it planned for, and exploit that to perform a successful dynamic soaring maneuver to a steady state. For this experiment, the point-mass model was used, and an initial telemetry was selected and used for both controllers. We ran the sample-based controller first and allowed it to glide until it reached a steady state, or until a flight condition was violated. Then we ran the learning controller and again allowed it to soar until it violated a flight condition or reached a steady state.

Results

Results of this experiment are depicted in Figures 5.7 – 5.11. Figures 5.7 – 5.9 show the paths of both controllers, with the sample-based controller depicted in green and the learning controller depicted in red. Figure 5.10 shows the altitude of both controllers, and Figure 5.11 shows the airspeed. The key finding from this experiment is that while the sample-based controller was able to successfully soar for two cycles, it destabilized after the third and violated the minimum airspeed flight condition. The path plots reveal why: in the side plot (Figure 5.8), we see that the aircraft did not dive sufficiently on its final cycle, and therefore did not gain enough airspeed to reach the next cycle. It then entered a death spiral toward the ground. By contrast, the learning controller performed as expected; on its first cycle, it recognized that it was

gaining more energy than we would have expected in an 8 m/s wind gradient and finished with a final cycle airspeed of 21 m/s, compared to 19 m/s for the sample-based controller. It then went on to perform two more energy gaining cycles, reaching a steady state on the third at a maximum airspeed of almost 30 m/s.

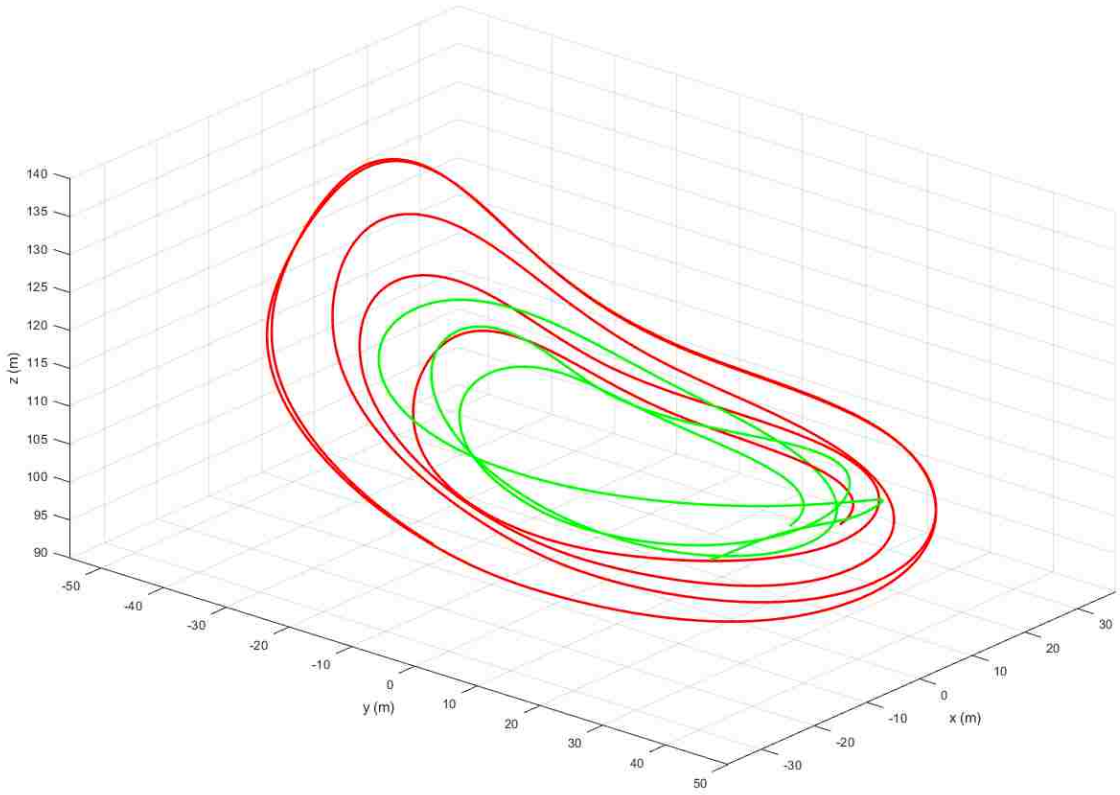


Figure 5.7 – A comparison of the learning controller (red) and the sample-based controller (green) under conditions of uncertainty in the wind field. The expected wind was $w_{max} = 8m/s$, but the actual wind was $w_{max} = 10m/s$. The sample-based controller crashed after only 3 cycles. The reinforcement learning controller soared until a steady state was reached.

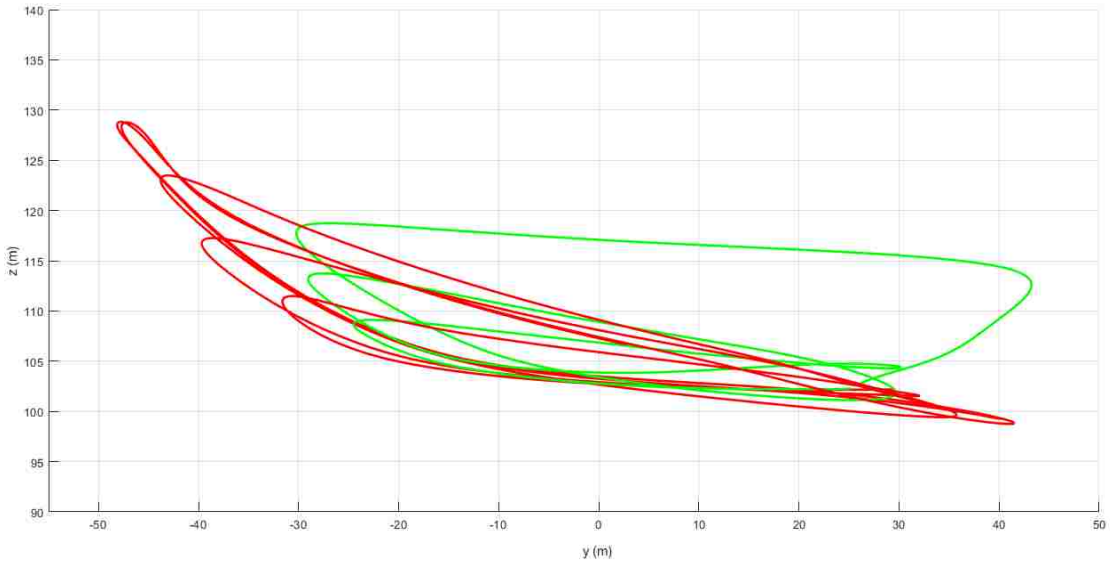


Figure 5.8 – A side view of the robustness comparison.

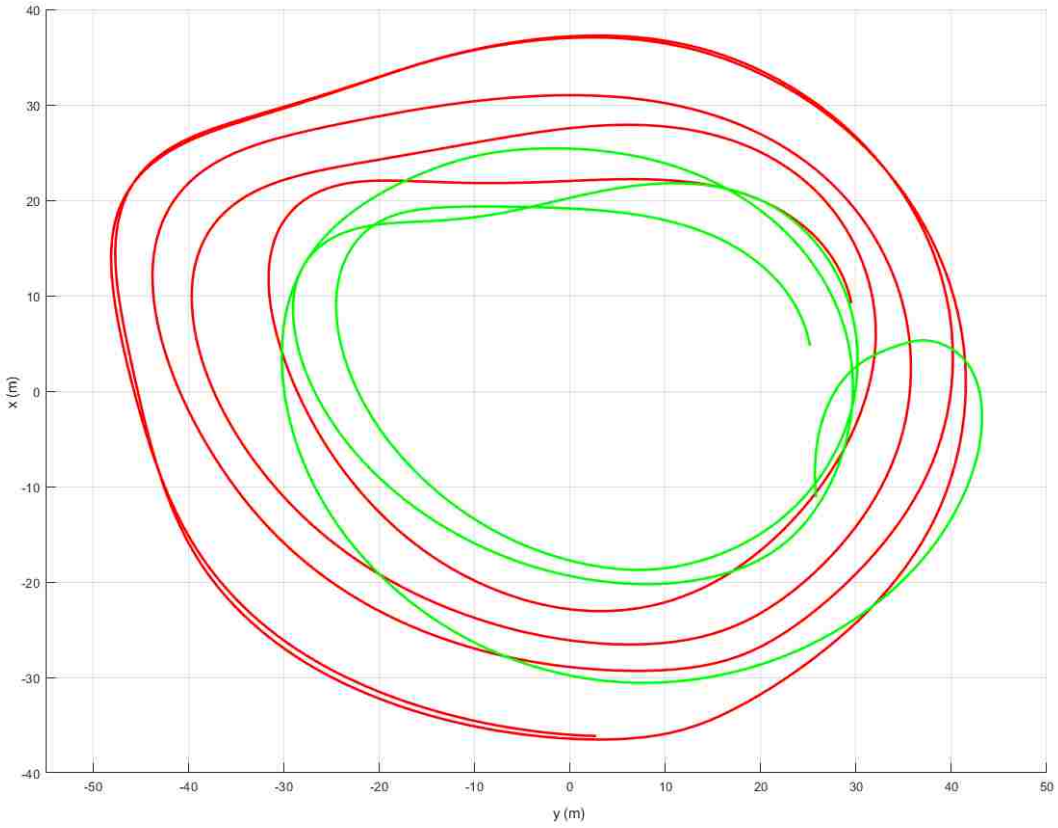


Figure 5.9 – A top view of the robustness comparison.

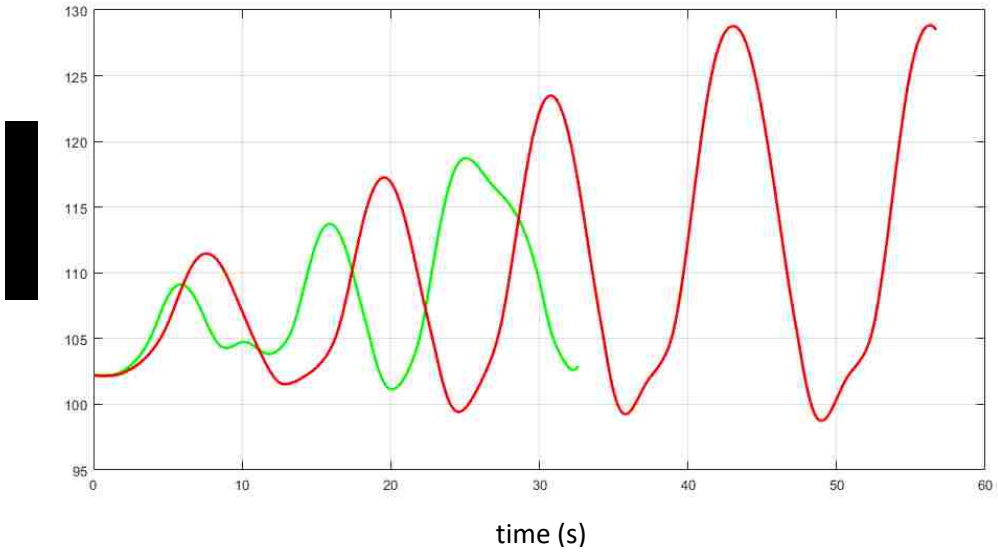


Figure 5.10 – A trace of the aircraft altitude for the robustness comparison. The green line is the sample based controller, the red line is the reinforcement learning controller. The sample-based controller crashes after only 3 cycles.

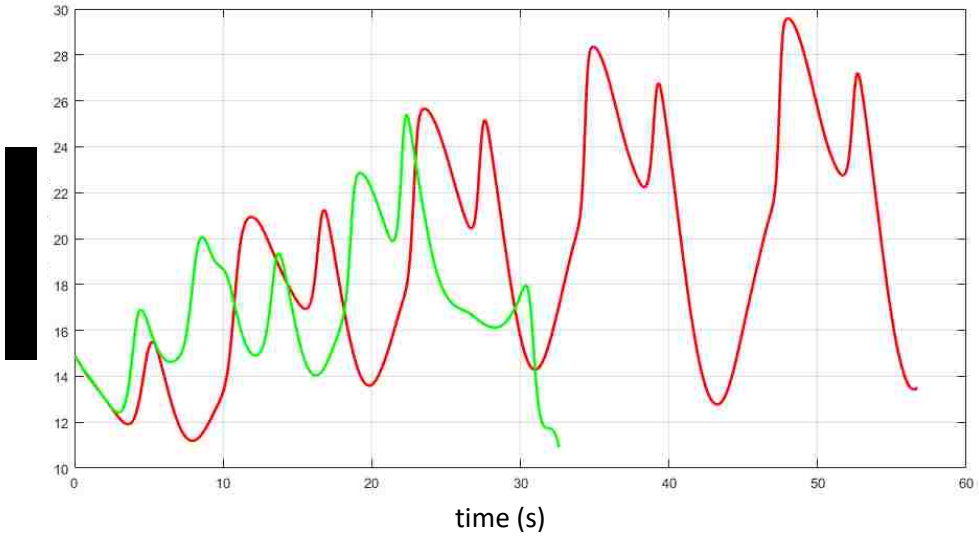


Figure 5.11 – A trace of the aircraft airspeed for the robustness comparison. The green line is the sample based controller, the red line is the reinforcement learning controller. This trace reveals that the aircraft was not able to gain enough airspeed on the third cycle and entered a stall state as it began the fourth loop.

5.5. Summary and Discussion

In this chapter we formulated the dynamic soaring problem in the framework of reinforcement learning. As a method for direct adaptive control, reinforcement learning maps more closely to the purpose of dynamic soaring, and therefore the solution is more elegant with fewer moving parts. However, the tracking control paradigm is necessary to train the learning controller, because it directs the learner toward important regions of the state space. Even with the state space reduction discussed in Section 5.3, the learning process probably would not converge at all without the teaching controller to direct the learner. With the teaching controller, we observed rapid convergence of the Q-function to the point where it was out performing the teaching controller in just a few dynamic soaring cycles.

We observed that the learning controller was able to outperform the tracking controller in terms of energy gained per cycle, because its purpose was in fact to maximize energy gain per cycle rather than to track a trajectory. Figures 5.12 and 5.13 show an instance of this. The blue line in Figure 5.12 represents a dynamic soaring trajectory. The green line is the path of a glider controlled by the sample-based controller. The red line is the path of a glider controlled by the reinforcement learning controller. Both controllers start at the same location, but because the purposed of the sample-based controller is to track the DS trajectory, it burns energy by making a positive bank angle, as shown in Figure 5.13. This positive angle serves to re-align the aircraft with the trajectory, but in doing so it burns about 50J of energy. The reinforcement learning controller learned that keeping the bank angle at most level results in a higher energy gain. Both controllers end up roughly at the same place,

but the reinforcement learning controller did so in a way that maximized its energy gain.

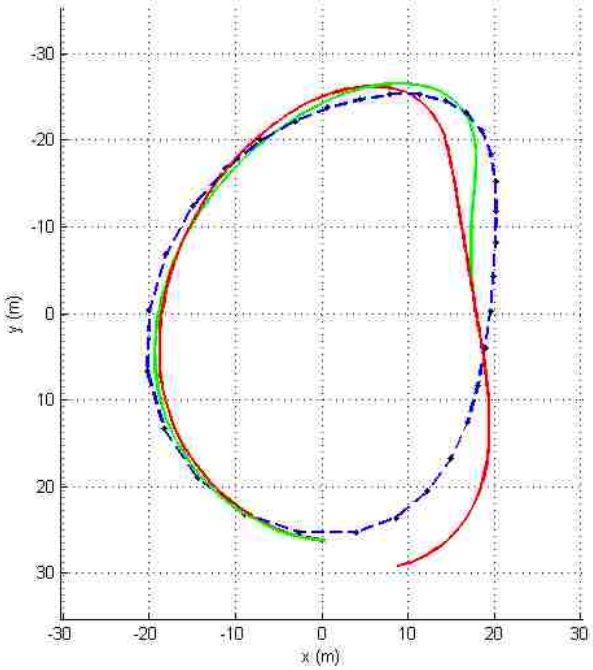


Figure 5.12 – A dynamic soaring trajectory is pictured as the blue dashed line. The green path was generated by the sample-based controller. It makes a positive bank angle to follow the trajectory. The red path was generated by the reinforcement learning controller. It flies straight to maximize energy gain.

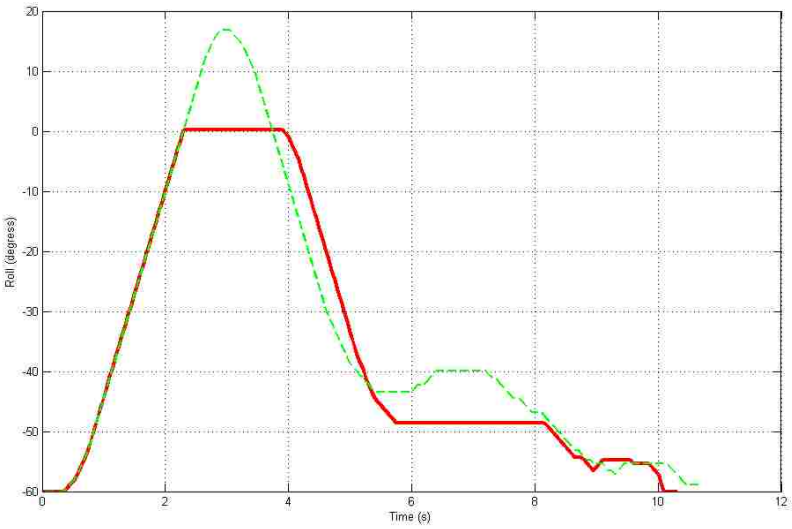


Figure 5.13 – A trace of the bank angle over the DS cycle. The green line is the sample based controller. The learning controller bank angle stays negative, which nets it an additional 50J of energy over the teaching controller.

Implications

The work presented here shows that exploration of direct adaptive controller for dynamic soaring, like reinforcement learning, are promising. A great deal of prior work discussed in Chapter 4 and 5 in the dynamic soaring domain has focused on the generation of optimal trajectories, and it has been recognized that the limitation of tracking controllers is in the time and computational resources it takes to generate these trajectories. Direct adaptive controllers like reinforcement learning obviate the need for global trajectory planners – instead of tracking an optimal trajectory, the aircraft attempts to gain energy through local measurements and past experience. Initial results presented here show that this form of controller can outperform tracking controllers, are more robust to measurement errors, require fewer moving parts, and can be transferred between simulation environments. The implication of these results is that future work should shift away from optimal trajectory generation and tracking and move toward the design and refinement of direct adaptive controllers, whether they be based on reinforcement learning or some other learning scheme.

Chapter 6

Discussion & Future Work

In this dissertation we presented an end-to-end platform for low altitude dynamic soaring in shear winds. This platform consists of hardware and software components. The hardware includes an airframe, autonomy-enabling computing hardware, and a ground control station. The software consists of a series of nodes that communicate via message channels. The nodes as a whole act as a controller for the aircraft. In this section we present future avenues of research that could extend the work presented in this dissertation.

Learning Model Refinement

This dissertation presents preliminary results for a reinforcement learning controller, but there is much more that needs to be done. First, we do not present a treatment for exploration of the state space while learning. This goes in hand with the safety requirement of a flying aircraft, as we don't want to perform any controls that were not tried before and recorded as safe. Finding a way to safely integrate exploration of the state space while soaring will yield a better controller and further gains in performance over time.

Second, more work needs to be done in evaluating the efficacy of the learning controller. We showed initial results of transferring the learned controller from one domain to another, but this should be tested more thoroughly, and should also be transferred to a real-world flight test.

Third, reinforcement learning algorithms should be compared. Deep reinforcement learning (DRL), while currently in vogue and yielding promising, was not tested in this work due to the mismatch between project requirements and the algorithmic costs of DRL. However, its efficacy in this domain should be evaluated, even if it is in an offline context. The vast training data requirements of DRL will need to be overcome, but this can be mitigated through simulation and work on transferring models to real-world domains.

Cloud Integration

The soaring platform presented here has strict computational limitations. Since the research done in 2014, small form factor computers have improved, and cloud computing has presented new opportunities to offload expensive computations to powerful computers that sit on the ground. Because of latency issues with using cloud computers they would not be a good fit for direct control of the aircraft, but it would be interesting to use cloud computers to perform expensive tasks like model refinement or tree balancing. Models could be stored in the cloud, and then selected as wind conditions change. This opens up the possibility of a world-wide wind map or Q-function (depending on the controller) which is dependent on the location of the aircraft. For instance, a dynamic soaring aircraft travelling across the ocean could upload wind maps or learned experience as it is gained, informing the performance of

other aircraft that are head its way. A soaring aircraft in a hurricane could upload its measurements to the cloud as it maps the hurricanes path and vitals. A network of soaring aircraft could even work as a literal cloud network, providing compute resources for other aircraft that are not necessarily soaring, but are flying by and come within range. There are a host of possibilities here that are very exciting avenues for future research.

Field testing

The most notable omission from this paper is the lack of real-world dynamic soaring results. There are two reasons for this. The first is the regulatory environment in the United States regarding drone research. Throughout the course of this research, regulations have changed several times, and remain a moving target. Functionally what this means is that acquiring the necessary permits, personnel, licenses, and certifications to pilot a drone for a dynamic soaring maneuver is time consuming and costly. In the winter of 2016 we had a test scheduled at Weldon California, but due to last minute FAA guidance which removed the glider exemption on unmanned aerial systems, our university risk management office cancelled the project at the last minute. Consequently, our window for soaring at that time had to be moved until we could acquire a Certificate of Airworthiness (COA) for our platform, which pushed back the project by almost a year, at which time testing funds were depleted.

The second reason real-world testing was not carried out is logistical. Dynamic soaring requires the right environmental conditions to perform the maneuver – a constant and strong shear wind. Because of the sensitive electronic nature of the equipment, we cannot soar over the ocean the way albatrosses do; GPS errors alone would

probably cause the drone to fly straight into the water. Even with the right geological formations on land, the wind is only strong enough certain times of the year, leaving brief windows of opportunity to make testing possible. For a dynamic soaring test to occur, a confluence of personnel availability (like the author himself), timing, funds, and weather conditions must converge.

Unfortunately, this opportunity never materialized for us, but we hope this dissertation gives you a head start on your way to making autonomous dynamic soaring a reality.

References

[1] P. L. Richardson, "Leonardo da Vinci's discovery of the dynamic soaring by birds in wind shear," *Notes and Record of the Royal Society Journal of the History of Science,* 2018.

[2] J. W. S. Rayleigh, "The Soaring of Birds," *Nature,* vol. 27, 1883.

[3] J. W. S. Rayleigh, "The Sailing Flight of the Albatross," *Nature,* vol. 40, 1889.

[4] C. D. Cone Jr., "A Mathematical Analysis of the Dynamic Soaring Flight of the Albatross with Ecological Interpretations," *Virtinia Institute of Marine Science Special Scientific Report,* 1964.

[5] M. Denny, "Dynamic Soaring: Aerodynamics for Albatrosses," *European Journal of Physics,* 2008.

[6] G. Sachs, J. Traugott, A. P. Nesterova, G. Dell'Omo, F. Kummeth, W. Heidrich, A. L. Vyssotski and F. Bonadonna, "Flying at No Mechanical Energy Cost: Disclosing the Secret of Wantering Albatrosses," *PLOS One,* vol. 7, no. 9, 2012.

[7] G. Sachs, J. Traugott, A. P. Nesterova and F. Bonadon, "Experimental Verification of Dynamic Soaring in Albatrosses," *The Journal of Experimental Biology,* vol. 216, 2013.

[8] N. J. Adams, C. R. Brown and K. A. Nagy, "Energy Expenditure of Free-Ranging Wandering Albatrosses Diomedea exulans," *Physiological Zoology,* 1986.

[9] C. J. Pennycuick, "Gust soaring as a basis for the flight of petrels and albatrosses (Procellariiformes)," *Avian Science,* vol. 2, no. 1, 2002.

[10] C. J. Pennycuick, "The fight of petrels and albatrosses (Procellariiformes) observed in South Georgia and its vicinity," *Philosophical Transactions of the Royal Society of London,* vol. 300, 1982.

[11] K. Q. Sakamoto, A. Takahashi, T. Iwata, T. Yamamoto, M. Yamamoto and P. N. Trathan, "Heart Rate and Estimated Energy Expenditure of Flapping and Gliding in Black-Browed Albatrosses," *The Journal of Experimental Biology,* vol. 216, 2013.

[12] G. Sachs, "Minimum shear wind strength required for dynamic soaring of albatrosses," *Ibis,* vol. 147, 2005.

[13] P. L. Richardson, "How do albatrosses fly around the world without flapping their wings," *Progress in Oceanography,* vol. 88, 2011.

[14] H. Weimerskirch, M. Louzao, S. d. Grissac and K. Delord, "Changes in Wind Pattern Alter Albatross Distribution and Life-History Traits," *Science,* vol. 335, no. 6065, 2012.

[15] R. J. Boucher, "History of Solar Flight," in *Proceedings of the 20th Joint Propulsion Conference*, 1984.

[16] A. North, M. W. Engel and R. Siegward, "Global Design of a Solar Autonomous Airplane for Sustainable Flight," *IEEE Robotics and Automation Magazine,* 2006.

[17] A. T. Klesh and P. T. Kabamba, "Solar-Powered Aircraft: Energy-Optimal Path Planning and Perpetual Endurance," *Journal of Guidance, Control, and Dynamics,* 2009.

[18] T. E. Noll, J. M. Brown, M. E. Perez-Davis, S. D. Ishmael, G. C. Tiffany and M. Gaier, "Investigation of the Helios Prototype Aircraft Mishap," NASA, 2004.

[19] M. Deittert, A. Richards, C. A. Toomer and A. Pipe, "Engineless Unmanned Aerial Vehicle Propulsion by Dynamic Soaring," *Journal of Guidance, Control, and Dynamics,* vol. 32, no. 5, 2009.

[20] G. Sachs and O. d. Costa, "Dynamic Soaring in Altitude Region below Jet Streams," in *AIAA Guidance, Navigation, and Control Conference*, Keystone, 2006.

[21] J. Grenestedt and J. R. Spletzer, "Optimal energy extraction during dynamic jet stream soaring," in *AIAA Guidance, Navigation, and Control Conference*, Toronto, 2010.

[22] G. Xian-Zohn, H. Zhong-Xi, G. Zheng, L. Jian-Xia and X. Chen, "Influence of Wind Shear to the Performance of High-Altitude Solar Powered Aircraft," *Journal of Aerospace Engineering,* 2013.

[23] J. Grenestedt, C. Montella and J. R. Spletzer, "Dynamic Soaring in Hurricanes," in *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS*, Philadelphia, 2012.

[24] J. Wharington, "Autonomous Control of Soaring Aircraft by Reinforcement Learning," Royal Melbourne Institute of Technology, 1998.

[25] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, Cambridge: The MIT Press, 1998.

[26] L. P. Kaelbling, M. L. Littman and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research,* vol. 4, pp. 237-285, 1996.

[27] R. Bellman, Dynamic Programming, Princeton University Press, NJ: Princeton, 1957.

[28] J. Kober, J. A. Bagnell and J. Peters, "Reinforcement Learning in Robotics: A Survey," *International Journal of Robotics Research,* 2013.

[29] C. Watkins, Learning from Delayed Rewards, Ph.D. Dissertation: King's College, 1989.

[30] C. G. Atkeson, "Using local trajectory optimizers to speed up global optimization," *In Advances in Neural Information Processing Systems,* 1994.

[31] W. D. Smart and L. P. Kaelbling, "Effective Reinforcment Learning for Mobile Robots," in *IEEE Interattional Conference on Robotics and Automation (ICRA)*, Washington D.C., 2002.

[32] R. D. Smallwoord, "The optimal control of partially observable Markov decision processes over a finite horizon," *Operations Research,* vol. 21, no. 5, 1973.

[33] M. Riedmiller, T. Gabel, R. Hafner and S. Lange, "Reinforcement Learning for Robot Soccer," *Autonomous Robots,* 2009.

[34] R. Brooks, "New Approaches to Robotics," *Science,* vol. 253, no. 5025, 1991.

[35] S. Thrun, W. Burgard and D. Fox, Probabilistic Robotics, Cambridge: MIT Press, 2005.

[36] R. C. Johnson, "Microsoft, Google Beat Humans at Image Recognition," EE Times, 2015. [Online]. Available: https://www.eetimes.com/document.asp?doc_id=1325712. [Accessed 2018].

[37] V. Mnih, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. A. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *CoRR,* vol. abs/1312.5602, 2013.

[38] W. Knight, "AlphaGo Zero Shows Machines Can Become Superhuman Without Any Help," MIT Technology Review, 2017. [Online]. Available: https://www.technologyreview.com/s/609141/alphago-zero-shows-machines-can-become-superhuman-without-any-help/. [Accessed 2018].

[39] Google, Inc., "The story of AlphaGo so far," Deep Mind, 2017. [Online]. Available: https://deepmind.com/research/alphago/. [Accessed 2018].

[40] M. B. E. Boslough, "Autonomous Dynamic Soaring Platform for Distributed Mobile Sensor Arrays," Sandia National Laboratories, 2002.

[41] R. J. Gordon, "Optimal Dynamic Soaring for Full Size Sailplanes," Air Force Institute of Technology, Thesis, 2006.

[42] P. L. Richardson, "High Speed Dynamic Soaring," *R/C Soaring Digest,* 2012.

[43] J. B. Patterson, "Manufacturing Techniques Developed for the JetStreamer Dynamic Soaring UAV," Lehigh University, Mater's Thesis , 2104.

[44] Cloud Cap Technologies, "Piccolo SL Autopilot," [Online]. Available: http://www.cloudcaptech.com/products/detail/piccolo-SL1. [Accessed 2018].

[45] J. Bird, J. Langelaan, C. Montella, J. Spletzer and J. Grenestedt, "Closing the Loop in Dynamic Soaring," in *AIAA Guidance, Navigation, and Control Conference*, 2014.

[46] J. J. Bird, "Wind Estimation and Closed-Loop Control of a Soaring Vehicle," The Pennsylvania State University, Master's Thesis, 2013.

[47] H. Jeff and M. Van, "Piccolo Vehicle Integration Guide," Cloud Cap Technology, 2012.

[48] R. Hartley, "In defense of the eight-point algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 19, no. 6, pp. 580-593, 1997.

[49] Eagle Tree Systems, "Instruction Manual for the Micro GPS Expander V4 1st ed.," 2010.

[50] J. W. Langelaan, J. Spletzer, C. Montella and J. Grenestedt, "Wind field estimation for autonomous dynamic soaring," in *IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, 2012.

[51] Y. H. Zhao, "Optimal Patterns of Glider Dynamic Soaring," *Optimal Controls Applications and Methods,* vol. 25, 2004.

[52] N. Akhtar, J. F. Whidborne and A. K. Cooke, "Real-time trajectory generation technique for dynamic soaring UAVs," in *UKACC International Conference on Control*, Manchester, 2008.

[53] O. K. Ariff and T. H. Go, "Dynamic Soaring of Small-Scale UAVs Using Differential Geometry," in *Proc. Of International Bhurban Conf. on Applied Sciences & Technology*, 2010.

[54] I. Mir, A. Maqsood and S. Akhtar, "Optimization of dynamic soaring maneuvers to enhance endurance of a versatile UAV," in *International Conference on Aerospace, Mechanical, and Mechatronic Engineering*, Bangkok, 2017.

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

[56] N. R. J. Lawrance and S. Sukkarieh, "Wind Energy Based Path Planning for a Small Gliding Unmanned Aerial Vehicle," in *AIAA Guidance, Navigation, and Control Conference*, National Harbor, 2009.

[57] T. Flanzer, *Robust Trajectory Optimization and Control of Dynamic Soaring Unmanned Aerial Vehicle,* Stanford University, 2012.

[58] X. Gao, Z. Hou, Z. Guo, R. Fan and X. Chen, "Analysis and Design of Guidance-Strategy for Dynamic Soaring with UAVs," *Control Engineering Practice,* 2013.

[59] S. Park, "Autonomous Aerobatic Flight by Three-Dimensional Path-Following with Relaxed Roll Constraint," in *AIAA Guidance, Navigation, and Control Conference*, 2011.

[60] J. Grenestedt and J. R. Spletzer, "Towards Perpetual Flight of a Gliding Unmanned Aerial Vehicle in the Jet Stream," in *IEEE International Conference on Decision and Control (CDC),*, Atlanta, 2010.

[61] B. P. Gerkey and K. Konolige, "Planning and Control in Unstructured Terrain," in *ICRA Workshop on Path Planning and Costmaps*, Pasadena, 2008.

[62] R. Sutton, G. Barto and R. Williams, "Reinforcement Learning is Direct Adaptive Optimal Control," *IEEE Control Systems,* pp. 19-22, April 1992.

[63] N. R. J. Lawrance and S. Sukkarieh, "Autonomous Exploration of a Wind Field with a Gliding Aircraft," *Journal of Guidance, Control, and Dynamics,* 2011.

[64] N. R. J. Lawrance and S. Sukkarieh, "Path Planning for Autonomous Soaring Flight in Dynamic Wind Fields," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, 2011.

[65] J. J. Chung, N. R. J. Lawrance and S. Sukkarieh, "Gaussian Processes for Informative Exploration in Reinforcement Learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, 2013.

[66] J. J. Chung, N. R. J. Lawrance and S. Sukkarieh, "Learing to Soar: Resource-constrained Exploratin in Reinforcement Learning," *The International Journal of Robotics Research,* vol. 34, no. 2, 2015.

[67] T. Woodbury, C. Dunn and J. Valasek, "Autonomous Soaring Using Reinforcement Learning for Trajectory Generation," in *AIAA SciTech*, National Harbor, 2014.

[68] G. Reddy, A. Celani, T. J. Sejnowski and M. Vergassola, "Learning to Soar in Turbulent Environments," *Proceedings of the National Academy of Sciences,* 2016.

[69] E. Lecarpentier, S. Rapp, M. Melo and E. Rachelson, "Empirical evaluation of a Q-Learning Algorithm for Model-free Autonomous Soaring," in *The French Meeting on Planning, Decision Making and Learning (JFPDA)*, 2017.

[70] G. Reddy, J. Wong-Ng, A. Celani, T. J. Sejnowski and M. Vergassola, "Glider soaring via reinforcement learning in the field," *Nature,* vol. 562, 2018.

[71] L. M. a. P. K. G. Novati, "Deep Reinforcement Learning for Gliding and Perching Bodies," *ArXiv e-prints,* 2018.

[72] W. D. Smart, "Making Reinforcement Learning Work on Real Robots," Brown University, Ph.D. Dissertation, 2002.

[73] D. F. Specht, "“Probabilistic neural networks," *Neural Networks,* vol. 3, no. 1, pp. 109-118, 1990.

[74] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM,* vol. 18, no. 5, 1975.

[75] L. Lin, "Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching," *Machine Learning,* vol. 8, 1992.

Appendices

Appendix A – Fox Platform Component Masses

Plane	g
canopy	62
prop	35
pitot wing airbrake	640
wing airbrake	634
pitot wing	634
wing	601
wing spar	29
wing spring	8
fuselage	955
motor	186
elevator	100
bulkhead	29
Control	
SBC + mount hardware	190
Piccolo SL + mounts	125
Steel Mount + tubes	242
Piccolo Harness	45
GPS antenna	58
Power	
3300mAh battery	276
2200mAh battery	194
1300mAh battery	117
BEC Pro	50
Talon 90	127

Appendix B – Power Calculations

Nominal Voltage (V)	Capacity (mAh)	C Rating	Max Current Discharge (A)	Max Power (W)	Energy (Wh)
12	3300	35	115.5	1386	39.6
12	2200	35	77	924	26.4
12	1350	20	27.0	324.00	16.2

Total System Power (W) **2634.00**
Total System Energy (Wh) **82.2**

Part	Nominal Voltage(V)	Current Draw (A)		Power Consumption (W)	
		Typical	Max	Typical	Max
SBC	5	1.85	2.695	9.25	13.475
Piccolo SL	12	0.3	1.5	3.5	18
Servos	5	1	2.8	5	14
Motor	12	25	45	300	540

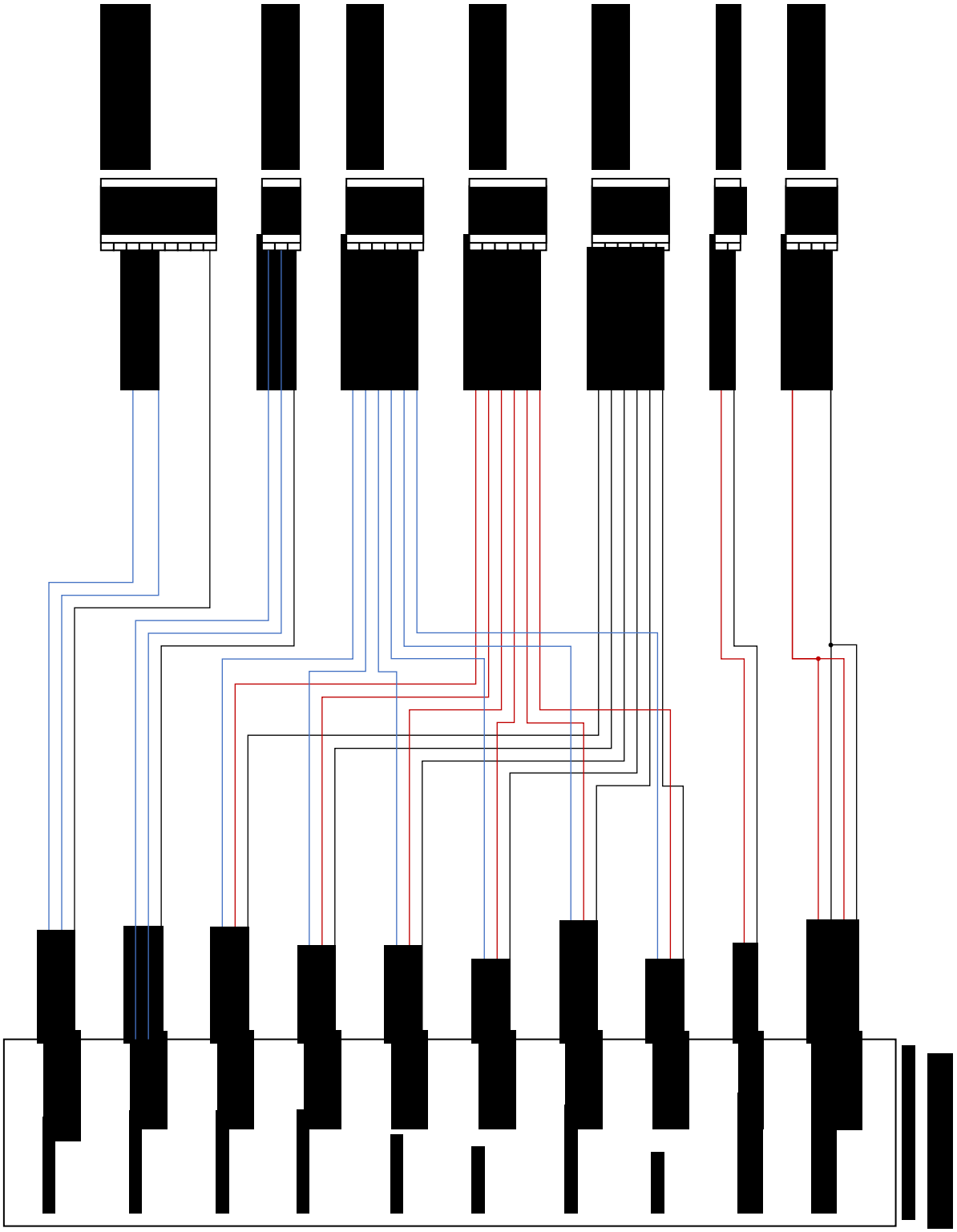
Total Power Draw (W) **317.75** **585.475**

Subsystem	Subsystem Power Draw (W)	Available Subsystem Energy (Wh)	Run Time (mins)
Telemetry	4	13.77	206.55
Propulsion + Actuation	305	39.6	7.79
SBC	9.25	22.44	145.56

Max powered flight time (mins) **7.79**
Max soaring flight time (mins) **146**

Cruise Airspeed (m/s) **15**
Powered Flight Range (km) **7.01**
Soaring Flight Range (km) **131.00**

Appendix C – Auto Pilot Interface



Vita

Corey Montella was born in Doylestown Pennsylvania on August 25, 1986 to Kenneth and Geraldine Montella. He graduated in 2009 with university honors from Carnegie Mellon University in Pittsburgh, Pennsylvania with a B.S. in physics and a B.S. in business administration. Corey worked as a software engineer at Kodowa, Inc. for three years in San Francisco, California before returning to Lehigh to complete this dissertation.