Spring 1-1-2015

An Energy-Aware Trajectory Optimization Layer for sUAS

William Anthony Silva
University of Colorado at Boulder, w.silva32@gmail.com

An Energy-Aware Trajectory Optimization Layer for sUAS

by

William A. Silva

B.S., University of California Los Angeles, 2012

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Masters of Science

Department of Aerospace Engineering Sciences

2015

This thesis entitled:
An Energy-Aware Trajectory Optimization Layer for sUAS
written by William A. Silva
has been approved for the Department of Aerospace Engineering Sciences

Prof. Eric Frew

Prof. Brian Argrow

Date ___________________

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Silva, William A. (M.S., Aerospace Engineering Sciences)

An Energy-Aware Trajectory Optimization Layer for sUAS

Thesis directed by Prof. Eric Frew

The focus of this work is the implementation of an energy-aware trajectory optimization algorithm that enables small unmanned aircraft systems (sUAS) to operate in unknown, dynamic severe weather environments. The software is designed as a component of an Energy-Aware Dynamic Data Driven Application System (EA-DDDAS) for sUAS. This work addresses the challenges of integrating and executing an online trajectory optimization algorithm during mission operations in the field. Using simplified aircraft kinematics, the energy-aware algorithm enables extraction of kinetic energy from measured winds to optimize thrust use and endurance during flight. The optimization layer, based upon a nonlinear program formulation, extracts energy by exploiting strong wind velocity gradients in the wind field, a process known as dynamic soaring. The trajectory optimization layer extends the energy-aware path planner developed by Wenceslao Shaw-Cortez [16] to include additional mission configurations, simulations with a 6-DOF model, and validation of the system with flight testing in June 2015 in Lubbock, Texas.

The trajectory optimization layer interfaces with several components within the EA-DDDAS to provide an sUAS with optimal flight trajectories in real-time during severe weather. As a result, execution timing, data transfer, and scalability are considered in the design of the software. Severe weather also poses a measure of unpredictability to the system with respect to communication between systems and available data resources during mission operations. A heuristic mission tree with different cost functions and constraints is implemented to provide a level of adaptability to the optimization layer.

Simulations and flight experiments are performed to assess the efficacy of the trajectory optimization layer. The results are used to assess the feasibility of flying dynamic soaring trajectories with existing controllers as well as to verify the interconnections between EA-DDDAS components. Results also demonstrate the usage of the trajectory optimization layer in conjunction with a lattice-based path planner as a method of guiding the optimization layer and stitching together subsequent trajectories.

Acknowledgements

First and foremost, I would like to thank my thesis advisor, Professor Eric Frew, for guiding me through my first foray into research. Professor Frew devoted his time above and beyond what was required to assist my understanding of the problem and provide honest feedback. I would also like to thank Wenceslao Shaw-Cortez for providing valuable information during my transition onto this project as well as Elizabeth Wong from UCSD for actively assisting with SNOPT. Finally, I would like to thank my cohort in the Research and Engineering Center for Unmanned Vehicles for providing insight and wisdom to the graduate student process.

Contents

Tables

Table

Figures

Chapter 1

Introduction

1.1 Motivation

Usually referred to as “drones” by the media and public, small unmanned aircraft systems (sUAS) have been responsible for launching a rapidly growing industry that aims to perform sensing or surveillance tasks that are regarded as dangerous or expensive for humans [15]. Industrial applications for fixed-wing platforms include agricultural crop management, high-frequency aerial photography, and persistent weather observation, to name a few. With the complexity of these systems considered, industry has regarded sUAS platforms as ”sensors in the sky” that can provide real-time information about our environment [20]. Fixed-wing sUAS can carry an advantage over quadrotor systems with longer endurance, increased range, and higher cruise velocity. In relatively open skies, these benefits enable greater mission coverage area with an ability to perform greater duration persistent sensing missions. However most sUAS are power-limited by the on-board battery for aircraft performance. Greater sUAS endurance is needed to make several conceptual applications feasible.

Attempts to address sUAS relatively short endurance have been made, particularly within the military sphere [2]. Refueling autonomous sUAS with another aircraft has proven to be a challenging task whose complexity often exceeds practicality and is not applicable to electric motor aircraft. Laser-based systems have been proposed and tested in which a high-power beam is focused on a photovoltaic array attached to the underside of an sUAS [15]. These systems are not only costly, but also ineffective in long-range applications or inclement weather that would obstruct direct line-of-sight. These strategies' drawbacks stem from generating power on the ground and then attempting to transfer it to the vehicle while in flight.

Soaring is a flight technique that extracts wind energy from the immediate surrounding environment and converts it to aircraft potential. Wind is an abundant source of kinetic energy in the atmosphere that can be effectively harnessed given sufficient knowledge of the environment. Soaring exploits the wind by gaining potential energy from wind or thermals and spends this potential to stay aloft. Based on the wind direction, magnitude, and gradient the sUAS executes maneuvers to ensure aircraft sink rate is less than the climb provided by the wind [16].

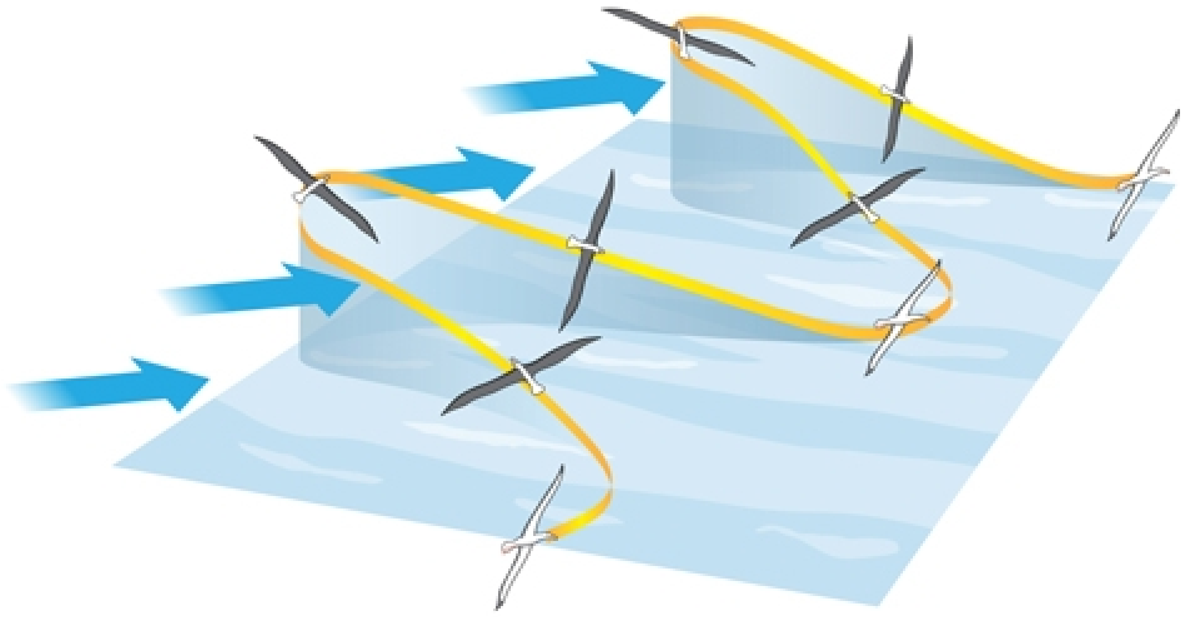


Figure 1.1: Dynamic Soaring [4]

Developing optimized dynamic soaring trajectories is a well-researched topic [1, 3, 10] and has led to several proposals for harnessing wind energy to improve sUAS performance. However, the majority of existing work fails to address the implementation of such a trajectory optimization layer for use in field testing to evaluate performance in a non-simulated environment. Developing a flight system architecture that incorporates online planning and control algorithms poses several practical software and hardware implementation challenges. Such a system must be robust to changing environmental conditions, must be adaptable for unexpected communication interruptions, and adapt to several mission profiles.

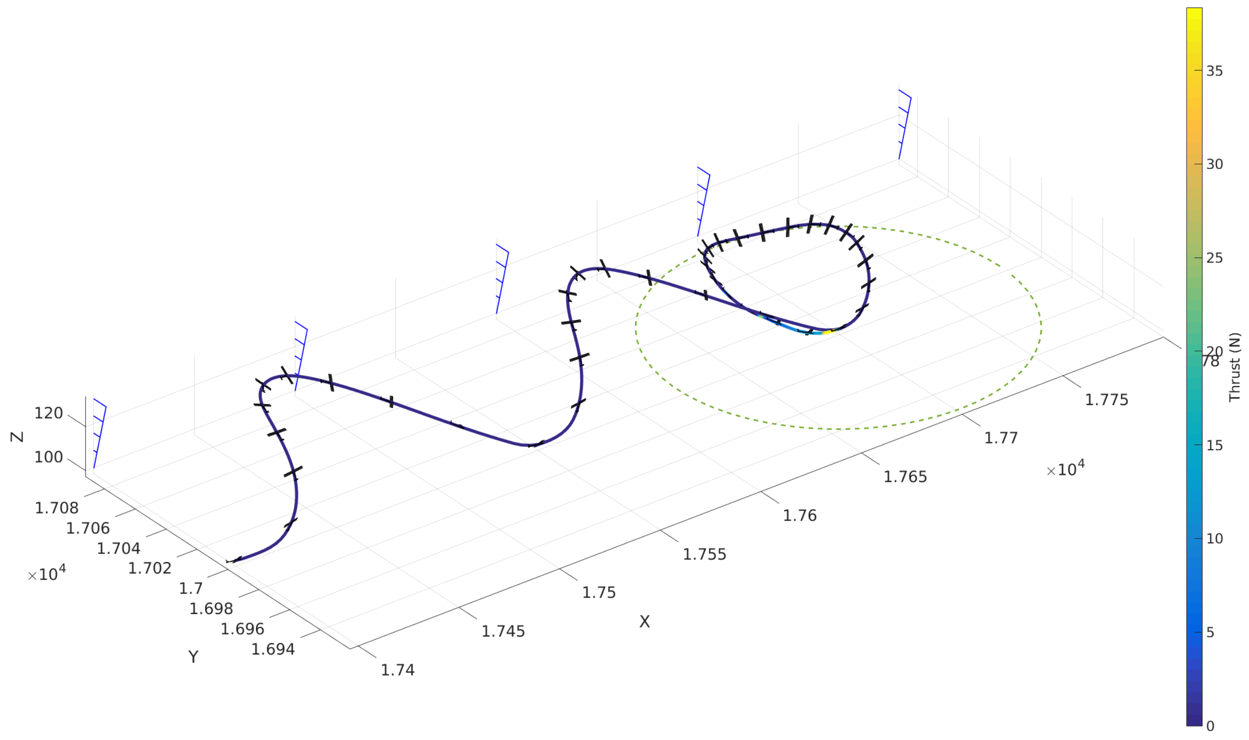


Figure 1.2: Dynamic Soaring Trajectory in boundary layer wind profile [18]

Related works have investigated using optimization to develop soaring trajectories [1, 3, 21, 14, 16]. These works examine the theoretical problem formulation for nonlinear optimization problems by evaluating different cost functions that aim to increase flight endurance. Most works evaluate generated trajectories by comparing them to a baseline Monte Carlo simulation. Bird and Langelaan [1] do demonstrate successful dynamic soaring within a hardware in the loop (HIL) environment. However, a simulated shear ridge is assumed for the wind simulations and all the trajectories are computed a priori. While this requires less online computation, the available library of trajectories may not contain a suitable match for the current wind conditions during flight.

Other works use Rapidly Exploring Random Tree-like (RRT-like) path planners to develop optimal trajectories [9]. This heuristic approach develops online soaring trajectories while simultaneously building a map of the wind environment. That work does admit that a wind map generated solely from on-board wind data using a Gaussian process limited dynamic soaring efficiency during the initial stages of flight. This work also considered a guidance and control strategy developed upon previous path planning work. This work does not consider performance during actual flight.

Another paper explores the use of dynamic soaring trajectories for performing a surveillance mission [6]. The work is based upon utilizing Dubins vehicle paths which represent a simplified model for an sUAS. Similarly to Langelaan, this work uses a best-fit approach, drawing from a library of Dubins curves to generate a feasible candidate trajectory.

This work focuses on the implementation of a dynamic soaring trajectory layer as a component of an Energy-Aware Dynamic Data Driven Application System (EA-DDDAS) and addresses the challenges of operating such a modular component in an online system [5, 17]. The trajectory optimization approach builds upon Wences Shaw-Cortez's work [16] in developing dynamic soaring trajectories by reformulating a nonlinear constrained optimization problem. This work will present the scalability of the layer by investigating numerous mission profiles, demonstrate robust operation within a complex tornadic storm simulation, discuss implementation challenges and solutions, and finally present flight results of the EA-DDDAS in operation.

1.2 Current Work

The goal of this work is to implement a trajectory optimization layer that enables persistent sampling and surveillance trajectories for sUAS in dynamic, previously unknown weather environments. This work examines implementation details for operating an online trajectory optimizer and presents simulation and experimental results. This work evaluates the feasibility of operating a trajectory optimizer in conjunction with a path-following controller in dynamic flight conditions.

One of the primary challenges in planning trajectories during flight is generating an appropriate length trajectory with sufficient lead time for the aircraft controller. A typical mission involves approaching a goal area, possibly loitering, and returning to the starting location. Attempting to perform a nonlinear optimization on such a long time-horizon mission proves to be infeasible with limited computational resources and time. For this reason, the long-term horizon planning is delegated to a lattice-based path planner that provides the trajectory optimization layer with an intermediate goal and initial trajectory. This allows for rapid re-planning of the mission if the wind environment changes without interrupting the near-term trajectory optimization process. By allowing a path planner to run on a longer time horizon, the trajectory optimization layer can stitch together subsequent trajectories during flight.

The trajectory optimization cost functions are divided into guidance and loitering types. The guidance mission is formulated to approach a goal while minimizing time and/or thrust. The loitering mission is concerned with maximizing endurance while remaining in a pre-allocated region.

The EA-DDDAS system consists of several components to enable energy-aware flight (Figure 1.3).

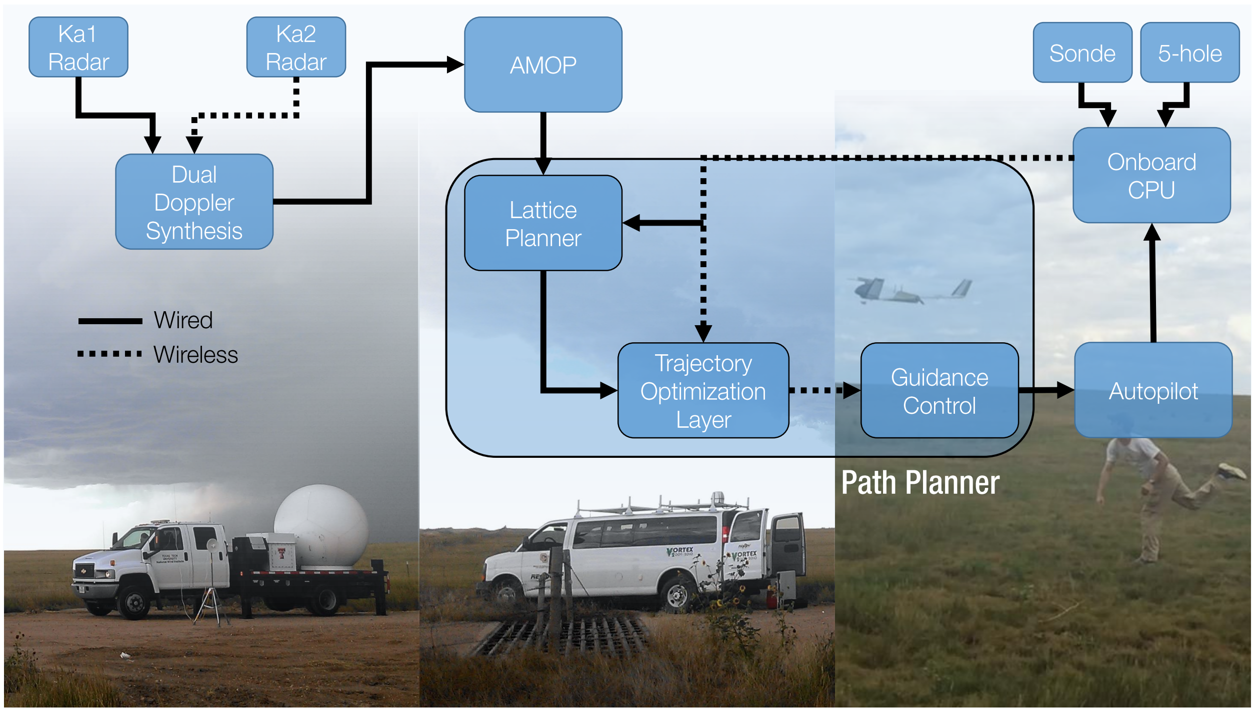


Figure 1.3: EA-DDDAS Block Diagram

The trajectory optimization layer, operated by the mobile ground station (MGS), acts as the interface between the lattice-based planner and the autopilot control logic. The proposed implementation parses the results from the lattice-based path planner into optimal dynamic soaring segments. Once deemed optimal, the segment is passed to the aircraft path following controller and the next segment of the flight is examined next. The trajectory optimization layer has limited scope within the ultimate mission goal and is only concerned with generating near-time horizon optimal trajectories (on the order of one to three minutes).

1.3 Thesis Contributions

(1) Restructuring existing trajectory optimization code for scalability and real-time use:

  (a) Functionality to quickly add new optimization constraints and cost functions;

  (b) Functionality to access a wind database and linearly interpolate wind during optimization;

(2) Formulation of a system architecture between existing components of the EA-DDDAS:

  (a) Formulation of guidance or loiter mission logic implemented during flight;

  (b) Implementation of a wind database that stores all recorded data during a flight;

  (c) Implementation of a server-client architecture between the path planner and trajectory optimization layer;

(3) Analysis and assessment of trajectory optimization layer:

  (a) Analysis of soaring trajectories in a severe storm simulation;

  (b) Analysis of a basic path following controller;

  (c) Assessment of new penalty functions;

  (d) Assessment of the system operating during actual flight;

1.4 Thesis Outline

The system model, equations of motion, and optimization problem formulation are briefly summarized in Chapter 2. Chapter 3 discusses the software and hardware implementation details of the trajectory optimization layer. Chapter 4 demonstrates the system operation simulated within a severe storm environment. Chapter 5 presents and discusses experimental flight results from the Lubbock, Texas deployment of summer 2015. Last, Chapter 6 provides concluding remarks and future work.

Chapter 2

The Optimization Problem

This chapter defines the nonlinear optimization problem to be solved in the trajectory optimization layer. The concept of guidance and loiter trajectories are discussed within the context of a mission selection algorithm.

2.1 Nonlinear Program

Energy-aware trajectory generation in dynamic environments is well-suited as a nonlinear program due to the high degree of non-linearity in aircraft kinematics and relatively large dimension of the aircraft state. The problem can be formulated as a general non-linear optimization problem (NLP) in which a cost function is minimized subject to a set of constraints. The cost function contains aircraft state and control variables that represent or directly affect the aircraft energy state while the constraints enforce aircraft kinematics and mission-specific boundaries. The solution space for an energy-aware trajectory optimization problem was found to be highly nonconvex which posed challenges in dealing with tightly clustered suboptimal local minima. These issues are addressed by reformulating the cost function and constraints to prevent the generation of sub-optimal or infeasible trajectories. This section is largely a summary of Shaw-Cortez's work in establishing an optimization framework using a simple kinematic aircraft model [16].

2.1.1 Aircraft Kinematics

In the context of the optimization problem, the sUAS is modeled as a three-dimensional point-mass aircraft [16]. This maintains realistic optimization convergence times during mission operation. The full aircraft state is described by seven kinematic states and three control inputs. The aircraft inertial position is described by a right-handed North-East-Down coordinate system defined as x, y, and z, respectively (Figure 2.1). Aircraft velocity is described by the magnitude of the air-relative velocity vector, V_a. The air-relative flight-path angle, γ_a, and air-relative course angle χ_a define the aircraft's pointing with respect to the wind. To fully define the aircraft attitude, roll angle ϕ is included as the final state. In this model, the flight-path angle also represents the angle of attack because the coefficient of lift is a directly controlled input. The aircraft state vector is defined as:

$$\mathbf{x}(t) = \begin{pmatrix} x & y & z & V_a & \gamma_a & \chi_a & \phi \end{pmatrix}^{\mathbf{T}} \tag{2.1}$$

Aircraft inputs are defined by the roll rate, $\dot{\phi}$, coefficient of lift, C_L, and thrust, T. The roll rate can be controlled by aileron deflections, the coefficient of lift by elevator deflections, and thrust by motor throttle. The three aircraft inputs are defined as:

$$\mathbf{u}(t) = \begin{pmatrix} \dot{\phi} & C_L & T \end{pmatrix}^{\mathbf{T}} \tag{2.2}$$

Defining air-relative flight path and course angles provides a relationship between the inertial and wind frames. As such, winds must also be defined. Wind inertial velocities are defined as w_x, w_y, and w_z in the vector

$$\mathbf{w} = \begin{pmatrix} w_x & w_y & w_z \end{pmatrix}^{T} \tag{2.3}$$

and the corresponding Jacobian, J_w is

$$J_w = \begin{bmatrix} \frac{\partial w_x}{\partial x} & \frac{\partial w_x}{\partial y} & \frac{\partial w_x}{\partial z} \\ \frac{\partial w_y}{\partial x} & \frac{\partial w_y}{\partial y} & \frac{\partial w_y}{\partial z} \\ \frac{\partial w_z}{\partial x} & \frac{\partial w_z}{\partial y} & \frac{\partial w_z}{\partial z} \end{bmatrix}. \tag{2.4}$$

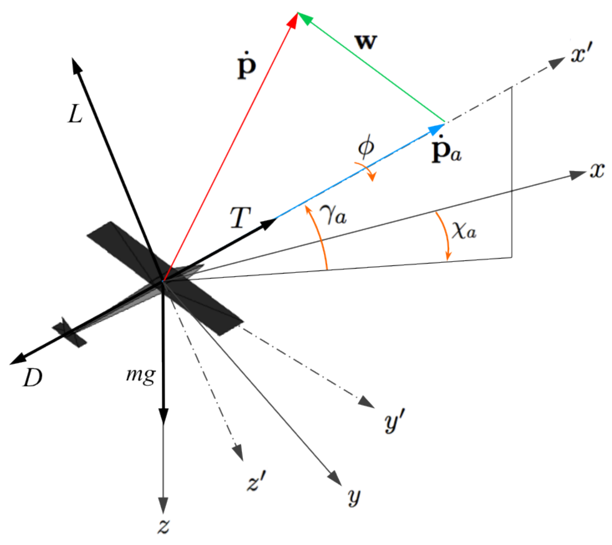


Figure 2.1: UAS Diagram

To transform the wind frame into the inertial frame, we use $R_{i/w}$ and $\mathbf{e}_1$ as defined in [16]. We define the inertial velocity as $\dot{\mathbf{p}}$ where

$$\dot{\mathbf{p}} = \dot{\mathbf{p}}_a + \mathbf{w} = R_{i/w} V_a \mathbf{e}_1 + \mathbf{w} \tag{2.5}$$

To calculate the primary aerodynamic forces lift (L) and drag (D), some basic aircraft properties must be defined. For the trajectory optimization problem, the aircraft is defined by its mass m, wing area S, Oswald's efficiency number e, aspect ratio AR, and zero-lift drag coefficient C_{d_0}. Environmental properties considered are the acceleration of gravity, g, and the density of air ρ, which are both assumed to be constant.

Using a transform to rotate from the wind frame into the inertial frame, we can derive the aircraft kinematic model as a function of the previously defined aircraft states and control inputs. The derivation is clearly outlined in Shaw-Cortez's thesis [16], therefore only the final equations will be listed here. Note that the equations of motion do not include the second control input variable u_2 because C_L is a control input and not a state (as it previously was in Shaw-Cortez's work).

The aircraft kinematic equations of motion are as follows:

$$\dot{\mathbf{x}}(t) = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{V}_a \\ \dot{\gamma_a} \\ \dot{\chi_a} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} V_a \cos\chi_a \cos\gamma_a + w_x \\ V_a \sin\chi_a \cos\gamma_a + w_y \\ -V_a \sin\gamma_a + w_z \\ -g\sin\gamma_a + \frac{T-D}{m} - \begin{pmatrix} \cos\gamma_a \cos\chi_a \\ \cos\gamma_a \sin\chi_a \\ -\sin\gamma_a \end{pmatrix}^{\mathrm{T}} J_w \dot{\mathbf{p}} \\ \frac{1}{V_a}(-g\cos\gamma_a + \frac{L}{m}\cos\phi + \begin{pmatrix} \sin\gamma_a \cos\chi_a \\ \sin\gamma_a \sin\chi_a \\ \cos\gamma_a \end{pmatrix}^{\mathrm{T}} J_w \dot{\mathbf{p}}) \\ \frac{1}{V_a \cos\gamma_a}(\frac{L}{m}\sin\phi + \begin{pmatrix} \sin\chi_a \\ -\cos\chi_a \\ 0 \end{pmatrix}^{\mathrm{T}} J_w \dot{\mathbf{p}}) \\ u_1 \end{pmatrix} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) \tag{2.6}$$

2.1.2 Nonlinear Optimization Problem (NLP)

The nonlinear program is formulated as a nonlinear optimization problem that can be solved numerically, held to a pre-defined feasibility and optimality tolerance. The generic nonlinear program is constrained by the set $\mathcal{F}(\mathbf{z})$ which consists of five groups of constraints. The first group, $\mathcal{F}_f(\mathbf{z})$, are continuous aircraft kinematics in equation (2.6) that are propagated over time and enforced by the Forward Euler approximation scheme

$$\mathbf{x}_{k+1} = \mathbf{x}_k + f(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k)\Delta t, \ \forall k \in [0, N-1]. \tag{2.7}$$

The kinematic constraints are defined as

$$\mathcal{F}_f(\mathbf{z}) = \mathbf{x}_{k+1} - \mathbf{x}_k - f(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w})\Delta t, \quad \forall k \in [0, N-1]. \tag{2.8}$$

The NLP is formulated by representing the continuous dynamics by N collocation points spaced evenly in time by the duration Δt. The decision vector z for the NLP includes the time interval,

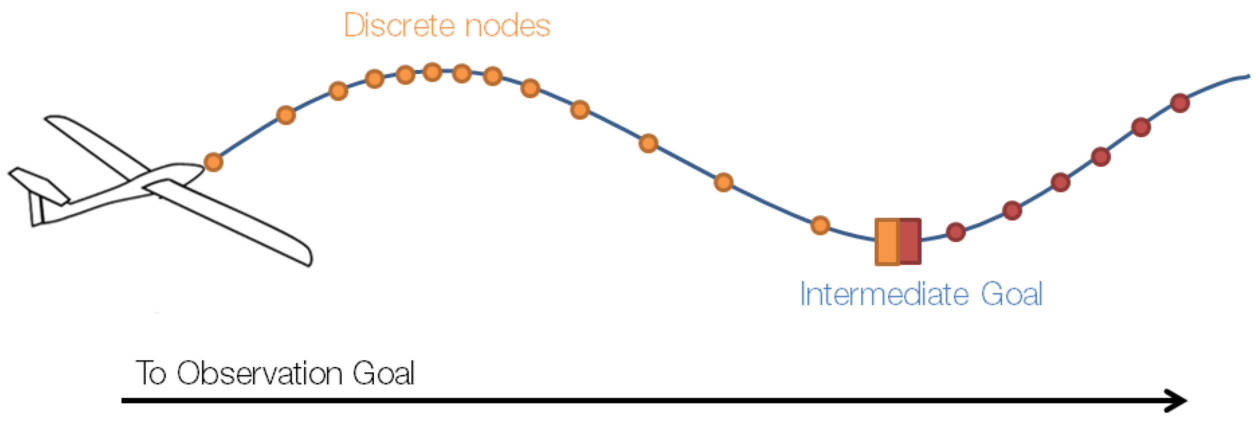


Figure 2.2: A discretized trajectory transitioning to the next.

system states at the collocation points, and system inputs:

$$\mathbf{z} = \begin{pmatrix} \Delta t & \mathbf{x}_0 & \mathbf{u}_0 & \mathbf{x}_1 & \mathbf{u}_1 & \dots & \mathbf{x}_{N-1} & \mathbf{u}_{N-1} & \mathbf{x}_N \end{pmatrix}^{\mathrm{T}} \tag{2.9}$$

in which Δt is part of the solution (Figure 2.2). Although x_0 and u_0 are included in the decision vector, they may be constrained to certain values as function of the current aircraft position and attitude obtained from GPS.

The system bounds, $\mathcal{F}_b(\mathbf{z}) \leq 0$, establish the system limits, such as geographical boundaries as well as aircraft properties. Periodic constraints, $\mathcal{F}_p(\mathbf{z}) = 0$, require the optimization program to match certain state variables from the initial and final trajectory node. The final two constraint categories, $\mathcal{F}_i(\mathbf{z}) \leq 0$ and $\mathcal{F}_o(\mathbf{z}) \leq 0$, are initial conditions and other constraints respectively.[16]. $\mathcal{F}_p(\mathbf{z})$ and $\mathcal{F}_b(\mathbf{z})$ are the periodic and boundary constraints respectively. The periodic constraints are defined as

$$\mathcal{F}_p(\mathbf{z}) \leq \begin{pmatrix} \mathbf{x_N} - \mathbf{x}_0 \\ \mathbf{u_N} - \mathbf{u}_0 \end{pmatrix}, \quad \forall k \in [0, N-1]. \tag{2.10}$$

As with the cost function, the boundary constraints are unique to each type of mission flown but often constrain final and initial states to obtain a desired trajectory behavior. Periodic constraints may only apply to certain states depending on the mission objective. Since the primary objective of generating an energy-aware path is reduce or eliminate battery power consumption, we will always include the control input thrust, T, into the cost function formulation as follows,

$$\mathrm{J}(\mathbf{z}) = \sum_{k}^{N-1} \left(T_k^2 + h_b(\mathbf{z}) \right) + h_t(\mathbf{z}) \tag{2.11}$$

where h_b and h_t are mission-specific cost terms. h_b represents cost terms that are summed over the entire trajectory while h_t is a terminal cost only added once per cost evaluation. In summary, the discrete optimization problem is formulated as

$$\begin{aligned} \textbf{min} \quad & \mathrm{J}(\mathbf{z}) \\ \textbf{s.t.} \quad & \mathcal{F}_f(\mathbf{z}) = 0 \\ & \mathcal{F}_p(\mathbf{z}) \leq 0 \\ & \mathcal{F}_b(\mathbf{z}) \leq 0 \\ & \mathcal{F}_i(\mathbf{z}) \leq 0 \\ & \mathcal{F}_o(\mathbf{z}) \leq 0 \end{aligned} \tag{2.12}$$

.

2.2 Energy Extraction

To better understand the relationship between the aircraft state and its ability to extract energy from the wind, an energy model must be defined. We can begin with a basic potential and kinetic energy equation that models the air-relative energy of the aircraft's mass,

$$e_a = -gz + \frac{1}{2} V_a^2. \tag{2.13}$$

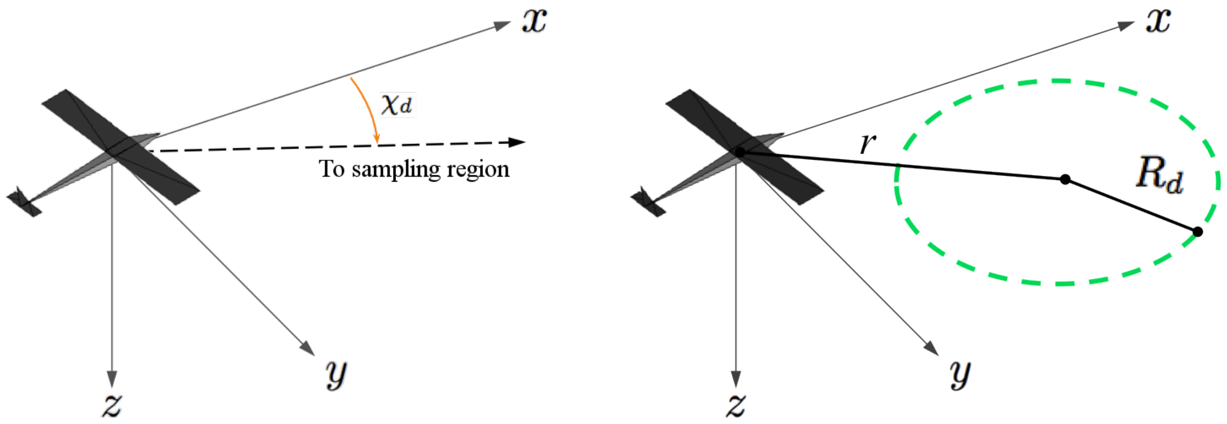
In other words, this represents the specific air-relative energy of the aircraft. In order to determine which of our defined aircraft states will have an impact on the energy extracted from the wind, we can take the derivative of equation(2.13) with respect to time. This yields an air-relative power equation, or power to weight ratio,

$$\dot{e}_a = -V_a \frac{D}{m} + V_a \frac{T}{m} - g w_z - Va \begin{pmatrix} \cos\gamma_a \cos\chi_a \\ \cos\gamma_a \sin\chi_a \\ -\sin\gamma_a \end{pmatrix} \begin{bmatrix} \frac{\partial w_x}{\partial x} & \frac{\partial w_x}{\partial y} & \frac{\partial w_x}{\partial z} \\ \frac{\partial w_y}{\partial x} & \frac{\partial w_y}{\partial y} & \frac{\partial w_y}{\partial z} \\ \frac{\partial w_z}{\partial x} & \frac{\partial w_z}{\partial y} & \frac{\partial w_z}{\partial z} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}. \tag{2.14}$$

This equation contains four distinct terms, with two being functions of the wind field. The first term captures the energy lost to drag and the second term captures the energy gained due to thrust from the aircraft propeller. The third term is known as the static soaring energy contribution. This term reflects the energy gained from vertical wind motion in the environment. The last term is known as the dynamic soaring contribution. This term reflects the energy extracted from the wind gradient itself. This last term is also a function of the aircraft's air-relative pose and inertial speed. The relationship between the gradient values, aircraft pose, and ground speed are not immediately obvious to inspection. For this reason, the problem is solved by a nonlinear optimization problem to minimize thrust used over the entire trajectory. This makes the implicit assumption that by lowering thrust, the aircraft will maximize the energy extracted from the wind and wind gradient. However, this air-relative specific energy rate equation fails to capture the chemical energy stored by the battery and its relationship with the thrust state. It should be noted that this term would be especially important if a battery's finite capacity is taken into account for the optimization. In this work, although the optimization is minimizing thrust usage and therefore battery power consumption, the explicit model relationship is not examined and the battery capacity is assumed to be infinite.

2.3 Guidance and Loiter Missions

This work is focused on implementing an energy-aware trajectory optimization layer in the context of performing persistent sampling missions in adverse conditions. The key performance metric for evaluating trajectories is the amount of thrust used, or energy consumed, by the aircraft to fly a mission. In most severe weather sampling deployments the observation goal area is not within the immediate vicinity of the aircraft launch site. The guidance type mission addresses this issue by generating a trajectory segment that directs the aircraft from its current location towards the observation goal (Figure 2.3a). The guidance mission cost and constraints are formulated to guide the aircraft towards a desired goal point in inertial space. The loiter type mission generates a trajectory in which the aircraft remains in or near a predefined sampling radius circumscribing



(a) Guidance Mission (b) Loiter Mission

Figure 2.3: Loiter and Guidance Missions

an observation goal area (Figure 2.3b). As recognized by Shaw-Cortez[16], the key challenge to implementing the complete sampling and surveillance mission is generating time- and situation-appropriate trajectory segments in real-time and stitching them together online.

For both types of missions, and their respective formulations, the mission objectives do not mandate zero thrust. If desired, a zero-thrust trajectory can be enforced by constraining the thrust, T_k, to zero for all time. R is a desired observation radius that can be set by the user at the beginning of the mission.

As an example, we examine the generic guidance and loiter cost functions in parallel. The cost function and constraint formulation for each respectively is

Guidance

$$\begin{aligned} &\textbf{min} && J(z) = \textstyle\sum_{k=0}^{N-1} k_T T_k^2 + k_{\Delta t}\Delta t + k_p d \\ &\textbf{where} && d = ||\mathbf{p}_N - \mathbf{p}_0|| \\ &\textbf{s.t} && \mathcal{F}(\mathbf{z}) \leq 0 \end{aligned} \tag{2.15}$$

Loiter

$$\begin{aligned} &\textbf{min} && J(z) = \textstyle\sum_{k=0}^{N-1} [k_T T_k^2 + k_p(r - R)] + k_{\Delta t}\Delta t \\ &\textbf{where} && r = ||\mathbf{p}_k - \mathbf{p}_g|| \\ &\textbf{s.t} && \mathcal{F}(\mathbf{z}) \leq 0 \end{aligned} \tag{2.16}$$

Although the generic formulations for the mission look similar, there are some key differences in their behavior. The guidance cost formulation minimizes thrust over the entire trajectory but only minimizes the distance, d, from the last point to the goal. On the other hand, the loitering trajectory minimizes the distance, r, from the trajectory to the goal for each point. Further differences can be found in the formulation of the periodic and boundary constraints. The loitering mission enforces periodicity between the initial and final aircraft states to enable trajectory stitching. The guidance formulation relaxes these constraints to allow the aircraft to traverse the storm environment, ensuring that total energy is not lost during the trajectory by constraining airspeed.

2.4 Longer Time-Horizon Missions

One of the primary motivations for developing energy-aware trajectories is the ability to extend mission life for endurance-limited sUAS platforms. In severe weather sampling missions, trajectory planning should consider a reasonable finite time-horizon for planning trajectories over the storm's duration. One of the limitations to using an NLP in an online algorithm is the relatively short time horizons over which feasible trajectory solutions can be found. As the allowable range of dt increases, so does the number of local sub-optimal minima that must be explored by the NLP. To ensure robustness of the TOL and the trajectories it generates, the allowable range for Δt is limited, forcing longer trajectories to be divided into segments. In addition to adding local minima, a longer time-horizon requires a greater number of discretization nodes to ensure a smooth trajectory. The quadratic constrained optimization sub problem is at least $O(n^2)$ with respect to the number of discretization nodes due to the Jacobian generation step of the optimization routine. Therefore the problem quickly becomes intractable to solve online during a mission.

To address this issue, a lattice-based planning approach has been developed to generate high-level path plans. This path planner uses a Dubin's vehicle model in combination with spatial-temporal storm data to generate a longer finite time-horizon trajectory that leads to the observation goal. A Dubin's vehicle represents a simple dynamic model that tracks position, velocity, and heading. The solution can be quickly updated to reflect macro-scale storm developments in order

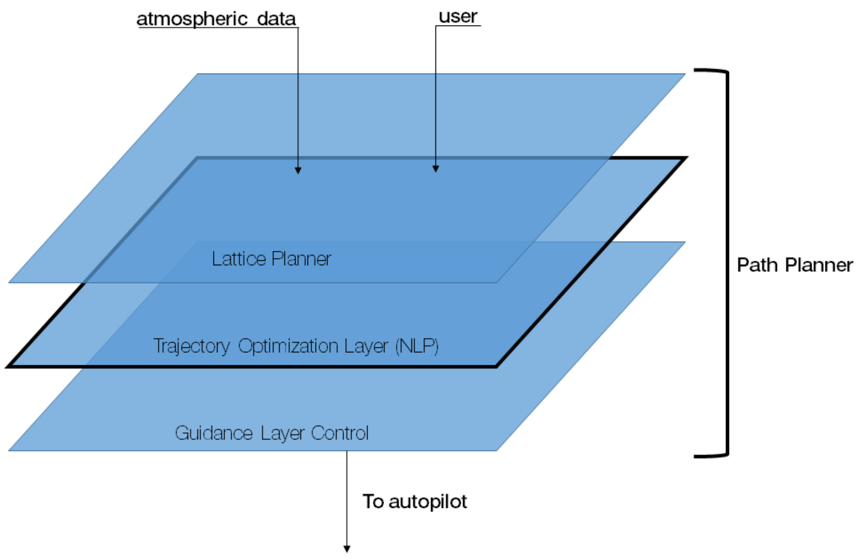


Figure 2.4: The three layers of the path planning system.

to avoid high-risk areas, such as hail cores, heavy precipitation, or electrical activity. The path planner provides the TOL with intermediate segments in order to generate optimal local trajectories which, in turn, are sent to the Guidance Control Layer on-board the aircraft (Figure 2.4). This division of computational labor allows NLPs to be solved online during a mission.

2.5 Mission Stitching

Once provided a new intermediate goal by the lattice planner, the trajectory optimization layer requires logic to determine which cost function and set of constraints to setup and solve the nonlinear optimization problem. A straight-line path from the aircraft location to the goal is segmented and the TOL generates optimal trajectories for each segment sequentially. The goal is to provide a feasible, stitched, and energy-optimal trajectory segment for the aircraft before it finishes its current segment. Trajectories are stitched by constraining the initial state to the previous trajectory's final state.

To ensure feasibility, each segment is first optimized using the thrust-inclusive mission. Due to high wind velocities in severe storms, some regions may be inaccessible to the aircraft even with full use of the motor. This optimization run can quickly determine whether the intermediate goal is reachable, even with non-optimal energy use. Once a baseline trajectory has been planned, each segment is optimized with the no-thrust cost function and constraints set. If the optimization fails, gains are subsequently modified and the optimization is run again. This loop is performed until either an optimal trajectory is found or the aircraft demands a new plan. If the optimization layer cannot find any suitable trajectories to the intermediate goal point, it notifies the lattice planner and requests a different intermediate goal point.This process is repeated until the aircraft nears the observation goal. A threshold is set prior to flight that establishes when the TOL should switch to a loiter type mission.

Chapter 3

Implementation

This chapter discusses the implementation details of the trajectory optimization layer (TOL) in the context of an EA-DDDAS operating online in the field. The various software interconnections to the TOL will be discussed (Figure 3.1) and an in-depth view of the trajectory optimizer will be explored.

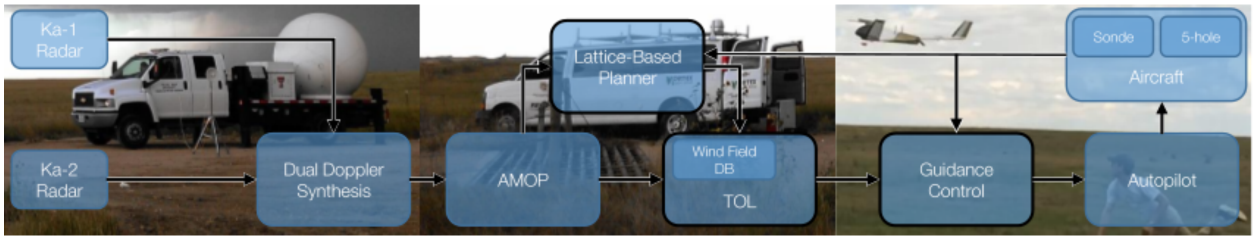


Figure 3.1: A graphical overview of the EA-DDDAS data flow.

3.1 Wind Field Database

The energy-aware trajectory optimization layer requires an efficient method to access the most up-to-date wind data being provided from dual-doppler radar synthesis and localized aircraft measurements. Such a system needs to have the ability to write large volumes of data quickly and also serve rapid function calls from both the trajectory optimization layer and the lattice-based path planner.

MongoDB is an open-source, NoSQL database software written natively in C++, designed specifically for writing, indexing, and looking up enormous data sets. The database is based on a collection-document schema where each document is a BSON-formatted (binary JSON) item. Collections can be assigned a compound-index which enables rapid look-up of multi-dimensional wind, making it ideal for a four-dimensional wind field. MongoDB also supports aggregation techniques to call large batches of data by loading portions of the database into memory based upon the nature of the aggregation command. Not only is this useful for simple look-up data calls but it can be employed to perform operations on large batches of data online in the field.

Within the TOL context, the database is deployed on the same machine as the mission selection logic and the optimizer, although due to its ability to provide data over a network, it could be deployed anywhere in the EA-DDDAS framework. The wind database populates its data from netCDF files generated by the Atmospheric Models for Online Planning software (AMOP). The AMOP is responsible for regularly generating current and forecasted wind fields sourced from Ka-radar dual-doppler synthesis data. Indexing automatically begins on the data once it is written, enabling faster look-ups for the trajectory optimizer and the lattice-based planner.

The wind field database also relieves the optimizer of data stewarding so data can be recorded regardless of the trajectory optimizer's status. By allowing the optimizer to only load a relevant subset of wind for the current trajectory goals, program memory leaks are reduced. This becomes a significant benefit when the optimization routine is run continuously for hours on end with hundreds of optimization runs performed consecutively.

The wind field is stored as a spatio-temporal grid over a 10 kilometer by 10 kilometer grid spanning 250 meters of altitude. The X- and Y-dimensions are gridded into 200 nodes each spaced by 50 meters. The altitude is separated into 5 levels, also with 50 meter spacing. The data is forecasted by the AMOP up to 30 minutes with 3 minute spacing between sets (Figure 3.2). The database is also capable of storing wind field data being collected by on-board sensors, such as a 5-hole probe or a sonde. Interfaces were developed to write data into the wind field from both netCDF files from the AMOP as well as a JSON stream over a UDP socket from the aircraft. The data are then accessed by the TOL and any other EA-DDDAS component via a network port on the machine hosting the wind field database. In this particular setup, since the database resides

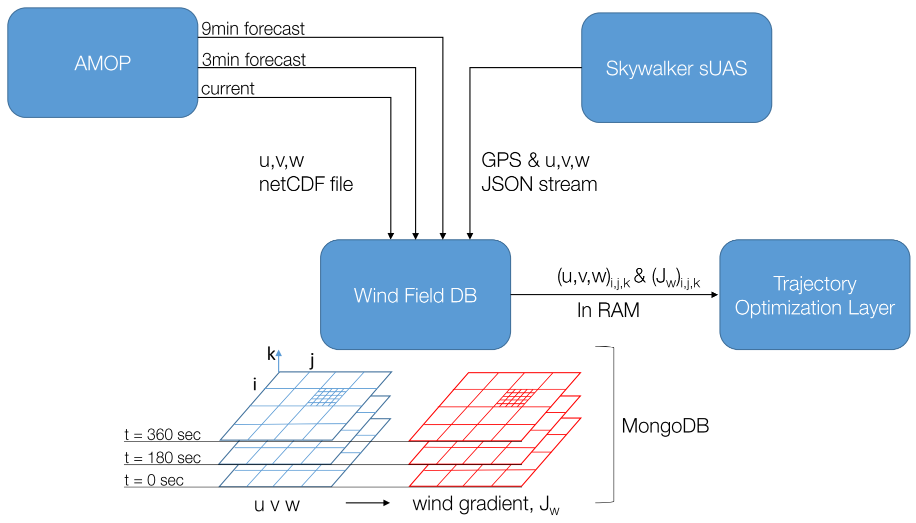


Figure 3.2: Estimated wind data from the AMOP to the wind database. The region containing the aircraft is finely gridded using linear interpolation.

on the same machine as the TOL, the data are sent via RAM to the TOL for queries.

The database can also store precipitation data collected from 88D products. This data can provide useful estimates of the storm track and help define no-fly zones where the sUAS could be at risk from physical damage during flight. Similarly stored on a spatial and temporal grid, the data can be rapidly updated to provide useful engineering estimates for the EA-DDDAS components. Although the current trajectory optimization layer does not directly consider precipitation, the lattice planner can mark these areas as high-risk and avoid sending the aircraft near them. The database enables both the TOL and the path planner to adjust and re-plan to the moving storm environment.

From a systems perspective, the MongoDB platform for atmospheric data management is ideal for providing data access to system components that are spread out in the field of operation. The database can be read and updated remotely via network link, ensuring that highly localized wind data collected on-board and dual Doppler synthesis forecast data are recorded when available and pushed to the rest of the network.

3.1.1 Wind Interpolation

The wind is stored on a discrete spatial-temporal grid, therefore interpolations are needed to provide the wind velocities and gradients for any candidate trajectory being evaluated by the trajectory optimizer. Since millions of these calls must be made online and the dual-doppler synthesis data is on a relatively large grid scale, a linear interpolation technique is used. The eight gridded datapoints surrounding the point of interest are acquired from the wind database. Next, the distance between the point of interest and each grid point is calculated. Eight shape functions for an eight-node trilinear hexahedron are defined with the origin of the coordinate system residing in the lower-left corner of the cube (Figure 3.3),

$$\zeta = \frac{x_p - x_0}{x_s} \quad \eta = \frac{y_p - y_0}{y_s} \quad \mu = \frac{z_p - z_0}{z_s} \tag{3.1}$$

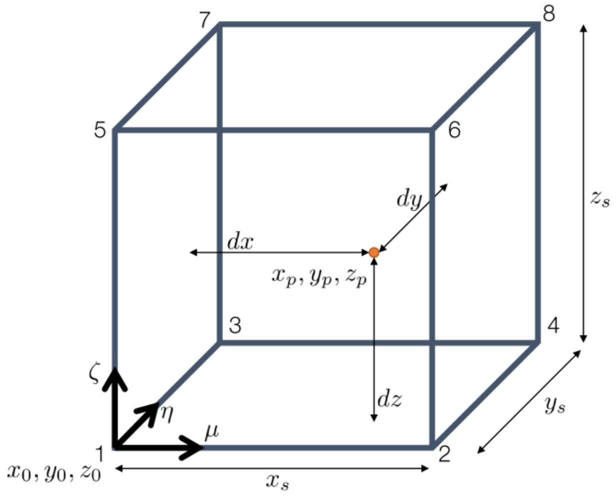


Figure 3.3: Voxel within a gridded wind field.

$$\begin{aligned} N_1 &= (1-\zeta)(1-\eta)(1-\mu) \\ N_2 &= (1-\zeta)(1-\eta)(\mu) \\ N_3 &= (1-\zeta)(\eta)(1-\mu) \\ N_4 &= (1-\zeta)(\eta)(\mu) \\ N_5 &= (\zeta)(1-\eta)(1-\mu) \\ N_6 &= (\zeta)(1-\eta)(\mu) \\ N_7 &= (\zeta)(\eta)(1-\mu) \\ N_8 &= (\zeta)(\eta)(\mu) \end{aligned} \quad . \tag{3.2}$$

Using these shape functions, we can compute the estimated three dimensional wind vector,

$$u = \sum_{i=0}^{8} N_i u_i \quad v = \sum_{i=0}^{8} N_i v_i \quad w = \sum_{i=0}^{8} N_i w_i \tag{3.3}$$

where u_i, v_i, w_i are the gridded velocities at the corner i, respectively. To compute the estimated wind Jacobian we can again turn to the shape functions. Taking partial derivatives with respect to the cube coordinates is relatively straight-forward. Once these are calculated, we can finally compute the estimate of the spatial wind Jacobian,

$$\begin{aligned}
\frac{\partial u}{\partial x} &= \sum_{i=0}^{8} \frac{\partial N_i}{\partial \zeta} u_i, \quad \frac{\partial u}{\partial y} = \sum_{i=0}^{8} \frac{\partial N_i}{\partial \eta} u_i, \quad \frac{\partial u}{\partial z} = \sum_{i=0}^{8} \frac{\partial N_i}{\partial \mu} u_i \\
\frac{\partial v}{\partial x} &= \sum_{i=0}^{8} \frac{\partial N_i}{\partial \zeta} v_i, \quad \frac{\partial v}{\partial y} = \sum_{i=0}^{8} \frac{\partial N_i}{\partial \eta} v_i, \quad \frac{\partial v}{\partial z} = \sum_{i=0}^{8} \frac{\partial N_i}{\partial \mu} v_i \\
\frac{\partial w}{\partial x} &= \sum_{i=0}^{8} \frac{\partial N_i}{\partial \zeta} w_i, \quad \frac{\partial w}{\partial y} = \sum_{i=0}^{8} \frac{\partial N_i}{\partial \eta} w_i, \quad \frac{\partial w}{\partial z} = \sum_{i=0}^{8} \frac{\partial N_i}{\partial \mu} w_i
\end{aligned} \tag{3.4}$$

3.2 Trajectory Optimizer

The trajectory optimizer is a C++ based code that sets up an NLP optimization problem and solves it using a commercial solver named Sparse Nonlinear Optimizer (SNOPT). The code is object-oriented to facilitate further development and expansion. This section will describe the approach and execution of the main sections of the code.

3.2.1 Parameters

Each problem formulation is defined by a set of four parameters files. Each parameter file is loaded into memory as a public object within the problem class. This allows the parameters to be accessed by the SNOPT code as well as any plotting or analysis functionality developed in the future.

3.2.1.1 Aircraft

The aircraft parameters file defines physical properties and limits of the aircraft such as mass, wingspan, and bank angle limits to name a few. The aircraft properties for the Tempest UAS are derived from an Athena Vortex Lattice (AVL) model (Table 3.1). Rate limits and stall speeds are constrained to more conservative limits to allow for trajectories that can be feasibly tracked by the autopilot's inner PID control loops.

3.2.1.2 Gains

The gain parameters file provides a default set of cost function gains for the given problem (Table 3.2). These gains can be overwritten by the mission selection logic (Section 3.3) during execution if needed. This file can be easily expanded to add gains for any new cost function formulations.

m	5.7419	kg	mass
b	3.0671	m	wing span
S	0.6282	m^2	wing area
e	0.9693	–	Oswaldś efficiency factor
AR	14.97	–	aspect ratio
C_{d_0}	0.0243	–	parasitic drag coefficient
$C_{L_{min}}$	-0.45	–	minimum lift coefficient
$C_{L_{max}}$	0.9	–	maximum lift coefficient
ϕ_{max}	45	$^\circ$	maximum bank angle
$V_{a_{min}}$	10	m/s	minimum airspeed
$V_{a_{max}}$	30	m/s	maximum airspeed
γ_{max}	20	$^\circ$	maximum air-relative flight path angle
$\dot{\phi}_{max}$	20	$^\circ/s$	maximum roll rate
T_{max}	0	N	minimum thrust
T_{min}	56.33	N	maximum thrust

Table 3.1: RECUV Tempest parameters from an AVL simulation.

T_k	100	thrust gain
P_k	1	position gain
V_k	0	velocity gain
A_k	0	angle gain
dt_k	0	time gain

Table 3.2: A sample set of gains for a guidance mission.

3.2.1.3 Limits

The limit parameters file establishes spatial and temporal limits to the trajectory problem (Table 3.3). These parameters can be overwritten within the software to allow dynamic geo-fencing for the trajectory optimization. For example, the minimum and maximum x limits can be set to a parametric equation to emulate a moving tracker vehicle. The moving bounding box will force the trajectory optimizer to generate paths that follow the vehicle. Note that because the trajectory optimizer uses a NED coordinate frame, z is positive in the down-direction.

3.2.1.4 SNOPT

The SNOPT parameters file includes SNOPT-specific parameters such as the number of time-segments states, states, control inputs, and boundary constraints for the problem (Table 3.4). It

dt_{min}	0.05	s	minimum dt
dt_{max}	0.12	s	maximum dt
x_{min}	-inf	m	minimum x
x_{max}	inf	m	maximum x
y_{min}	-inf	m	minimum y
y_{max}	inf	m	maximum y
z_{min}	-inf	m	minimum z
z_{max}	0	m	maximum z

Table 3.3: Minimum and maximum spatial-temporal limits.

also includes feasibility and optimality tolerances. It is important that these parameters match the information supplied in the problem-specific source file that defines the objective function and boundary constraints. The number of states and control inputs will generally remain constant between problems.

ts	100	number of time steps
n_{states}	7	number of states
$n_{controls}$	3	number of control inputs
$n_{boundary}$	12	number of boundary constraints
opt_{tol}	1E-4	optimality tolerance
$feas_{tol}$	1E-5	feasibility tolerance

Table 3.4: SNOPT parameters.

3.2.2 Problem

The optimization code parses arguments passed from the mission selection logic code (Section 3.3). These arguments instruct which mission type is to be initialized for the run. An instance of the base class named "problem" is constructed associated with an inheriting child class corresponding to the mission type (Algorithm 1). The base class constructor generates the four parameter class objects as public attributes. A network connection is then established to the wind database and stores a small wind cache to be robust against a lossy connection to the database. Since the aircraft kinematics do not differ between mission types, the forward-Euler aircraft constraints are generated a priori using a C++ symbolic engine that compiles automatically to source code. SNOPT-specific variables are initialized and the limits are set on each discrete state for the trajectory. An upper and lower limit is provided via the SNOPT interface to enforce equality or inequality constraints. The mission specific constructor provides the mission cost, constraints, and required expressions. Next, a goal-specific initial trajectory is calculated, and the symbolic sparse gradient is generated.

Algorithm 1 Problem class constructor

```
Require: goal, mission
1: Generate parameter objects
2: Connect to Wind Field DB
3: Build wind cache
4: Initialize SNOPT
5: Set limits
6: Generate seed trajectory
7: Evaluate sparse Jacobian
8: return instance
```

3.2.3 Cost and Constraints

The cost and constraints are input to the C++ source code by the user prior to mission start. The mission-specific child class inherits methods from either a guidance or loiter class which inherits from the general problem class (Figure 3.4). The objective function is defined by a method named "cost" that involves a for-loop to sum terms over the entire trajectory. The boundary constraints, which include periodic constraints and other physical limitations is defined by a method named "Boundary Constraints".

3.2.4 Sparse Jacobian

The most challenging portion of setting up an NLP for a complex, multi-constrained problem is the efficient computation of a sparse Jacobian. The sparse Jacobian is provided to SNOPT in order to assist with the gradient traversal portion of the SQP algorithm. Since the state vector to be optimized includes all states and control inputs over all time, the resulting Jacobian is $n_{constraints} = (n_{states} + n_{inputs}) * n_{steps} + n_{periodic}$. For a typical problem, this includes approximately 1600 constraints over 2000 discrete states, resulting in a 3.2 million entry Jacobian. While fortunately,

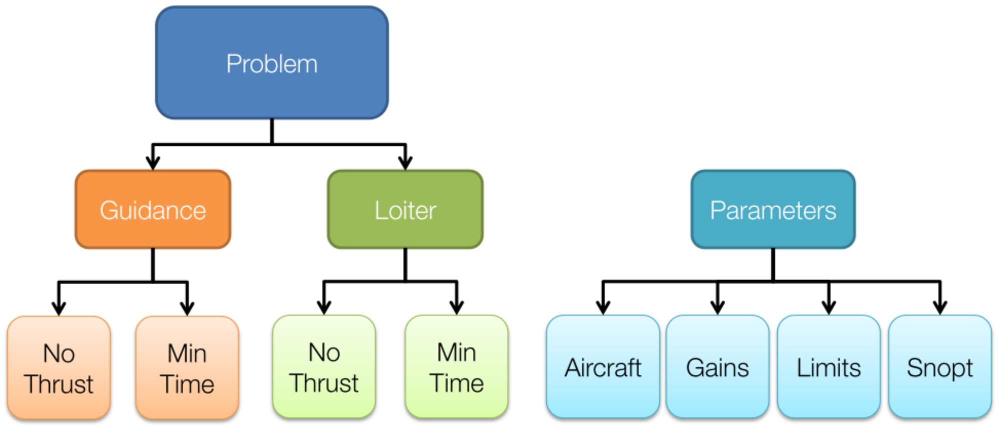


Figure 3.4: Class inheritance structure for optimization problems.

most of this matrix is sparse, it is critical to accurately place each entry in the matrix and have it be adaptable to several different cost functions and boundary constraints, both in formulation and number (Figure 3.5). The size of the Jacobian grows exponentially with an increase in discretization steps. To address this scalability and adaptability challenge, a symbolic C++ engine was used to analyze the symbolic derivatives of the cost function, kinematic constraints, and boundary constrains for all constraints over all time steps. This sparsity pattern was used to generate a map relating constraint functions to non-zero entries during numeric evaluation. Unfortunately, as with most symbolic engines, using a substitution method to calculate numerical results from a symbolic expression is costly with respect to CPU time. To prevent this, time-generalized versions of the constraints are created using a map to prevent duplicates. Another map keeps track of the specific constraint and state that were used to generate the derivative. These time-generalized expressions are then used to generate and compile C code during execution and link the resulting dynamic library back to the main code. Now the time-generic versions of all of the constraint derivatives are available in a C dynamic library for efficient execution. To generate a Jacobian from this library requires no further logic. The constraint map is used to make the correct function calls and evaluations from the new C library. While the initial generation of this library takes no more than a couple of seconds, these libraries can be saved and re-linked upon another execution of the optimizer, increasing time savings for future runs.

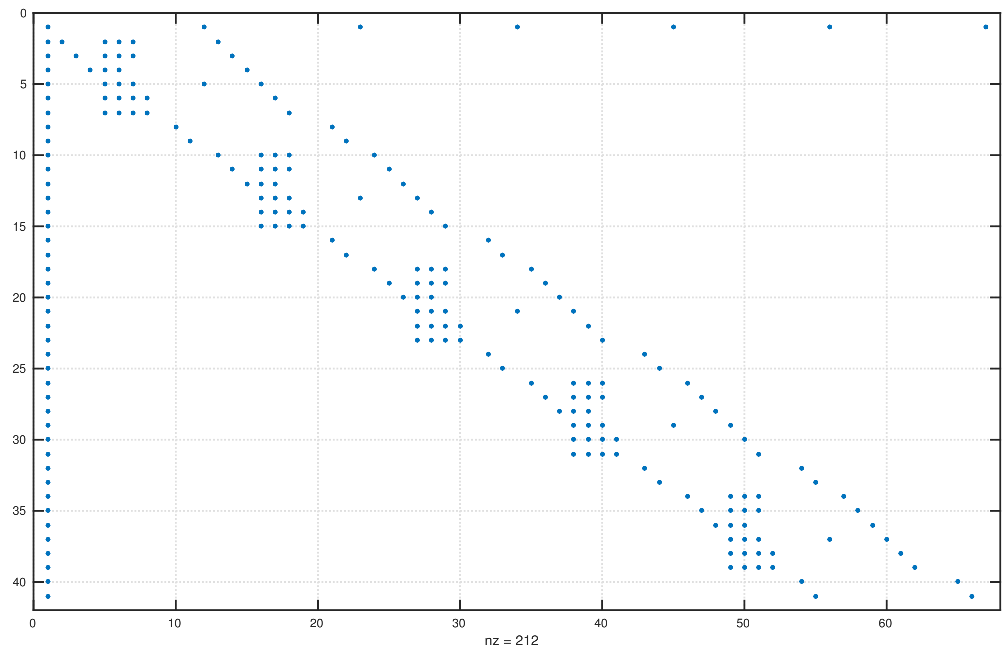


Figure 3.5: The Jacobian sparsity pattern for 5 time steps of a guidance mission. The objective gradient is the top most row.

3.3 Mission Selection Logic

The mission selection logic software is the interface for all network communication between other EA-DDDAS components and the trajectory optimization layer (Figure 3.6). Written in a scripting language (Python), this software is easily modifiable to the addition or removal of EA-DDDAS components in the future development of the program. It is the outer-level controller for the trajectory optimization runs and also acts as a steward for the wind field database.

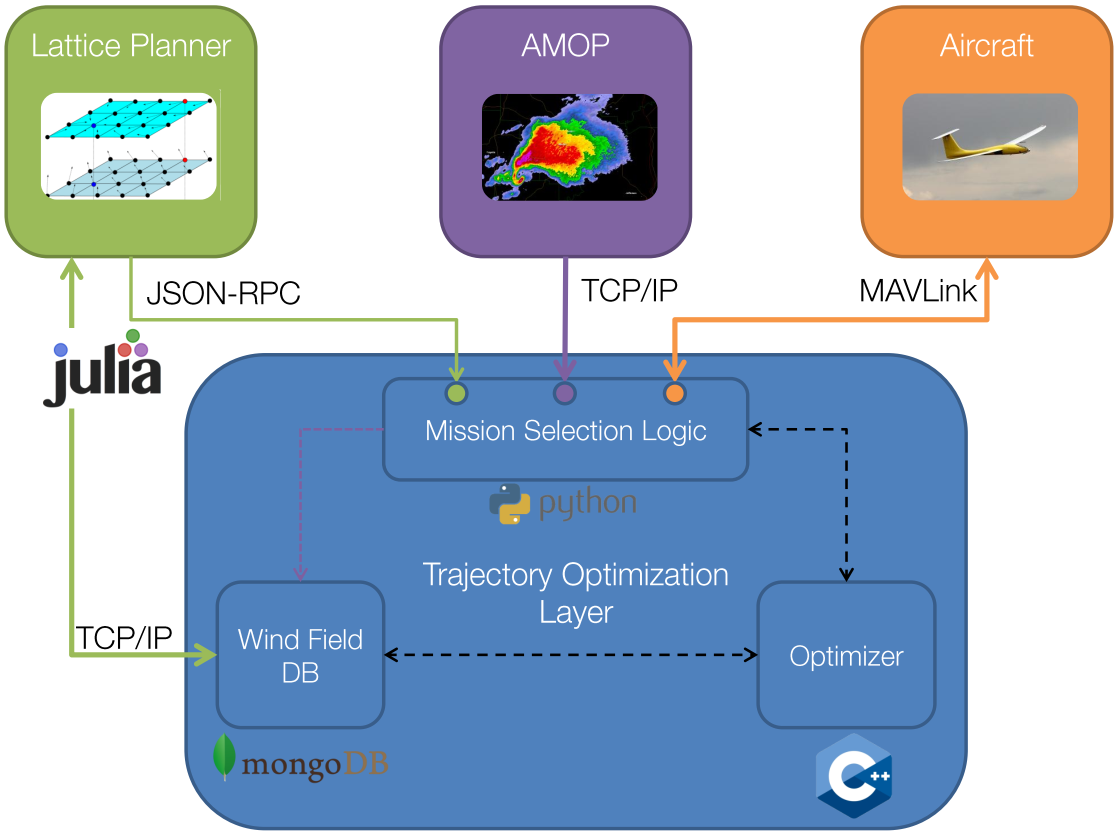


Figure 3.6: Network diagram for the TOL.

3.3.1 Communication with the AMOP

The AMOP regularly generates current and forecasted wind fields sourced from Ka-radar dual-doppler synthesis data. The mission selection logic acts as a converter for the raw data files (netCDF format) by sequentially loading them into memory and performing a bulk-write operation to the wind field database. From preliminary benchmarks, the database is capable of writing approximately 250,000 entries per second. This is most likely limited by the computer's 7200 RPM disk drive. The write performance could be largely boosted by a solid-state drive or a sharded database cluster across servers.

Fortunately, MongoDB is an ACID (Atomicity, Consistency, Isolation, Durability) database that prevents race-condition issues for access during a batch-write operation. The details of the wind field database are discussed at length in Section 3.1.

3.3.2 Communication with the Lattice Planner

The mission selection logic is responsible for making calls to the extended time-horizon lattice planner, written in Julia, in order to receive intermediate goal points. To implement this demand and supply model, a JSON remote procedure call (JSON-RPC) server and client system is used to handle data communication between the two systems (Figure 3.7).

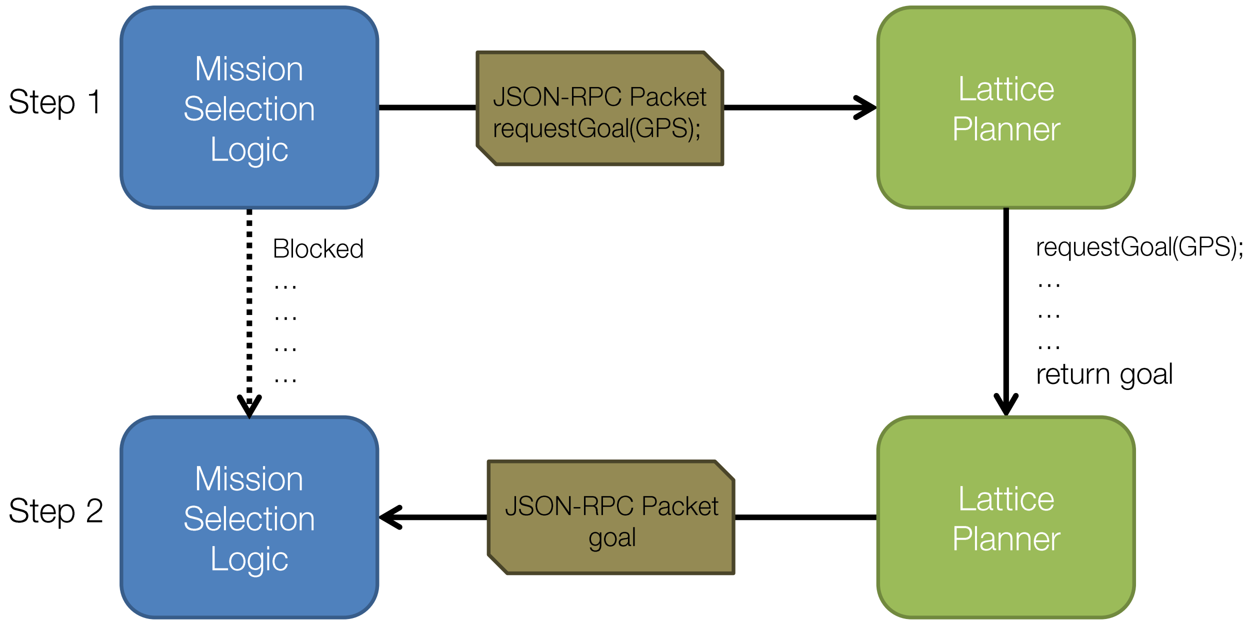


Figure 3.7: JSON-RPC in the TOL context.

The mission selection logic packages a function request along with the aircraft GPS coordinates. The high-level path planner operates as an RPC server, listening for JSON formatted function calls to arrive. On a separate thread, the lattice planner is continuously updating its cost map of the environment to adapt to changing storm conditions. Once a request is received from the mission selection logic, the lattice planner generates an intermediate goal point and returns the data across the RPC link. As an RPC client, the selection logic is blocked until an intermediate goal point is received, barring a time-out due to disconnection. This interface allows the lattice planner to generate uninterrupted updates of its own cost map while providing online intermediate goal points to the mission selection logic. This JSON-RPC framework also handles the complex timing and blocking requirements that are often needed between two parallel processes that need to communicate time-sensitive information.

The lattice planner cost update thread also communicates with the MongoDB wind field server through a TCP/IP link to access wind field data online (Figure 3.6). This communication link does not run directly through the mission selection logic because there is no guarantee that the wind field server will be executing on the same hardware as the mission selection logic. In other words, if the wind field database were moved to a cloud computing system, the software interfaces between mission selection logic, the lattice planner, and the wind field server would remain the same.

3.3.3 Calling the Optimizer

After obtaining an intermediate goal and segmenting the trajectory for stitching, the mission selection logic is responsible for initiating sequential optimization runs (Algorithm 2). The mission selection logic runs as a while loop until the mission_incomplete flag is set to $False$, implying that the mission is complete. Each segment is formulated as a function call with arguments that include the intermediate goal, desired observation radius (if applicable), cost function gains, and the mission type.

Once an optimal candidate trajectory is returned by the optimizer, the selection logic initializes the next segment at the previous trajectory's final state. The C++ based optimizer code is pre-compiled as a dynamic library and loaded into Python as a package. This allows for seamless function calls and data transfer between the two codes. The optimizer is spawned by the selection logic as a separate parallel process, allowing the selection logic code to continue performing tasks while the optimization runs. In the future, this feature could be exploited to run parallel optimization runs for a given segment with different gains (Figure 3.8). The best run, measured by the cost

Algorithm 2 Mission selection logic stitches energy-aware trajectory segments

```
Require: $goal$
 1: while mission incomplete do
 2:   $loc$ = obtain aircraft GPS coordinates
 3:   distance to goal = $goal - loc$
 4:   if distance to goal $> leg$ then
 5:     Run Optimizer($guidance$, $leg$)
 6:   else if distance to goal $> radius$ then
 7:     Run Optimizer($guidance$, distance to goal - $radius$)
 8:   else
 9:     Run Optimizer($loiter$, $radius$)
10:     mission incomplete = $False$
11:   end if
12:   if run is optimal then
13:     Advance to next leg
14:   else
15:     Adjust $gains$, $leg$ and re-run
16:   end if
17: end while
18: return $trajectory$
```

metric, would be stored and used to initialize the next batch of optimization runs. Another use could be to plan sequential legs simultaneously and ensure that the state end-points are within a reasonable limit to allow stitching.

3.3.4 Communication with the Aircraft

The aircraft and mission selection logic communicate GPS information and trajectories using MAVLink, a well-established UAS communication protocol [11]. This communication link is facilitated by a cellular LTE connection as it is widely available in the US and allows high-speed communication over long distances between the sUAS and the mobile ground station. GPS data is being streamed from the PixHawk autopilot over a UDP socket at 30Hz. The GPS coordinates are converted to East-North-Up coordinates relative to the Ka-1 radar to provide aircraft location in context of the measured wind field. Data is only sent to the PixHawk if a pre-stitched candidate trajectory has been solved by the optimizer. For preliminary path following tests, the data is sent as a series of interlaced waypoints and airspeed commands. Adhering to MAVLink

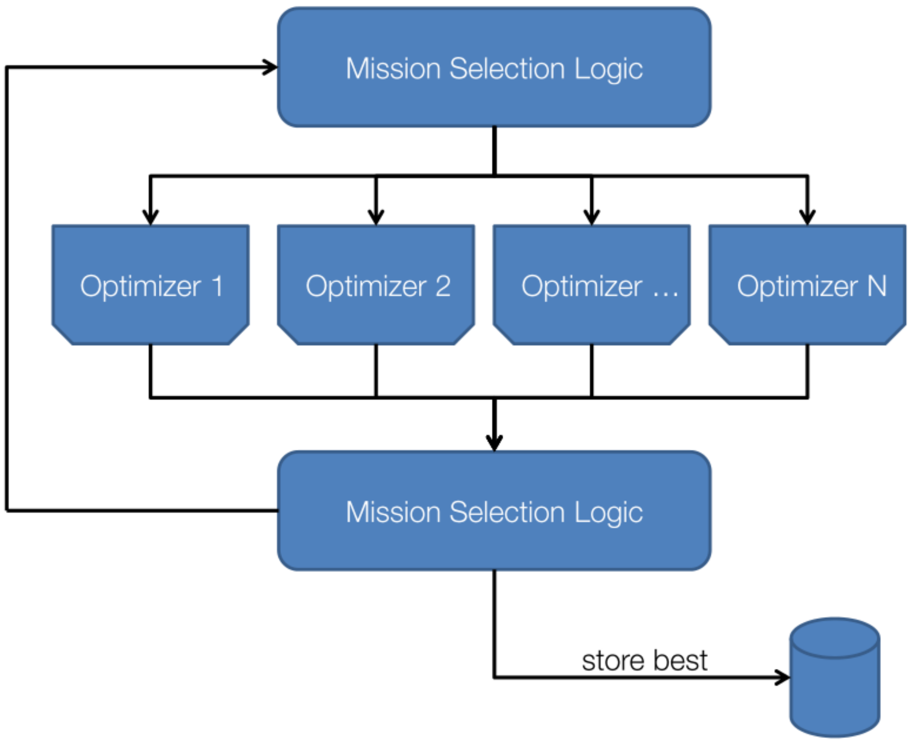


Figure 3.8: Possible use of the parallel optimization ability.

protocol, the Mission Selection Logic must perform a series of handshakes with the PixHawk via the UDP socket (Figure 3.9). However, due to high latencies between the MGS and the aircraft, the MAVLink handshake was not directly possible. A circular buffer algorithm was implemented to reduce the number of latent connections that the waypoints must travel to reach the PixHawk autopilot on-board the aircraft.

A circular buffer stores a fixed amount of data and overwrites the oldest data as new data are provided. This data structure was used because the number of mission waypoints on the PixHawk is difficult to change without erasing the entire flightplan. Therefore a blank 100 waypoint plan is written to the aircraft prior to take-off. This plan is then modified in a rolling fashion to provide the aircraft with a continuous, up-to-date flight plan. The buffer was chosen to be 100 waypoints because it can store a planned segment of roughly 40-80 waypoints and can also be downloaded to the mission planner software in a reasonable amount of time.

The waypoints generated by the mission selection logic are packaged into a Python-specific binary format called a “pickle”. This binary data is sent via TCP to the antenna tracker ground station computer and unpacked. The TCP link ensures that the data arrives in a reliable order and preserves the data integrity despite the latent and lossy cellular connection. A separate software running in the antenna tracker initiates the MAVLink protocol with the aircraft via a 900 MHz link. Although this link is also highly-latent and is prone to packet-loss, it is sufficient for the low data stream required by the waypoint sending protocol. The mission selection logic will wait until the aircraft is clear of the impending waypoint write before sending the next pickle to the antenna tracker. The mission selection initiates the process by sending a MISSION_COUNT command with

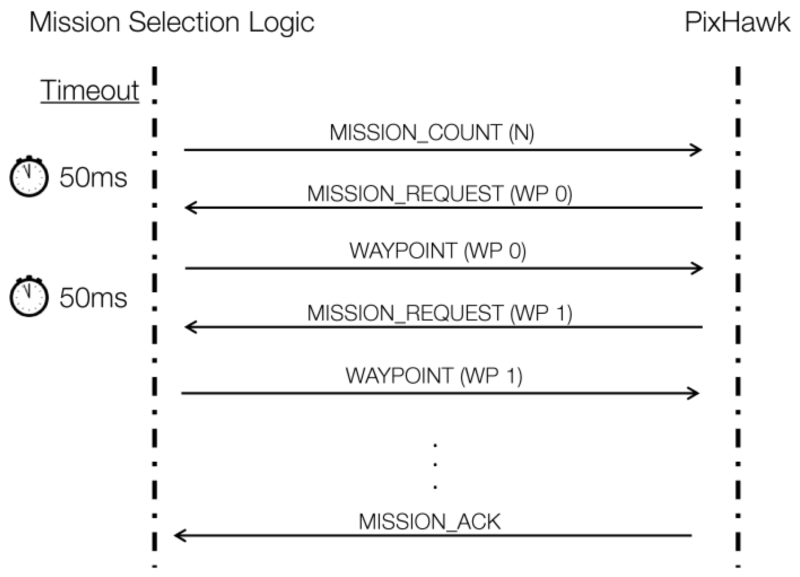


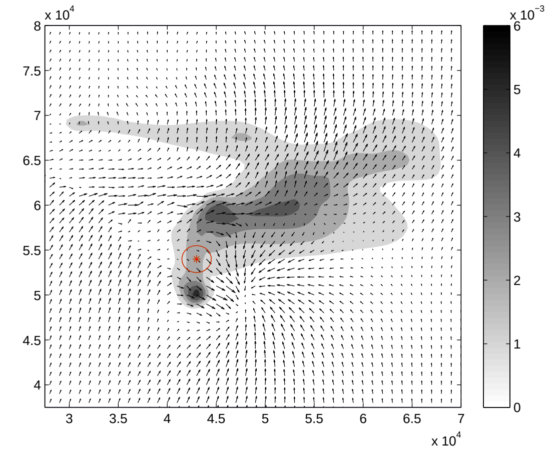
Figure 3.9: Network diagram for the TOL.

an integer designating the number of waypoints it intends to send. The PixHawk acknowledges this request by sending a MISSION_REQUEST with an integer designating the waypoint index to deliver. PixHawk interprets the delivery of a waypoint as an acknowledgement from the mission selection logic and requests the next waypoint. A timeout is set in the selection logic to avoid being trapped in a dead connection. The process is restarted from the beginning once connection has been reestablished.

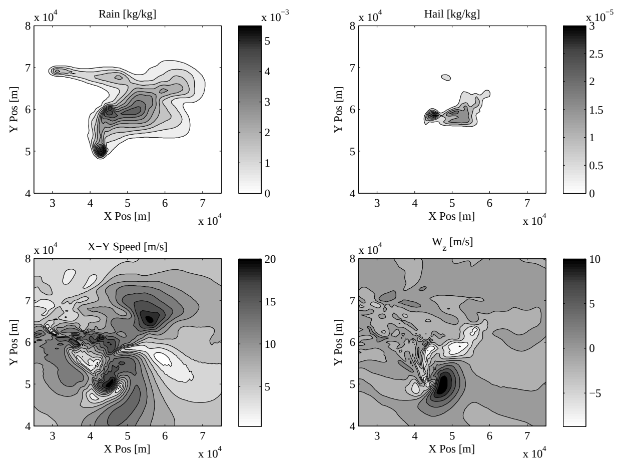
Chapter 4

Assessment in Severe Storm Simulations

This chapter provides results and analysis of the trajectory optimization layer in a simulated storm environment. A set of simulated storm data is used to validate the wind database system and the linear interpolation scheme for the trajectory optimization layer. The simulated storm data sets are provided by Jerry Straka of the School of Meteorology at the University of Oklahoma [19, 8, 7, 12, 13]. The set is planar two-dimensional data for a single instant in time for an altitude



(a) Wind field and precipitation contours from simulated storm data.



(b) Rain, hail, and storm velocities from simulated storm data.

Figure 4.1: Simulated storm data.

of 500 meters. Storm data is generated on a 210 by 210 grid with 500 meter resolution in a moving reference frame that tracks the storm. Data includes rain, hail, and three-dimensional wind velocity components (relative to the storm frame of reference). Atmospheric data such as rain and hail are useful for optimal path planning because they represent avoidance zones during path generation. The storm simulation is aligned such that the positive y-axis corresponds to North. The storm data including rain, hail, planar (x-y) wind speed, and the z-component of wind vector are useful for detecting dense energy regions (Figure 4.1b). The primary purpose of the data will be to provide the trajectory optimization layer with wind magnitudes and directions (Figure 4.1a).

4.0.5 Storm Approach

The guidance mission is simulated in a storm environment using identical optimization parameters and gains detailed in section 3. The simulation assumes that the sUAS approaches the storm cell from the west and traverses to the east to approach the rear-flank downdraft (RFD) (Figure 4.2). This region was selected as a penetration vector into the storm because of the average wind flow towards the region of interest. The RFD contains high velocity winds and could present a challenge for a sUAS autopilot without knowledge of the environment.

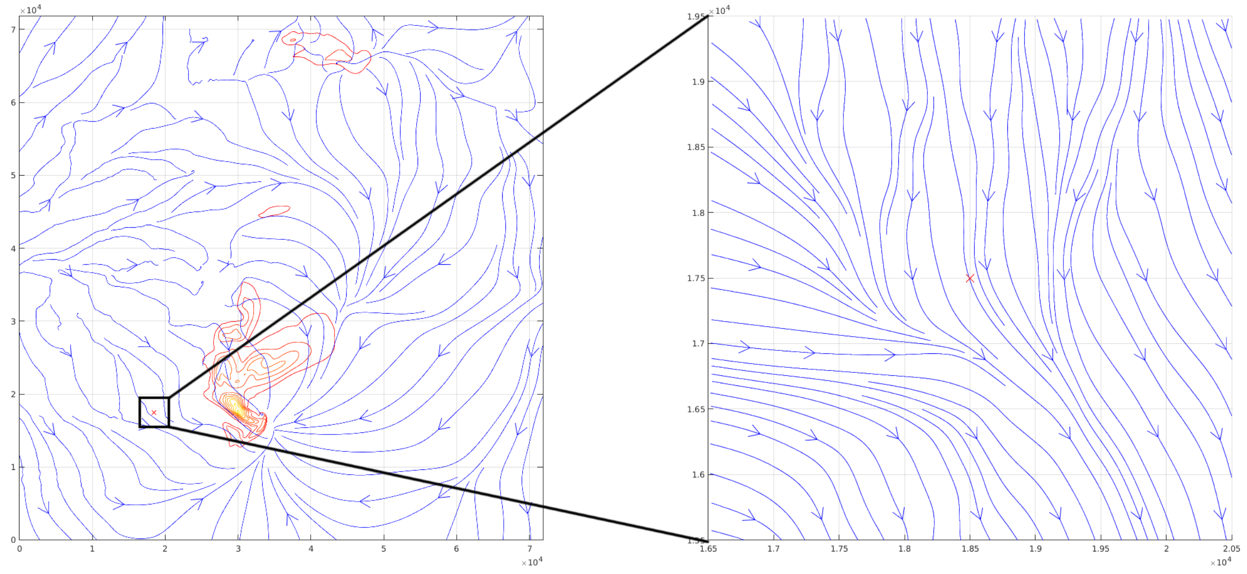


Figure 4.2: Region considered in the storm penetration simulation. Approach is from west to east.

Despite attempting to reach the same goal of 300 meters in the x-direction, the optimal trajectory optimization was divided into three separate legs, each becoming more direct (Figure 4.3). While each optimization was initialized with a 150 meter leg length, the optimization routine shortened each leg until feasibility was satisfied. This demonstrates the adaptability of the TOL with respect to dynamic localized environments. Without user intervention or tuning, the TOL was able to generate three unique optimal trajectories and stitch them together to reach the goal point.

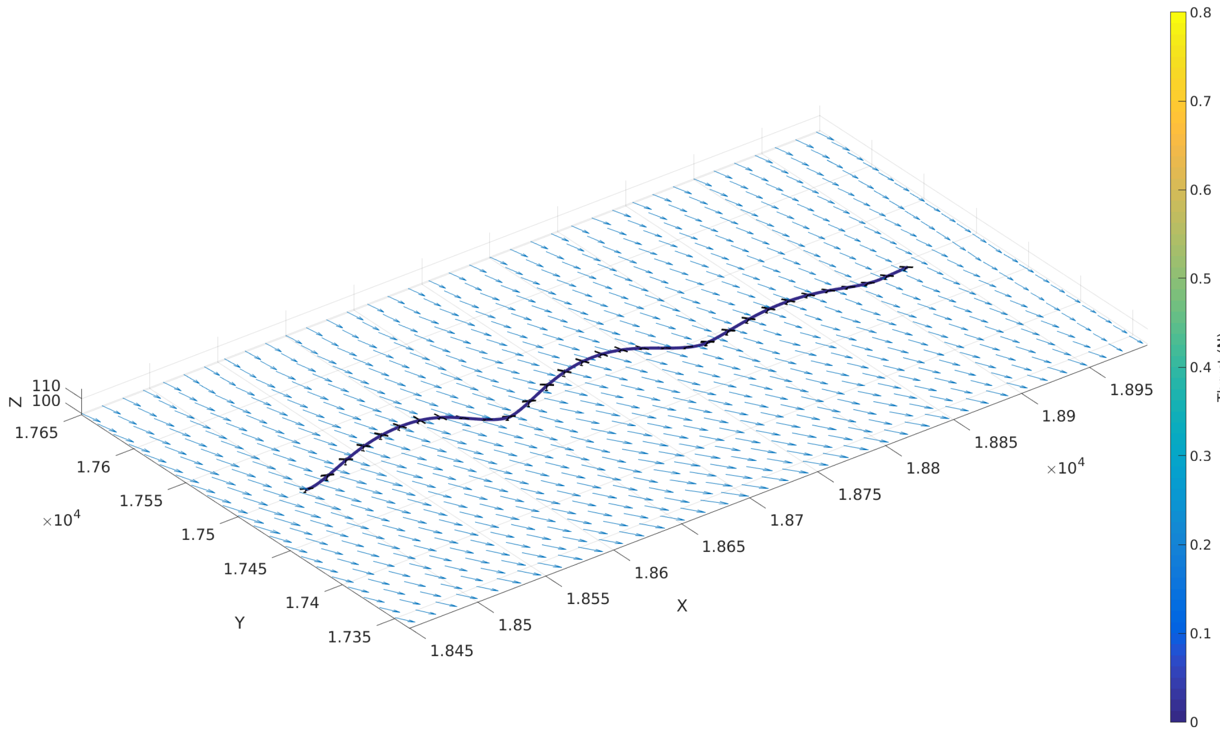


Figure 4.3: A stitched guidance trajectory to penetrate the supercell and enter the RFD. Interpolated wind vectors are shown in blue.

To understand the difference between the optimization behavior in an idealized wind shear and simulated storm data, some properties of the wind field are examined. The average wind shear of the simulated storm data set for the trajectory is $0.156s^{-1}$ which is significantly weaker than the idealized $0.25s^{-1}$ used in validation tests. The differences between the first and last two legs of the trajectory can be explained by the sudden change in vertical wind velocity. The average vertical wind velocity for the first leg is approximately zero meters per second, but this quickly increases to -0.65 meters per second for the final trajectory. The aircraft loses energy throughout the trajectory because of this sinking air however the TOL was constrained to generate a trajectory towards a

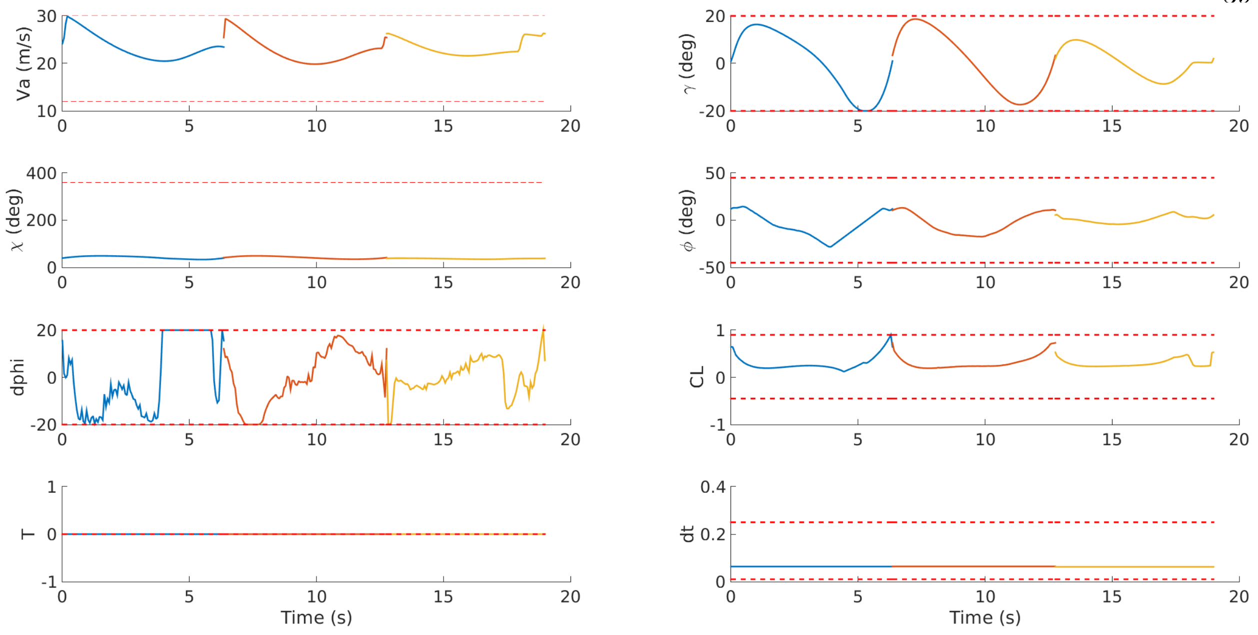


Figure 4.4: The sUAS states for the stitched guidance trajectory. Each color represents an optimized leg. Red dashed lines indicate limits placed upon the optimizer.

pre-defined goal (300 meters in the x-direction) (Figure 4.4). In this trajectory, the sUAS drops airspeed by climbing with a high flight-path angle. Due to the increased flight path angle, the aircraft gains lift and airspeed follows suit. A maximum airspeed constraint forces the trajectory to decrease flight path angle and level off. To satisfy the periodic constraint, the sUAS once again pitches up just before transitioning to the next leg. The flight path angle, γ, demonstrates the effect of an increasing downward wind over each leg. As the downward velocity increases, a high angle of attack will force the sUAS downwards. Therefore, the TOL reduced the flight path angle adjustments to avoid being forced to lose altitude.

4.0.6 Loitering in the RFD

Once the sUAS has successfully navigated to the region of interest for atmospheric sampling, a stitched guidance-loitering mission is engaged. Using the stitching algorithm (Algorithm 2), the TOL plans guidance trajectories directly towards the sampling region and transitions to a loiter trajectory. The region of interest for loitering flight is the rear-flank downdraft (Figure 4.5). Due to the high wind speeds, a zero thrust loiter was not feasible in this region of the storm. While the trajectory will use thrust to complete the loiter loop, the thrust constraint is penalized within the objective function by setting thrust gain, k_T, to 100.

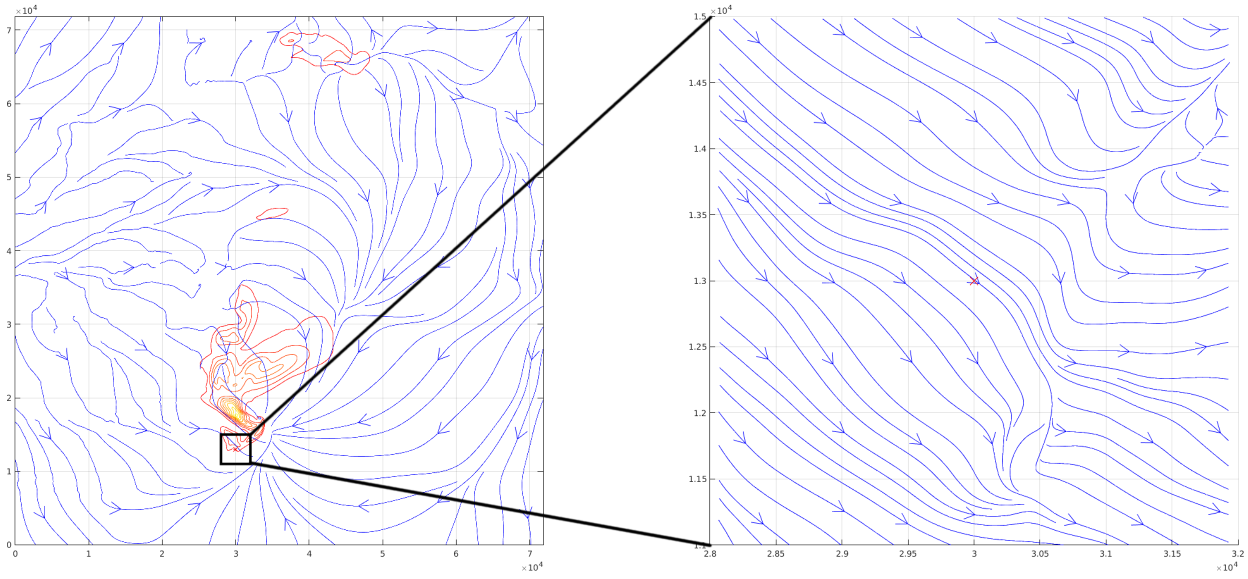


Figure 4.5: Region considered for sampling the RFD. Approach is from west to east.

Although a feasible loitering trajectory for zero-thrust could not be found in this region of the gust front, a small amount of thrust was required to achieve a different feasible solution (Figure 4.6). The loitering segment uses throttle when turning against the wind to prevent drifting out of the loitering radius. This enables the aircraft to also soar up into the wind, gaining potential energy to expend as it makes its return loop. The circular trajectory is expanded to fill the entire loitering radius. This is because less control effort (thrust) is required to fly a larger radius circle. If the k_p gain is increased, the loitering circle becomes smaller at the expense of more thrust used.

This presents a trade-off between loitering in a local region and the endurance of the sUAS. The aircraft kinematics, as well as the current wind environment, will limit the size of a feasible circular trajectory.

Once the sampling mission is satisfied, the sUAS navigates along the return trajectory, planned from the apex of the loitering trajectory to allow the sUAS to seamlessly transition out of the loitering trajectory. This return trajectory differs from the approach guidance segment due to the air-relative aircraft heading. The sUAS proceeds to the start location, soaring into the wind to gain energy. There is no final periodic altitude requirement for the return guidance mission. In the simulation, the sUAS returns to the original x-y start coordinates with an increased altitude of approximately ten meters. This gained potential energy can be expended during its return to launch guidance trajectory.

Observing the evolution of the aircraft states in both the guidance and loitering missions, it

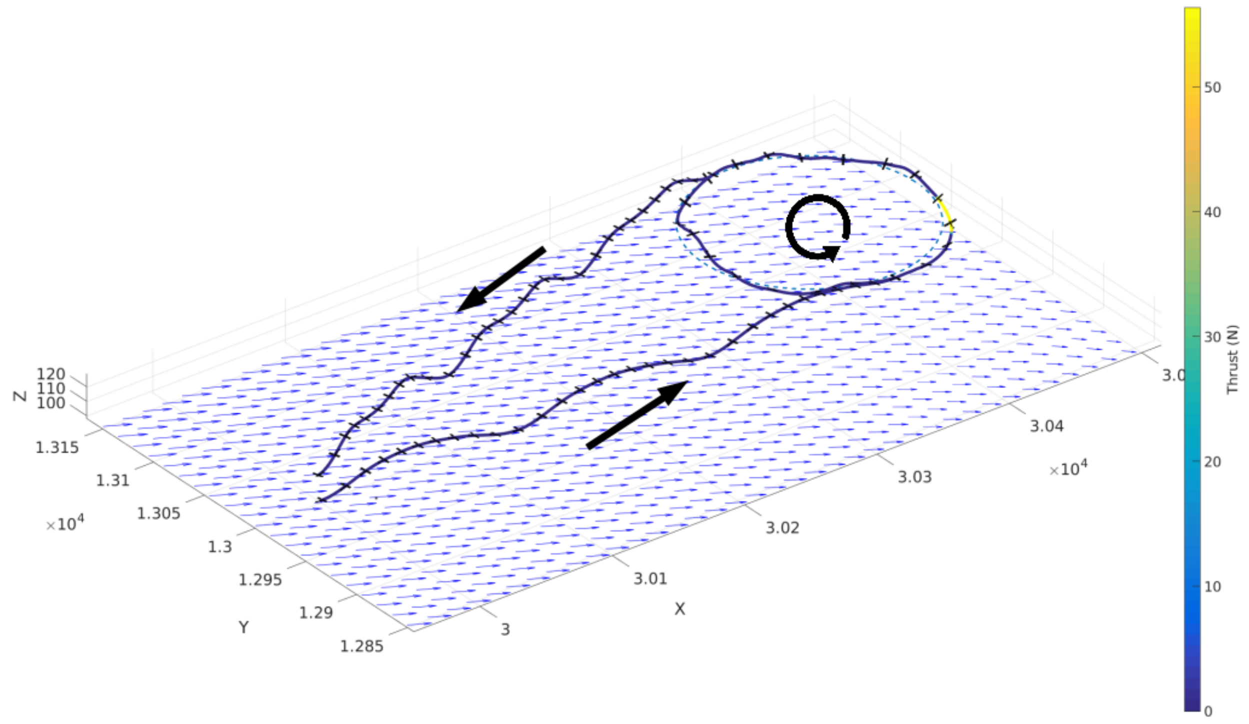


Figure 4.6: Stitched guidance and loitering mission in the RFD. The blue dotted line represents the loitering radius.

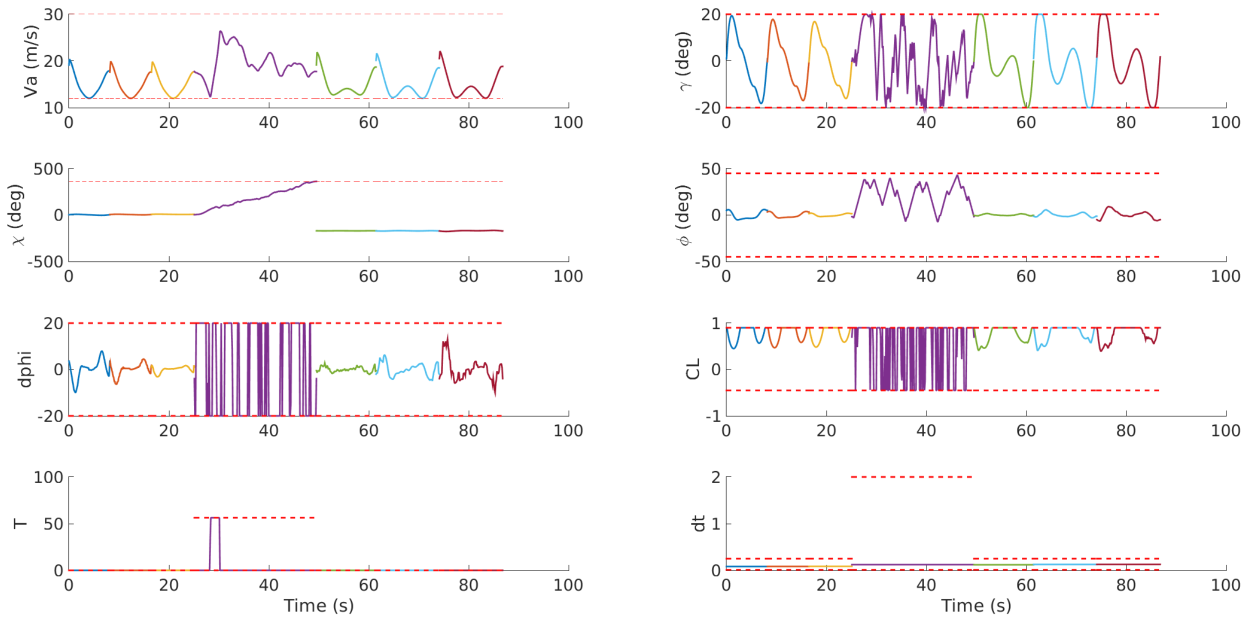


Figure 4.7: sUAS states for the approach, loiter, and return trajectory.

should be noted that there are rapid changes in C_L as well as γ (Figure 4.7). This changes are due to the lack of bounding any rate of change on these states. This could have implications on battery usage from high-frequency changes to the elevator. A suitable controller may decide to perform a smoothing on these inputs to avoid over-working the elevator servos.

Energy optimal trajectories within a simulated storm environment have been examined. Insight was developed on approaching the RFD to conserve the greatest amount of energy. A fully stitched guidance and loitering trajectory was generated and assessed. The guidance mission was investigated independently at first and then combined with a loiter trajectory near the observation goal. The trajectory generated demonstrated that it may be feasible to plan an energy-aware trajectory online during a severe storm UAS mission. To measure the performance of such a system requires further investigation and simulation.

Chapter 5

Flight Results

This chapter provides a flight assessment of the trajectory optimization layer. Flight experiments to assess the EA-DDDAS framework were performed in June 2015 operating near the Reese Airfield in Lubbock, Texas. The objectives of the deployment were to fully connect the EA-DDDAS from end-to-end and perform a full system test. Objectives also included evaluating the performance of each EA-DDDAS component individually operating within different configurations and environments. The Trajectory Optimization Layer was successfully connected with the Lattice Planner, AMOP, and aircraft. The Tempest sUAS was used as the flight platform, fitted with a 3DR PixHawk autopilot performing flight guidance and control. Several loitering trajectories were planned and executed by the aircraft during the several days of flying. Stitched guidance trajectories were also flown nearly the entire length of the airfield to demonstrate the functionality of the lattice planner guiding the TOL. This section will discuss three independent loitering trajectories that were planned and flown as well as a stitched guidance trajectory flown diagonally across the airfield (Figure 5.1).

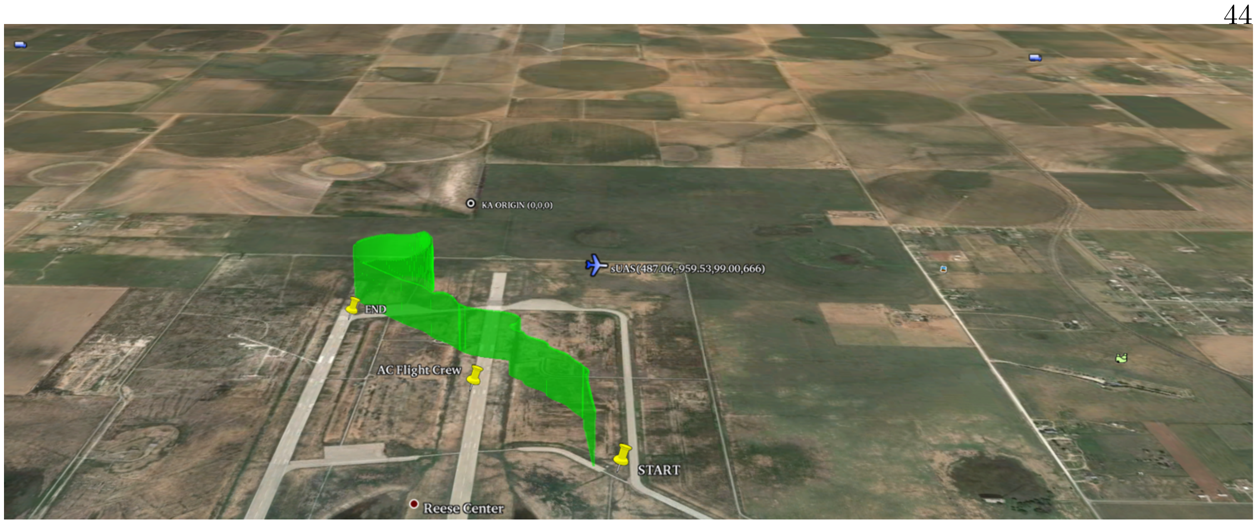


Figure 5.1: The Google Earth interface for the TOL.

5.1 Loiter 1 - 6/23/2015 10:41 AM

5.1.1 SNOPT Planned Trajectory

The first loitering mission of the day was designed to loiter within approximately a 100 meter radius at 100 meters altitude relative to the ground. Measured winds were relatively calm with a consistent flow out of the south heading north at a 7 meters per second average (Figure 5.2). Any wind gradient was virtually non-existent within flight altitudes. Measured shears in any direction of wind were less than 0.01 s^{-1} (Figure 5.3). The cost function used for the loitering missions were dependent on thrust only and initialized with a 100 meter radius circle. The planned trajectory deviated greatly from the initial seeded trajectory by aligning the longer legs at approximately a 45 degree angle into and out of the winds (Figure 5.4). The trajectory is colored to represent the amount of instantaneous thrust used. The planned cost value is the resultant objective function value, in this case $\frac{1}{2}T^2$ for all collocation points. This allowed the aircraft to perform cyclical porpoising motions as it flew upwind, taking advantage of the south to north wind flow. The northeast corner of the trajectory also appears to be following a dynamic soaring trajectory by climbing up into the wind and descending back out of the wind. Maximum theoretical thrust for the trajectory never exceeded four Newtons (7.15% of maximum thrust for the Tempest, Figure

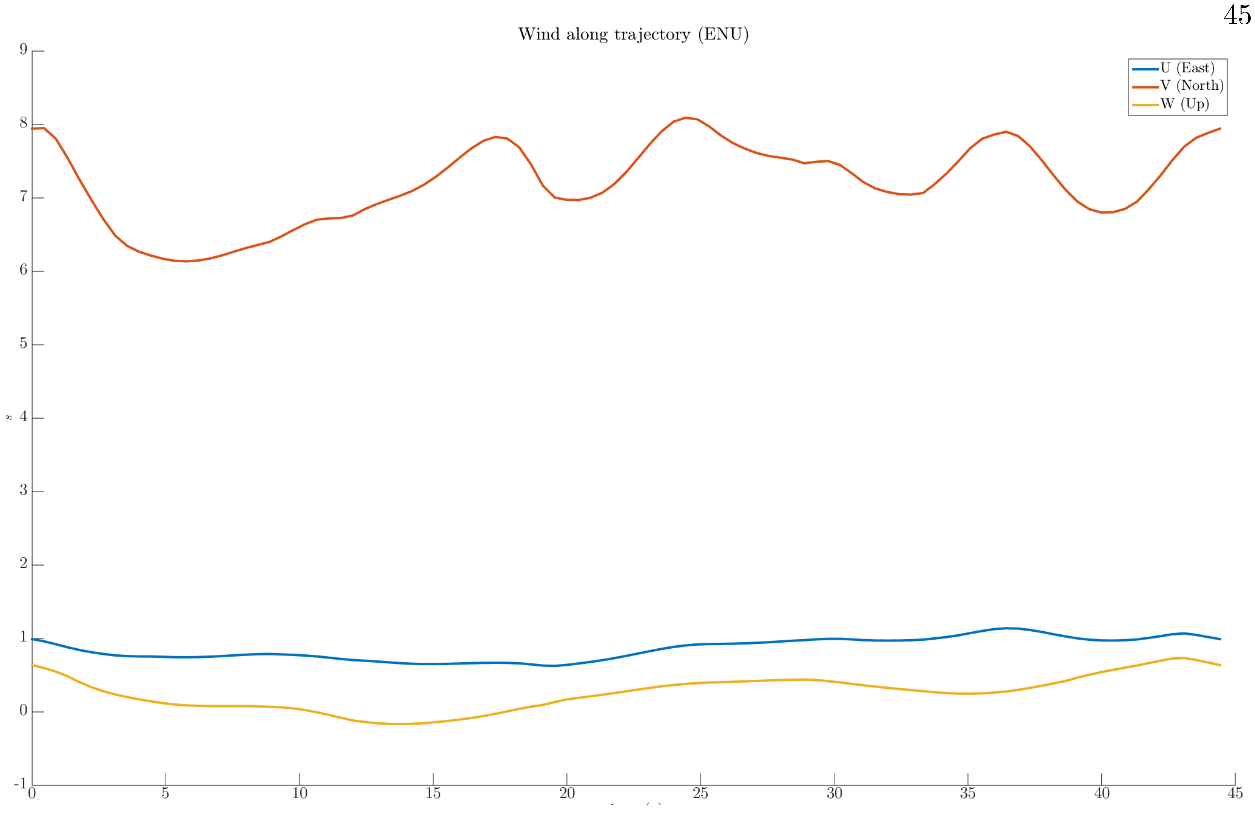


Figure 5.2: Interpolated winds during the planned aircraft trajectory.

5.5). The coefficient of lift and roll rate states are often reaching their imposed limits while the airspeed remains as low as allowed. The aircraft pitch and associated flight path angle modulate at the same period as the airspeed cycle explaining the porpoising action of the trajectory. The aircraft climbs until a minimum airspeed is reached and then descends again gaining airspeed. This cycle could conceivably save throttle usage by gliding into the wind, exchanging kinetic and potential energy while making forward progress (Figure 5.6). Evaluating the contributing components to the air-relative energy of the aircraft, the dynamic soaring term is nearly zero (Figure 5.7). Drag is removing energy from the system as expected and thrust conversely adds energy to the system. Due to a vertical wind, albeit relatively low, there is a contribution to the total energy by means of static soaring. For the latter half of the trajectory, this static soaring term brings the net energy rate positive, countering the drag force. This increase in energy coincides with the pseudo-dynamic soaring trajectory planned by the optimizer in the northeast corner of the loiter. The aircraft is

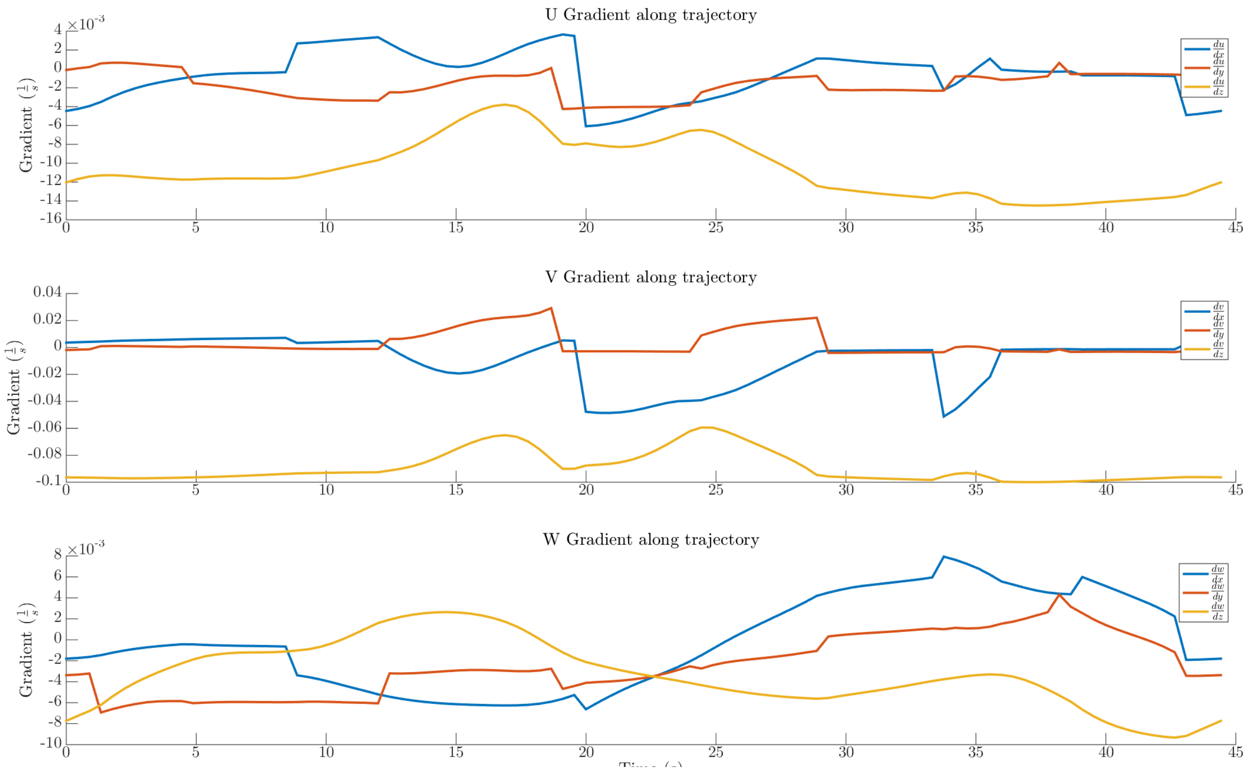


Figure 5.3: Interpolated wind gradients during the planned aircraft trajectory.

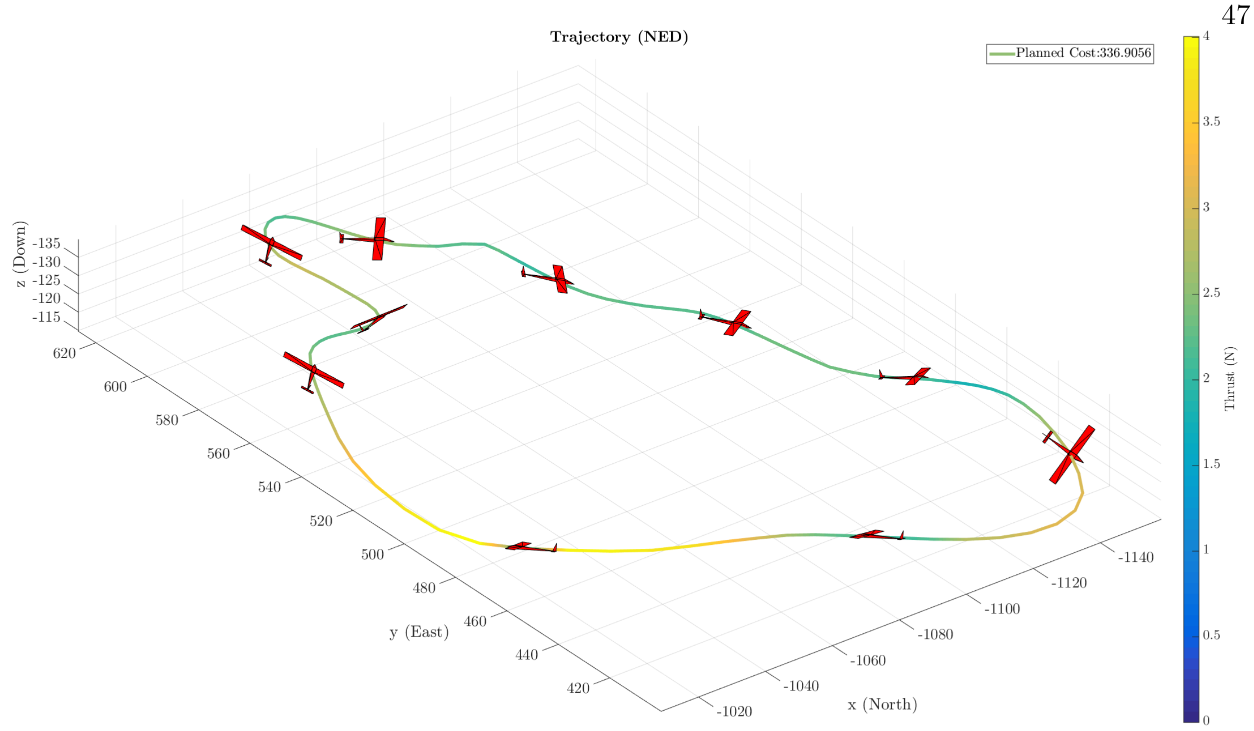


Figure 5.4: SNOPT generated trajectory.

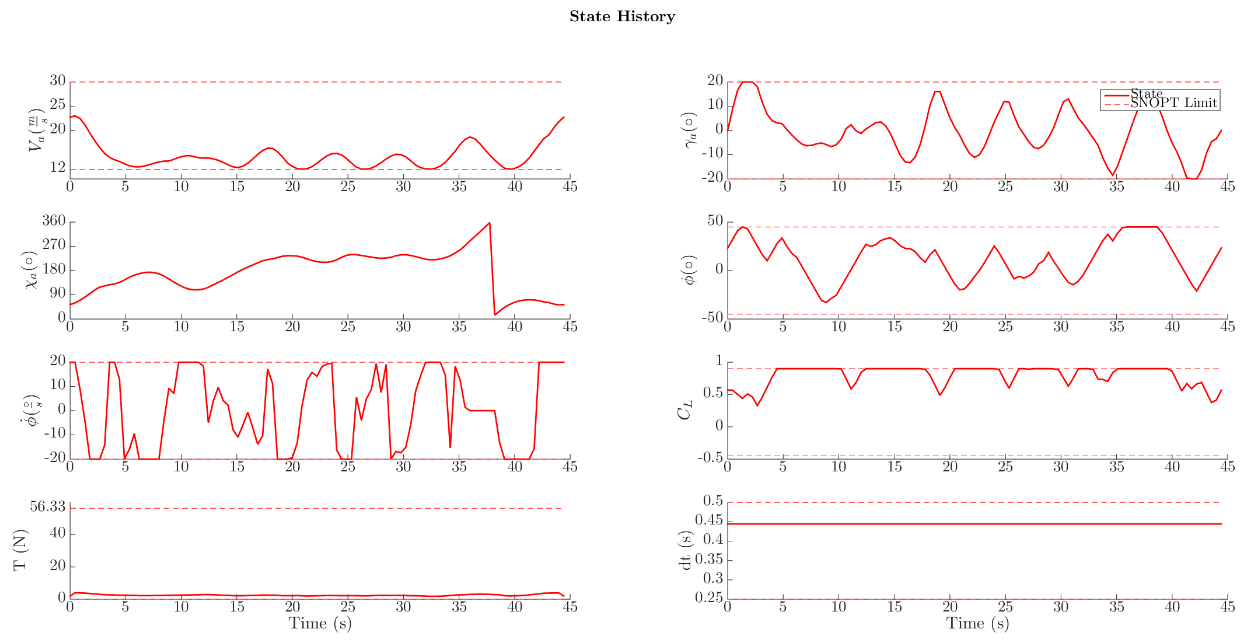


Figure 5.5: The SNOPT planned states.

making turns in this area to conceivably gain as much as energy as possible before passing through

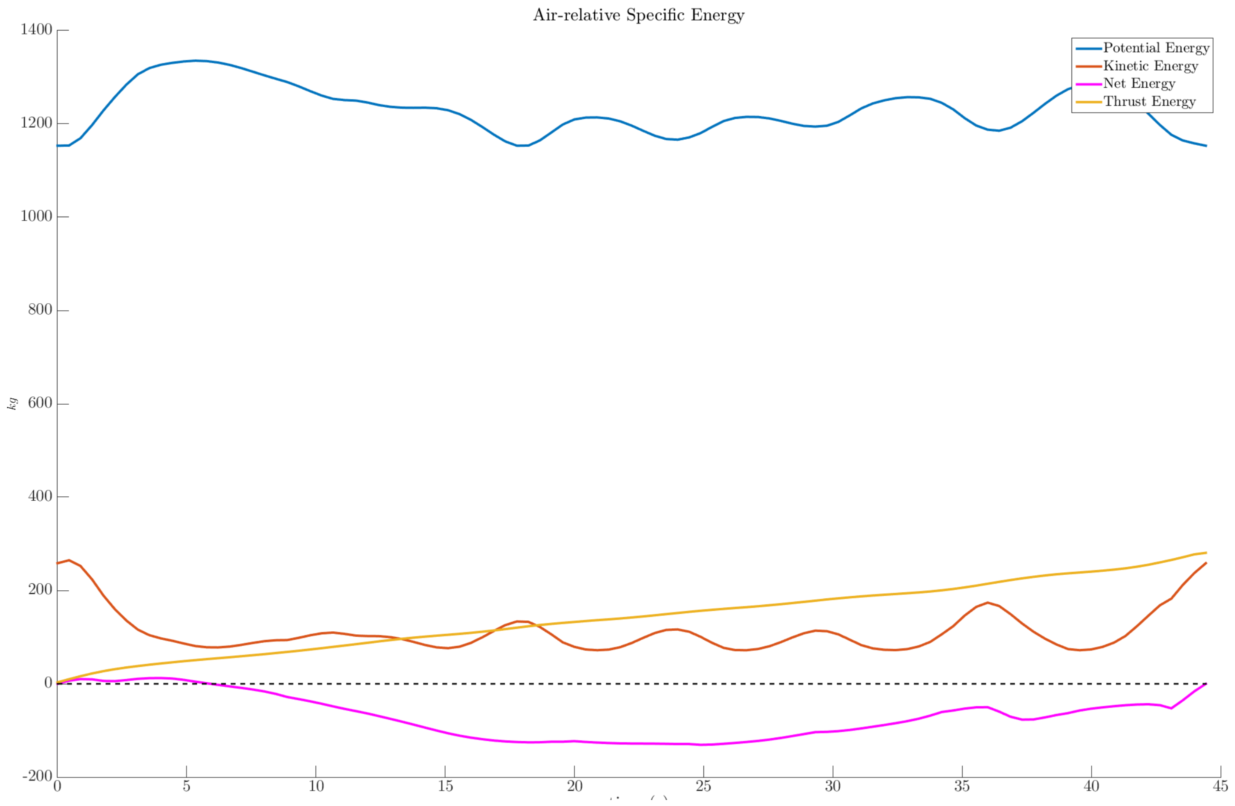


Figure 5.6: Specific air-relative potential and kinetic energy for the planned trajectory.

to lesser vertical motion areas.

5.1.2 Flown and Baseline Trajectories

The trajectory was submitted to the PixHawk autopilot immediately after generation. The assumption was made that the wind environment would remain relatively static between planning and execution of the path. However the plan was generated with forecasted winds from the AMOP in an attempt to predict the wind velocities ahead of time. The baseline flown trajectory was commanded by the flight crew to establish a rough comparison of the executed optimized trajectory with a simple circular loiter path. The baseline trajectories were flown at the same altitude and roughly the same location in an attempt to correlate wind velocities between the two trajectories (Figure 5.8). Circular paths were generated by establishing a mission loiter waypoint at the appropriate altitude. The PixHawk autopilot then generates the control inputs to fly a coordinated turn

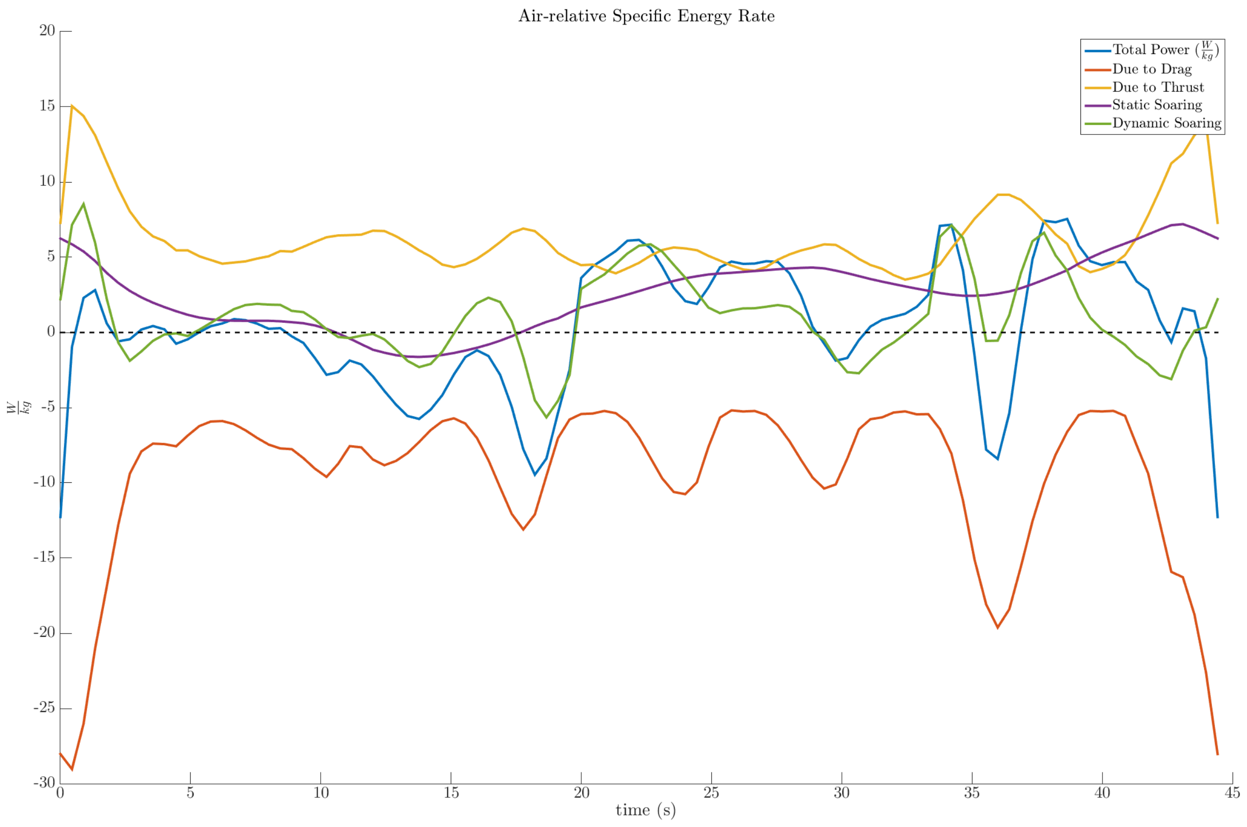


Figure 5.7: Specific air-relative energy-rate for the planned trajectory.

about the point.

The plan was adhered to relatively well by the PixHawk waypoint and speed following controller (Figure 5.8). The autopilot was not attempting to control aircraft pose other than inertial position and airspeed. The controller struggled to maintain tracking during the static soaring portion of the trajectory and instead used considerable amounts of throttle to compensate. The planned and flown states follow the same increasing and decreasing trends (Figure 5.9). The roll rate is does not match closely but the roll matches well, implying that the planned roll rate may benefit by being run through a low pass filter to eliminate noise. For this particular experiment, roll rate was not explicitly tracked so it had no impact on the PixHawk performance. The primary difference between the two states is the thrust. Thrust for the flown trajectory is calculated by dividing the measured power usage from the on-board battery by the corresponding airspeed. This

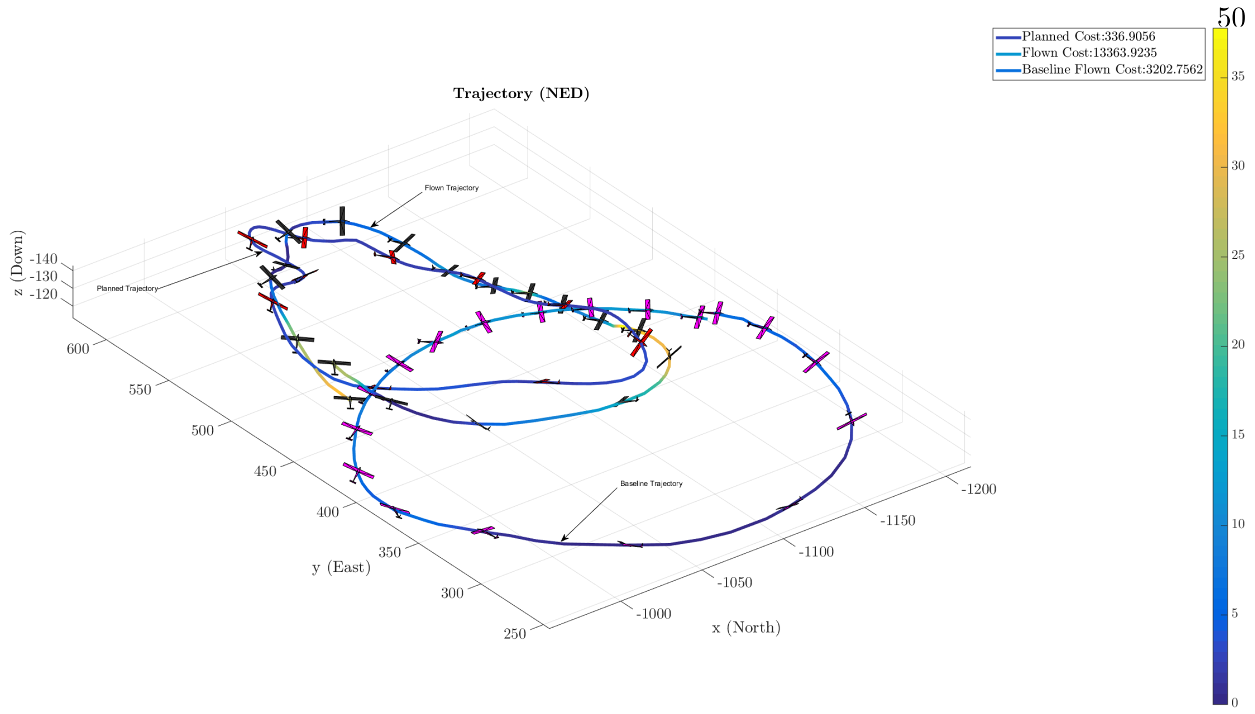


Figure 5.8: Comparison of the planned, flown, and baseline trajectories.

neglects any inefficiencies due to power transfer from the propulsion battery to the propeller and should be taken as a very rough estimate. Nevertheless, the trajectory cost is significantly higher for the flown trajectory compared to the planned trajectory. This could be largely attributed to the type of control scheme used by the PixHawk to track points in inertial space. Large amounts of control effort were expended by the autopilot in an attempt to reduce cross-track error on the planned trajectory. This would lead to large bursts of throttle usage, raising the measured cost considerably. Another reason for the increased throttle usage could be an insufficient drag model as part of the three dimensional point mass assumption. If significant drag contributions were neglected, additional thrust would be required to compensate.

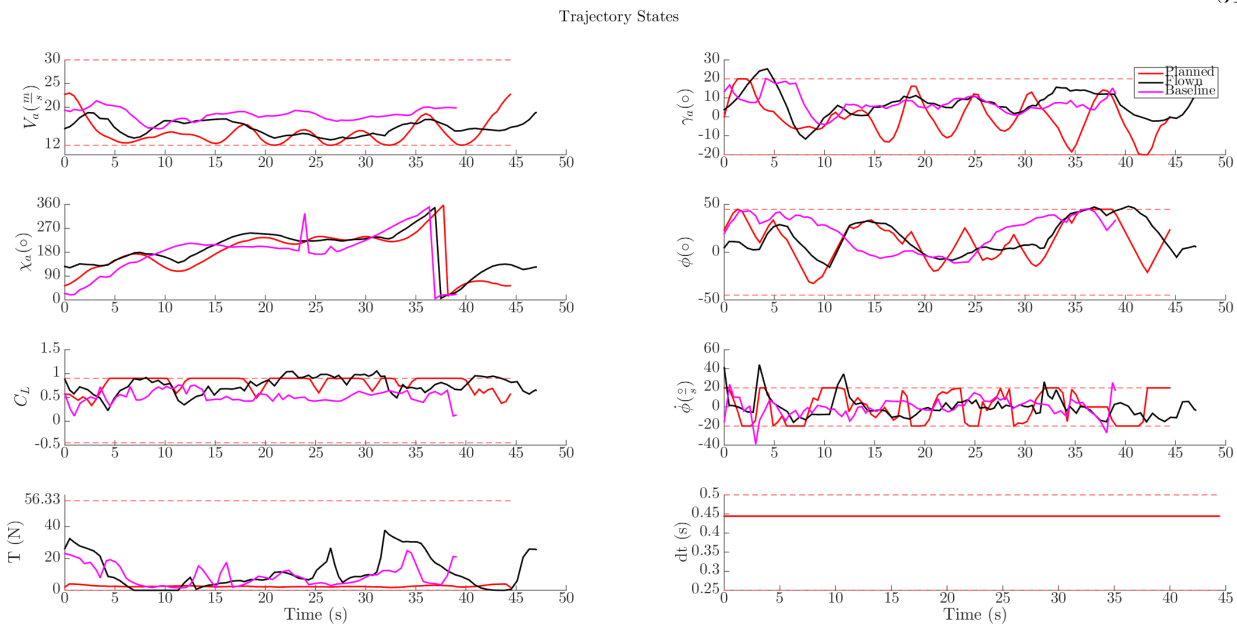


Figure 5.9: Comparison of the planned, flown, and baseline states.

5.2 Loiter 2 - 6/23/2015 11:31 AM

5.2.1 SNOPT Planned Trajectory

Later in the same morning on the same day as the first test, additional tests were performed with adjusted AMOP and Dual-Doppler configurations. The result also reflects the adaptability of the system to generate energy optimal paths over short-periods of time, assuming new wind field information is available. A substantial vertical wind component was present around this time, while the wind remained at a steady nine to ten meters per second out of the South (Figure 5.10). East-West winds were nearly non-existent. Unfortunately, the wind shear was calm in all directions as well, preventing any chance of generating a dynamic soaring trajectory (Figure 5.11).

The SNOPT planned trajectory optimized the trajectory within the given wind field and converged upon a zero thrust trajectory with a cost of 4.9E-7 (Figure 5.12). The trajectory increases in altitude near the North end turn-around, indicating an increase in potential energy without the use of thrust. The remaining trajectory is relatively flat and is aligned with direction of the South-North winds. This trajectory is faster than the previous with an airspeed average of 23.4 meters

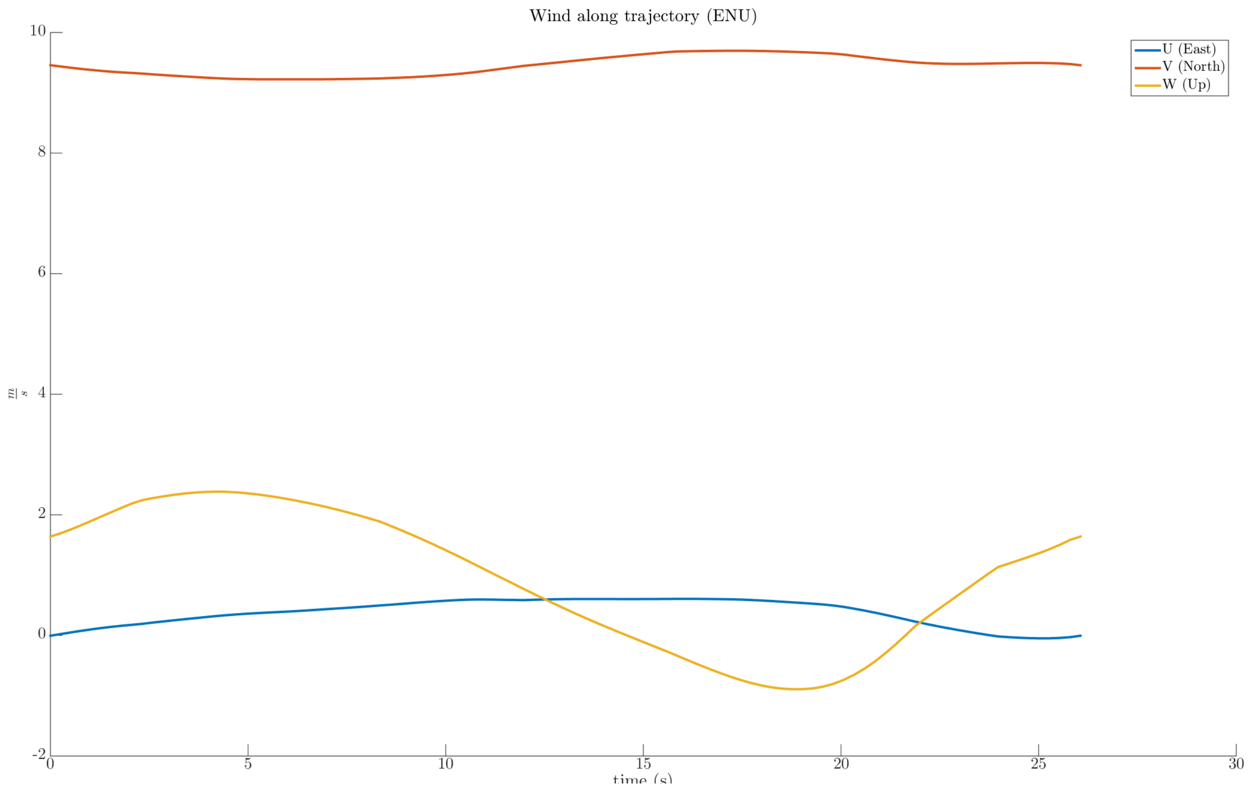


Figure 5.10: Interpolated winds during the planned aircraft trajectory.

per second (Figure 5.13). This is also reflected in the decreased coefficient of lift and less periodic longitudinal motion indicted by the steadier flight path angle. A small Δt was also selected by the optimizer to reduce the trajectory time length.

The kinetic and potential energy trade-off (Figure 5.14) reflects the simple race-track nature of the trajectory. The aircraft initially loses kinetic energy as it climbs but gains a net positive energy due to the upwards air motion. The aircraft then descends regaining air-relative kinetic energy, but loses it quickly due to the increased tail-wind.

The specific energy rate again reveals the lack of any dynamic soaring contribution due to the small gradient in the wind field (Figure 5.15). However, a large contribution from the static soaring term sufficient to counter the drag force confirms the upwards motion on the North end of the trajectory. Assuming this vertical wind motion remained, the aircraft could conceivably remain in this area indefinitely.

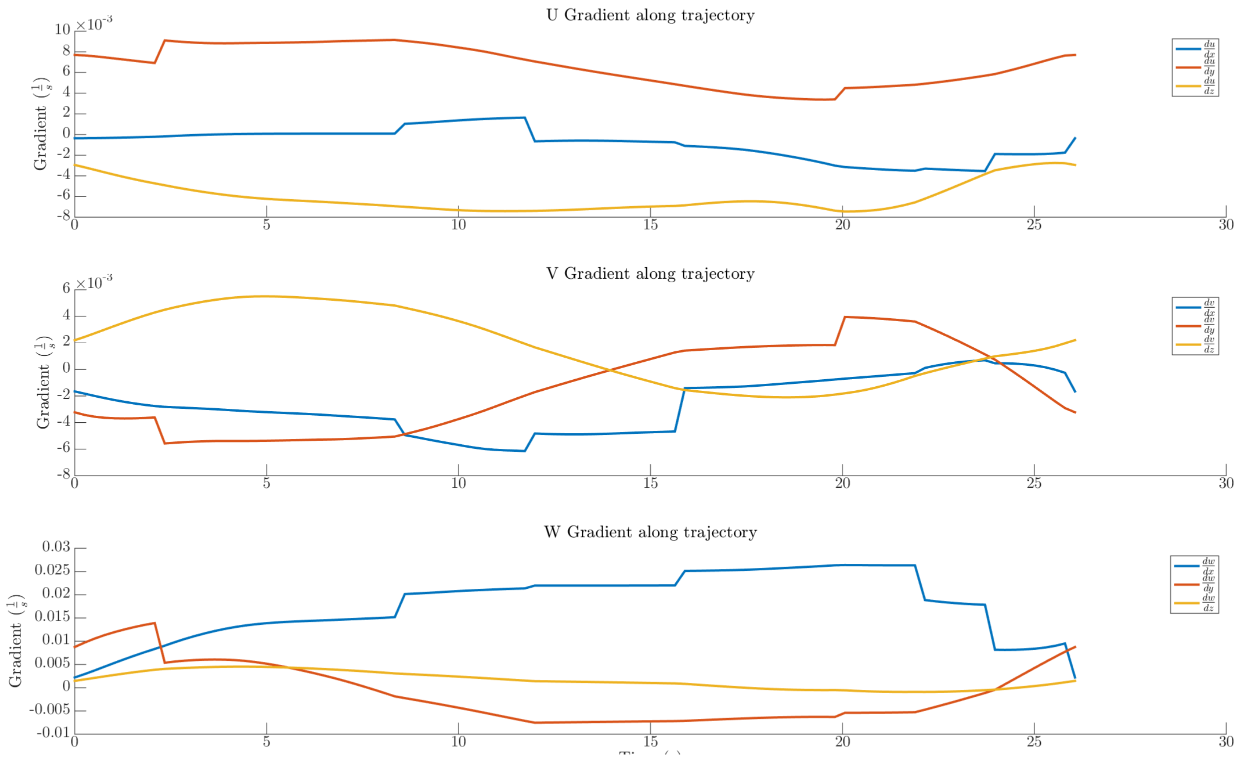


Figure 5.11: Interpolated wind gradients during the planned aircraft trajectory.

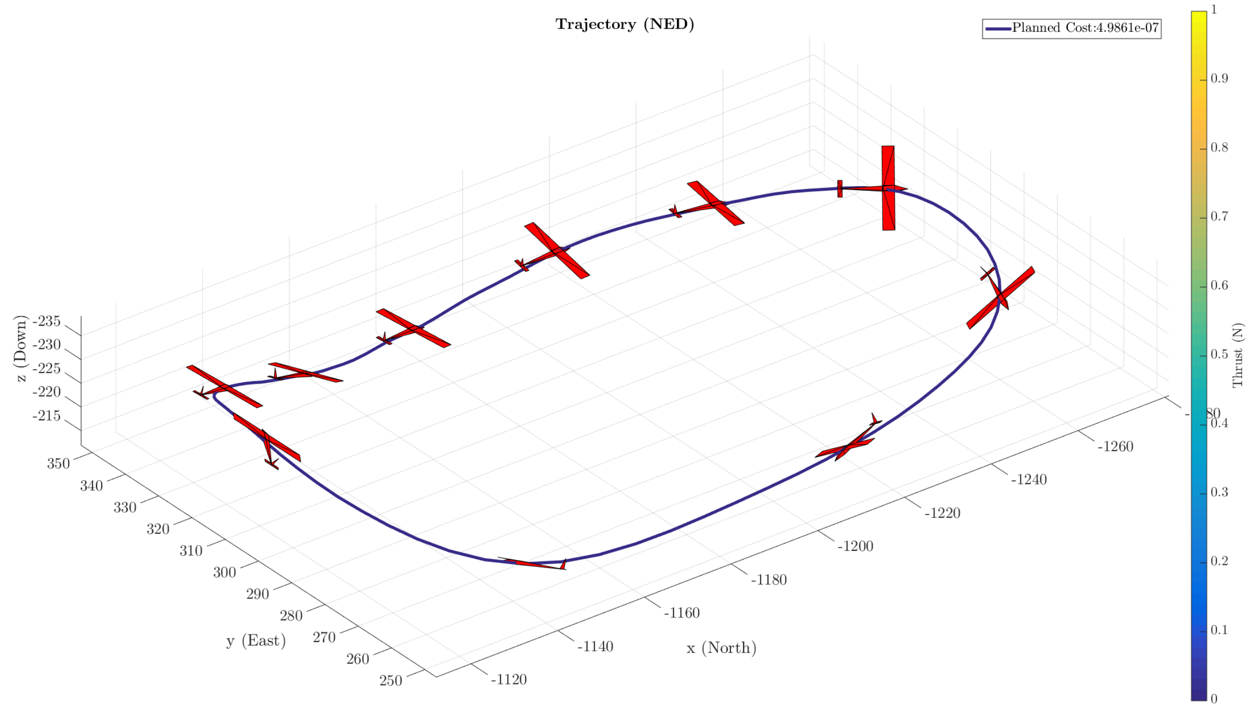


Figure 5.12: SNOPT generated trajectory.

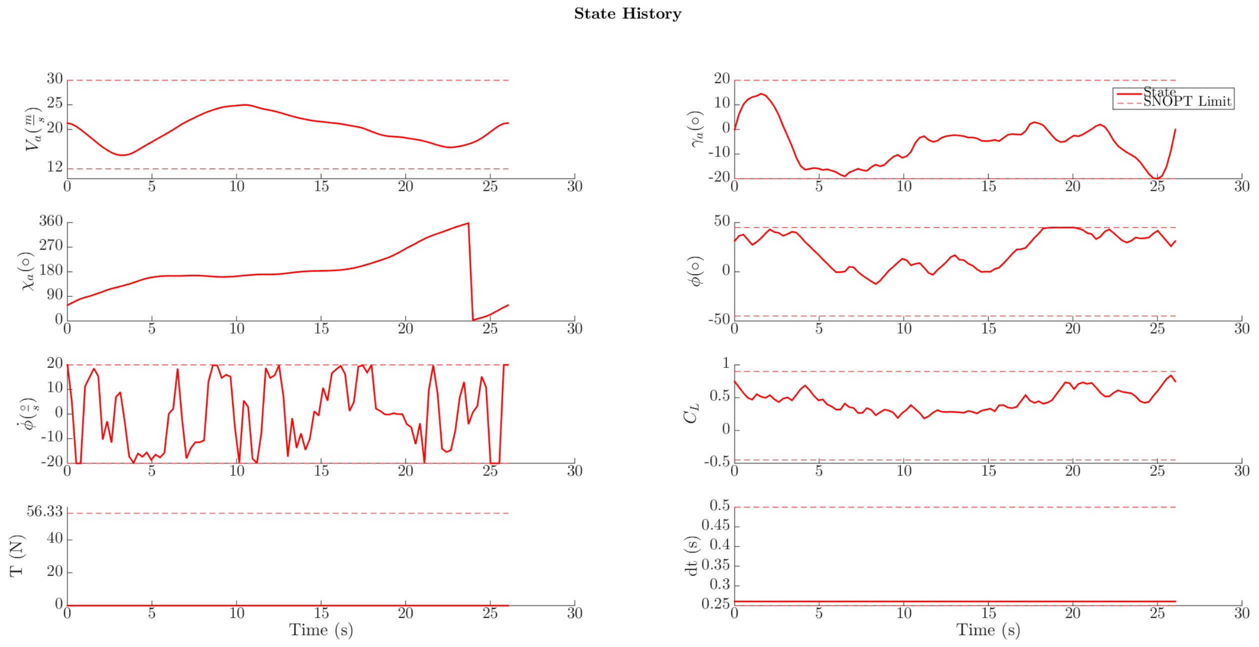


Figure 5.13: The SNOPT planned states.

5.2.2 Flown and Baseline Trajectories

Despite the encouraging zero-thrust trajectory, the baseline was a little less than four times more efficient than the flown trajectory (Figure 5.16). Similarly to Loiter case 1, the trajectory

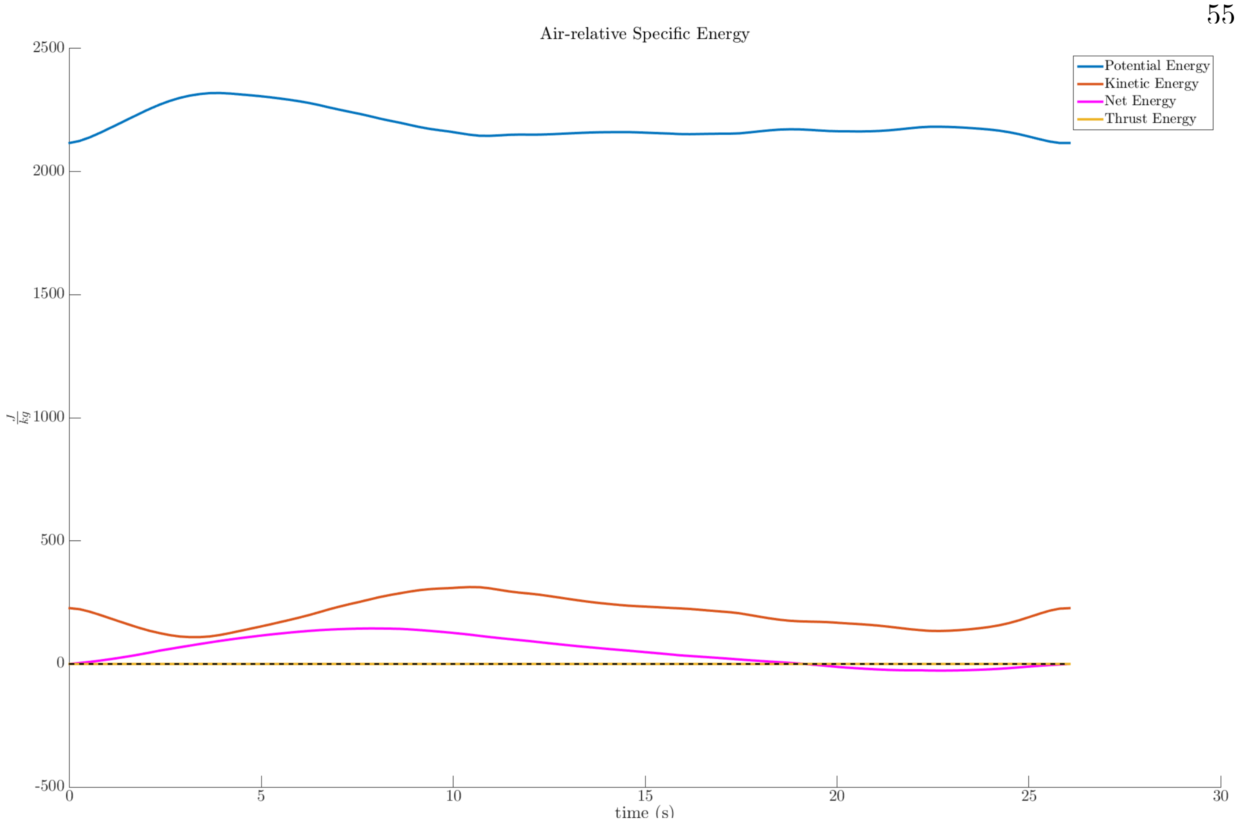


Figure 5.14: Specific air-relative potential and kinetic energy for the planned trajectory.

appears to be too aggressive for the PixHawk waypoint following controller to manage. The large cross-track error leads to increased thrust usage to reduce the distance error from the optimized path. The aircraft does see a minor benefit from the region of vertical wind motion as the amount of thrust required to increase altitude near the North end of the trajectory is reduced. The baseline trajectory maintains a steady, constant velocity and bank angle, allowing for an efficiently flown path (Figure 5.17). The baseline trajectory is also 1.4 times longer than the planned and flown trajectories, indicating that a lower overall airspeed may have been a more prudent choice for efficiency in the given wind environment.

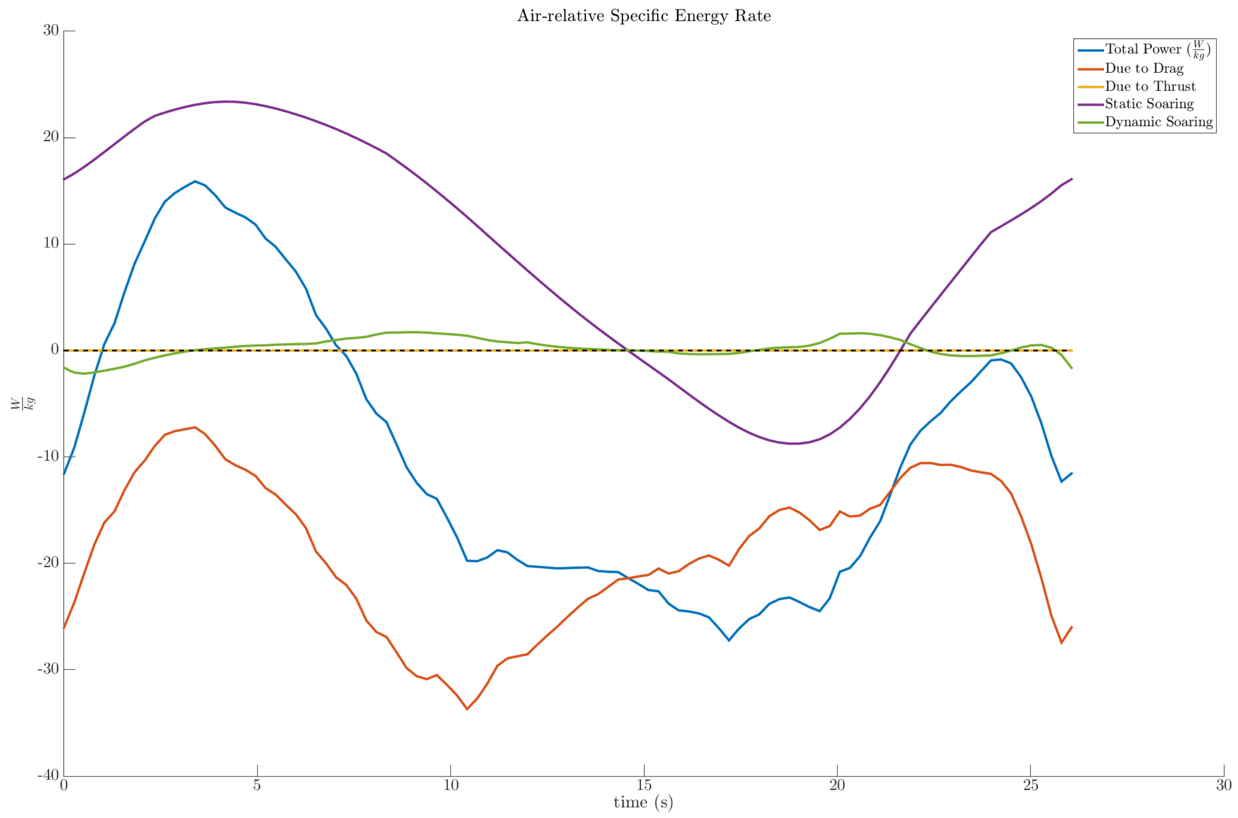


Figure 5.15: Specific air-relative energy-rate for the planned trajectory.

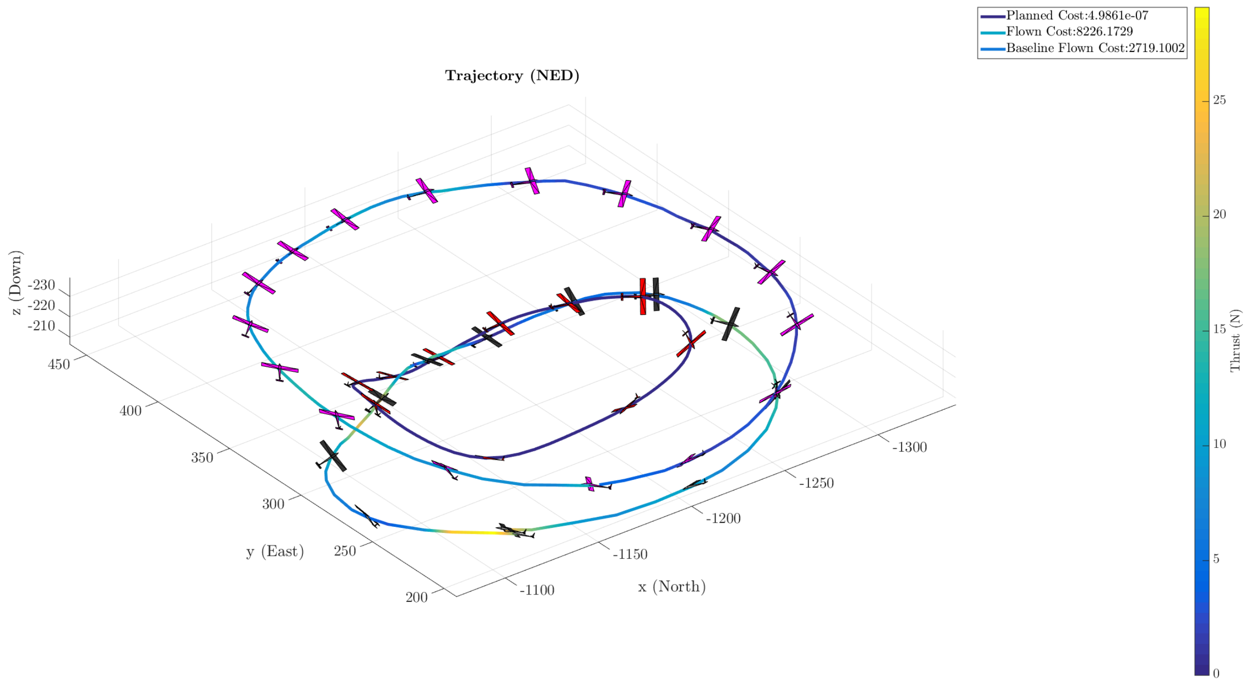


Figure 5.16: Comparison of the planned, flown, and baseline trajectories.

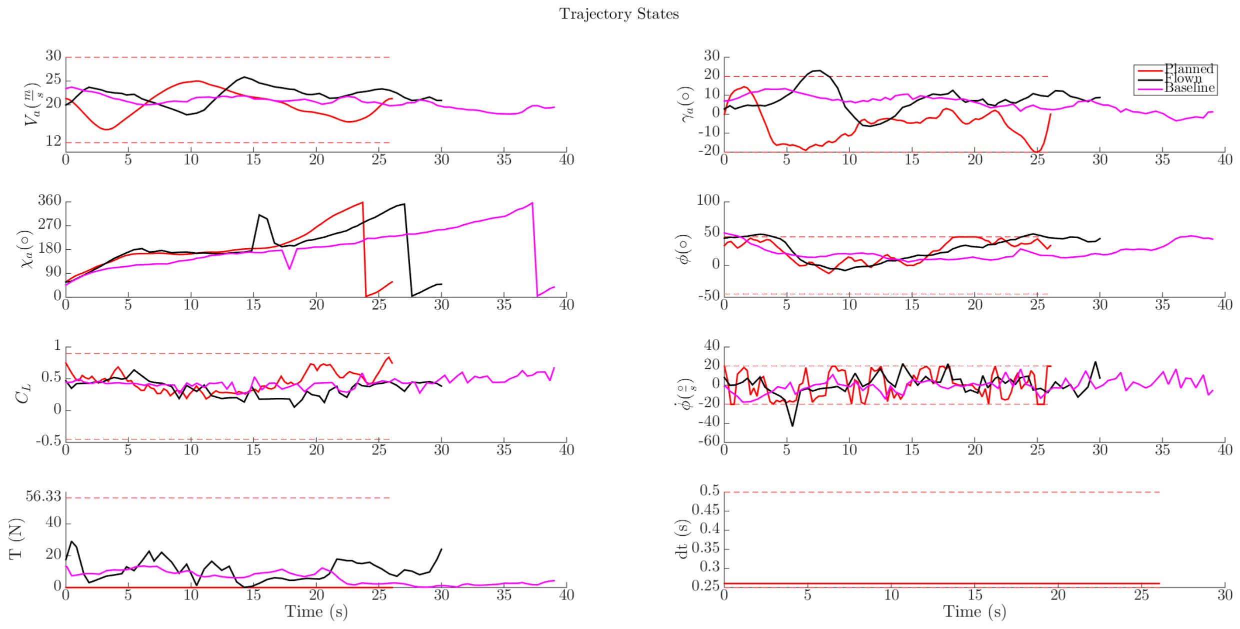


Figure 5.17: Comparison of the planned, flown, and baseline states.

5.3 Loiter 3 - 6/23/2015 11:50 AM

5.3.1 SNOPT Planned Trajectory

For this flight result, the radar performed a velocity-azimuth display (VAD) analysis. This assumes no vertical wind and averages over a complete rotation of the radar to calculate a single horizontal wind velocity at a given altitude, resulting in w_z to be zero (Figure 5.18). Therefore no static soaring can be planned by the aircraft. While dynamic soaring is still possible, the other components of the gradient are near zero as well (Figure 5.19). The airspeed is kept low to preserve thrust (Figure 5.21) while flight path angle is adjusted frequently to maintain a high coefficient of lift. Bank angle is also frequently adjusted, most likely for similar reasons.

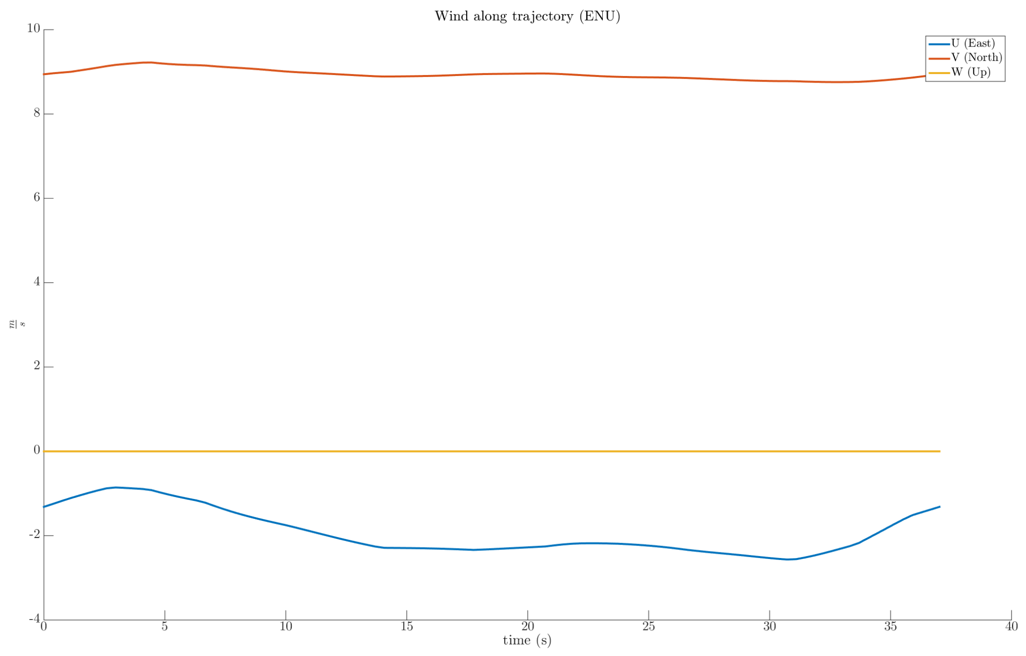


Figure 5.18: Interpolated winds during the planned aircraft trajectory.

The planned trajectory (Figure 5.20) has a low thrust cost total of 34.1 by utilizing the South-North wind to re-gain kinetic energy on the return leg. At the South end of the trajectory, the aircraft turns and banks up into the wind, maximizing the energy gain potential from the

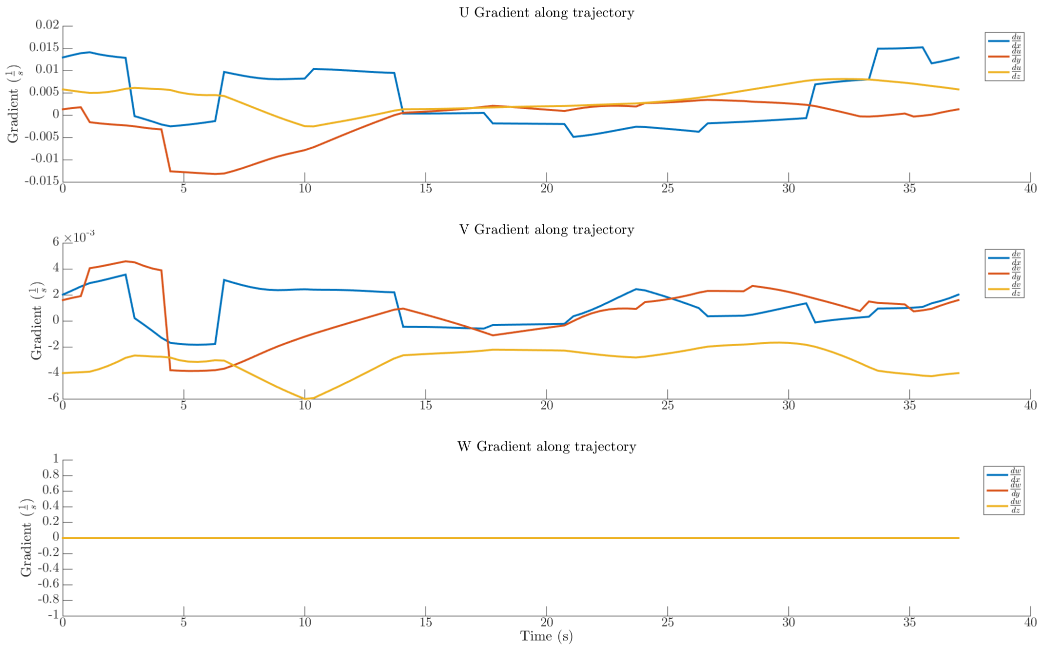


Figure 5.19: Interpolated wind gradients during the planned aircraft trajectory.

oncoming 10 meters per second wind. The remainder of the trajectory is kept short to minimize thrust usage.

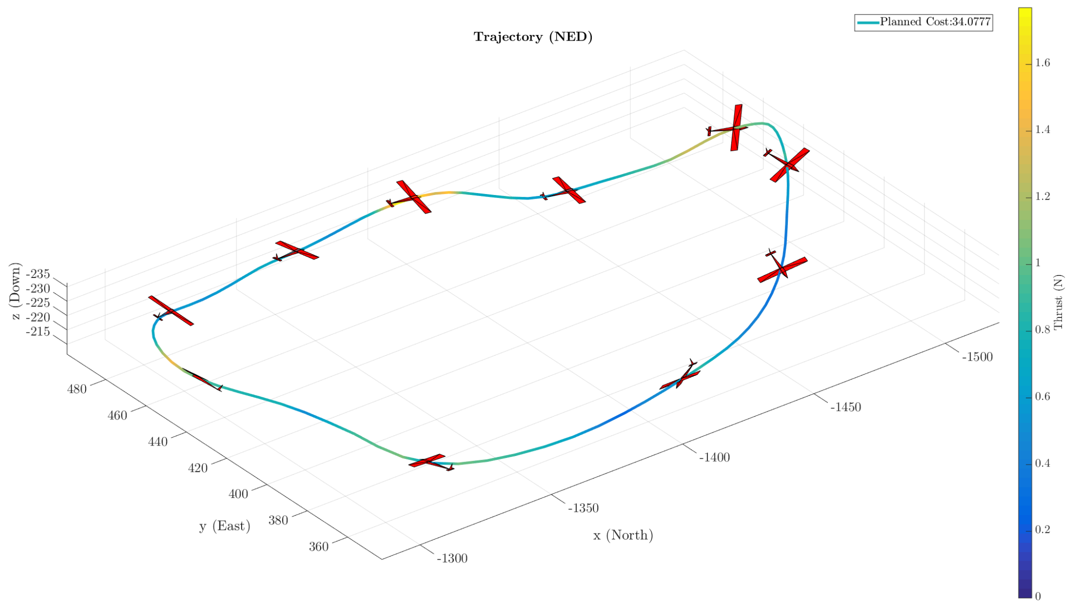


Figure 5.20: SNOPT generated trajectory.

This trajectory's specific energy profile supports the evidence that the aircraft is gaining kinetic energy from the wind to remain energy neutral throughout the flight (Figure 5.22). Besides this and the brief use of throttle to re-gain kinetic energy in order to satisfy the periodic airspeed constraint, the trajectory is trading kinetic for potential energy. Drag is the majority contribution to the energy rate profile (Figure 5.23) and thrust is used sparingly to maintain the minimum allowable airspeed of 12 meters per second. The dynamic and static soaring components of power are nearly zero, as expected from the lack of any substantial wind gradient or vertical wind motion.

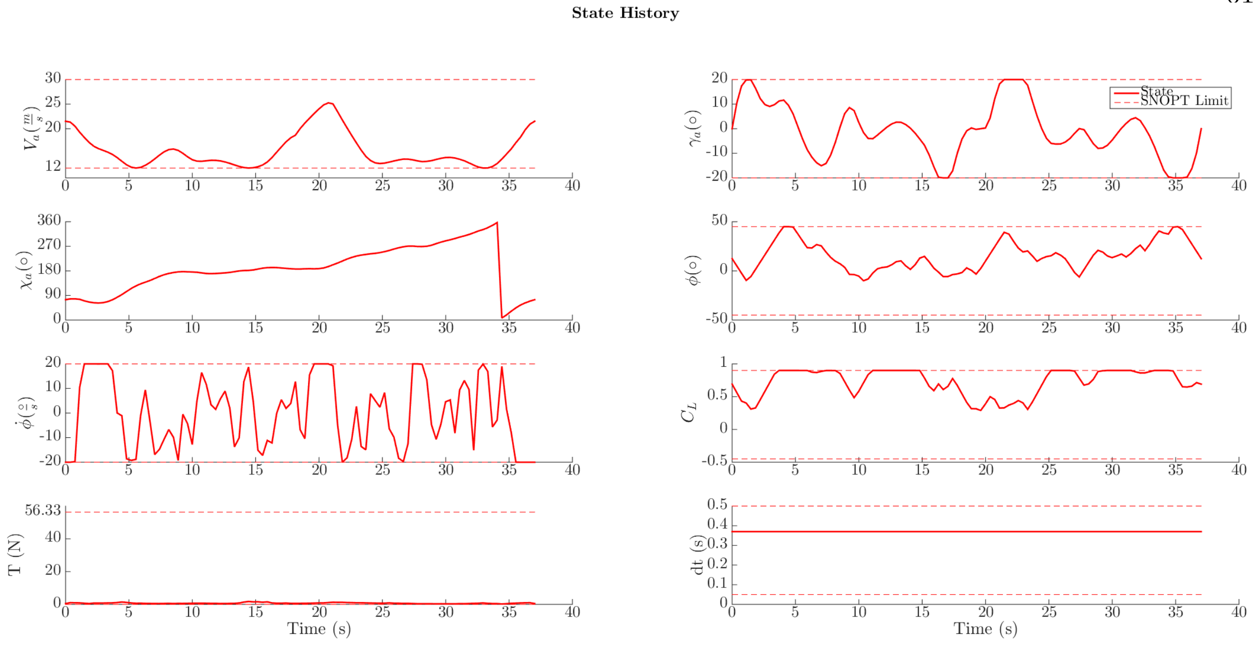


Figure 5.21: The SNOPT planned states.

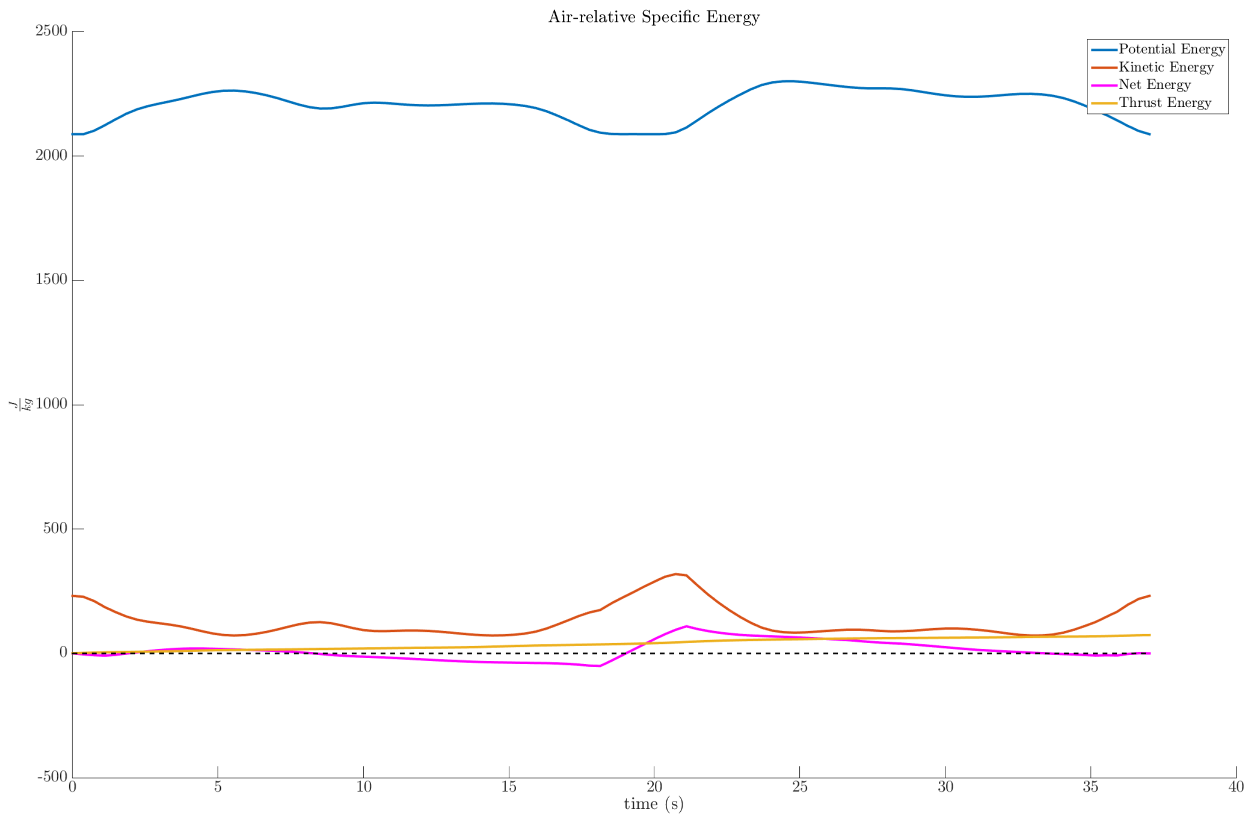


Figure 5.22: Specific air-relative potential and kinetic energy for the planned trajectory.

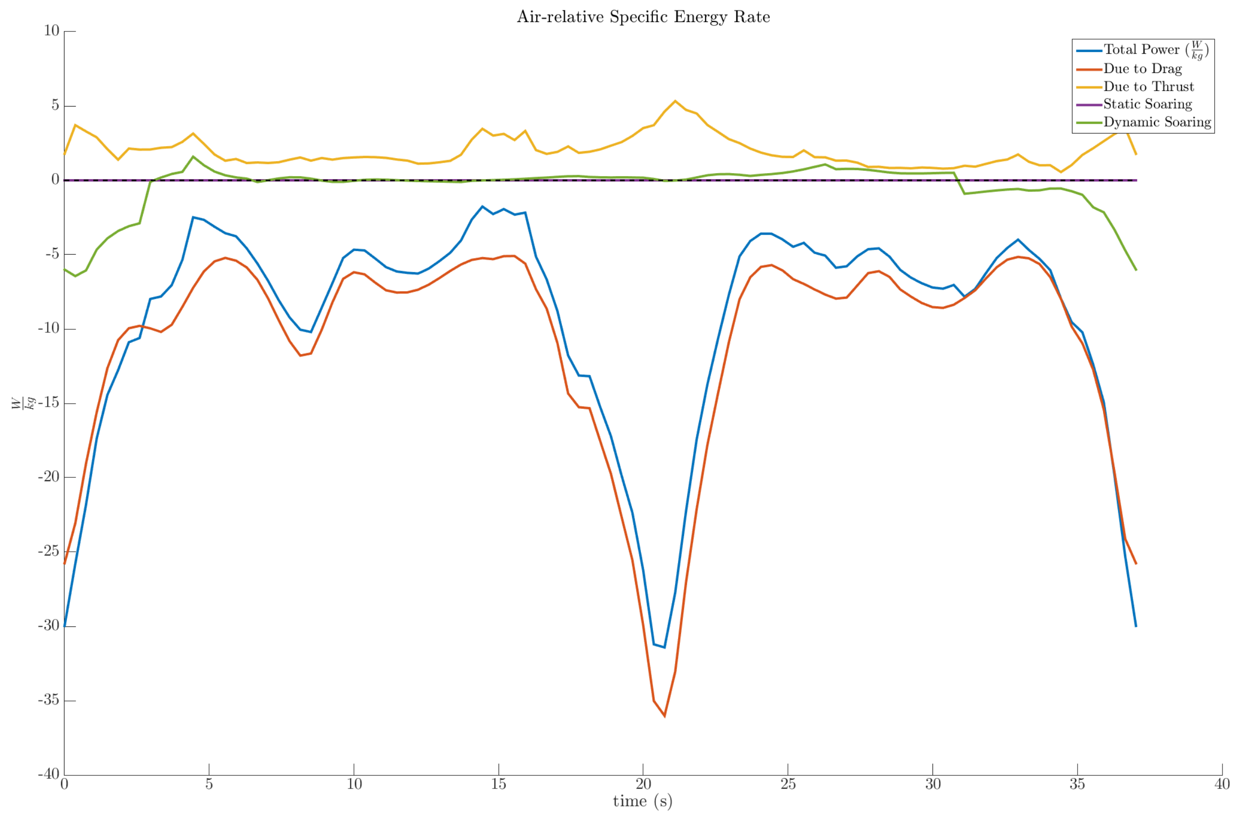


Figure 5.23: Specific air-relative energy-rate for the planned trajectory.

5.3.2 Flown and Baseline Trajectories

The baseline trajectory is less efficient than the flown trajectory with a cost of 2719 and 505.85, respectively (Figure 5.24). The trajectory is followed well with the autopilot capturing the climbing and descending turn on the South end of the trajectory. The flown trajectory also glides for a portion of the path with zero thrust on the Eastern leg as it climbs into the oncoming wind flow (Figure 5.25). In comparison with the baseline trajectory, which struggles to maintain a constant altitude into oncoming flow, the flown trajectory is much more efficient. Despite the lack of a significant wind gradient or vertical air motion, the trajectory optimization layer planned a more efficient trajectory by relaxing the altitude constraints during the flight, while the end constraints were still held to a periodic condition. As a result, the trajectory is approximately five times more efficient than flying a constant altitude circle.

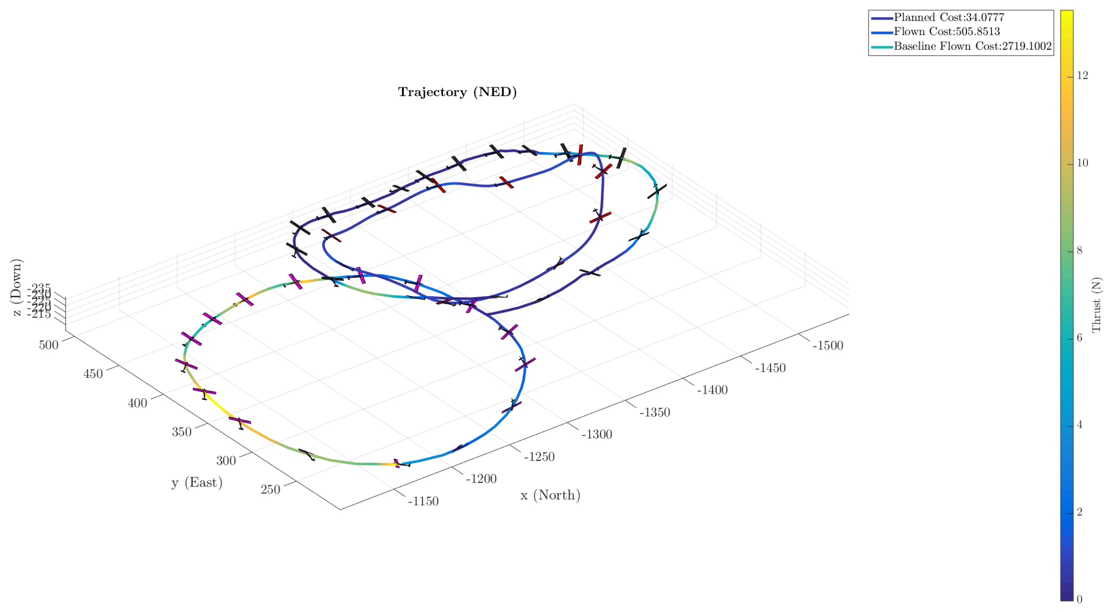


Figure 5.24: Comparison of the planned, flown, and baseline trajectories.

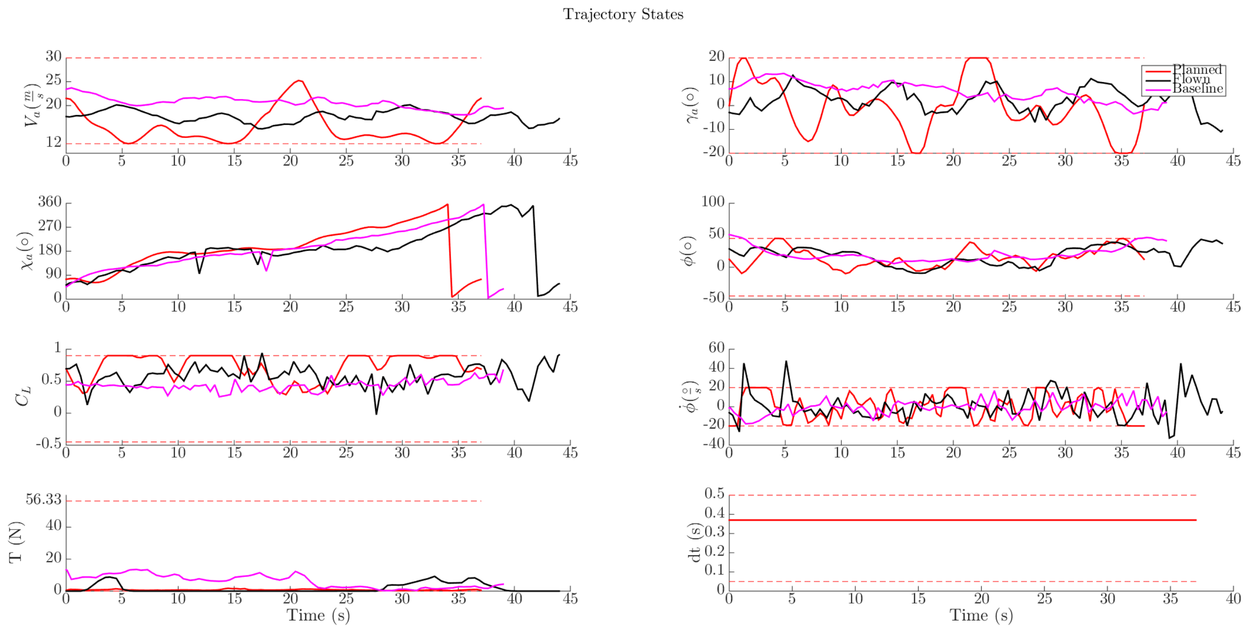


Figure 5.25: Comparison of the planned, flown, and baseline states.

5.4 Summary of Results

Although three loiter trajectories were described in detail in the previous section, several more trajectories were planned and flown during the two week experiment in Lubbock, TX in June

2015. The full EA-DDDAS was implemented over four days of flights, each day containing multiple flights and experiments. Although the majority of the results were not more efficient than their baseline, the suggestions in this document may improve future experiment results.

Date	Flight No.	Time of Day (CST)	Duration (sec)	Cost	Better than baseline?
6/22/2015	1.B	12:17:10 PM	55	6260	–
–	1.1	12:32:00 PM	35	5519	Yes
6/23/2015	1.B	10:34:05 AM	45	3207	–
–	1.1	10:41:06 AM	47	13382	No
–	1.2	10:56:17 AM	39	6258	No
–	2.1	11:32:00 AM	31	8237	No
–	2.B	11:39:31 AM	39	2723	–
–	2.2	11:50:09 AM	44	507	Yes
–	3.1	1:07:23 PM	50	3930	No
–	3.2	1:14:11 PM	49	5268	No
–	3.B	1:15:49	38	3068	–
–	3.3	1:30:58 PM	47	5398	No
6/24/2015	1.1	11:02:26 AM	38	6421	No
–	1.B	11:11:03 AM	53	1723	–
–	1.2	11:15:18 AM	41	6036	No
–	1.3	11:47:39 AM	41	3956	No
6/25/2015	Rest Day	–	–		
6/26/2015	1.1	11:17:07 AM	100	–	
–	2.1	2:03:57 PM	140	–	–
–	3.1	4:26:31 PM	130	–	–
–	3.2	4:42:19 PM	105	–	–
–	3.3	5:07:35 PM	98	–	–
6/27/2015	Motor Failure	–	–	–	

Table 5.1: A summary of flight results from the Lubbock, TX 2015 deployment.

Chapter 6

Conclusion

6.1 Summary of Findings

In this work, a trajectory optimization software for sUAS was presented that enables long endurance real-world flights for the purposes of performing persistent sampling in dynamic environments. In Chapter 1 the motivation for developing a trajectory optimization software in the context of an EA-DDDAS was presented. In addition, the concept of dynamic soaring was introduced.

In Chapter 2, the optimization problem is presented and the aircraft equations of motion are defined. The states, control inputs, and decision vector are defined for a 3D point mass model sUAS. A nonlinear program is formulated as a nonlinear optimization problem that can be solved explicitly by a software called SNOPT. The concept of guidance and loitering trajectories as base classes are introduced and the notion of a longer-horizon lattice planner operating above the TOL is described.

In Chapter 3, the implementation challenges and mitigations are discussed in detail. The concept of a wind field database is introduced and the implementation of a MongoDB server operating within the TOL is described. The trilinear interpolation technique using shape functions is outlined. The trajectory optimizer C++ code is outlined and presented by its major sections, including problem setup, execution, and stitching. The optimization parameters are described and a new method for calculating the large, sparse, Jacobian matrix are introduced. Finally the communication with the other EA-DDDAS components via the Mission Selection Logic (MSL) is discussed in detail.

In Chapter 4, simulations of the TOL are performed within the context of a supercell tornadic storm. The simulation involves planning a penetrative trajectory into the rear flank downdraft of the storm and performing a loitering mission within an area of potential meteorological interest. A sample trajectory is planned and examined with respect to the wind field used by the optimizer. The stitching functionality of the TOL is also demonstrated by autonomously planning a guidance approach, loiter, and departure trajectory in real-time.

In Chapter 5, the trajectory optimization layer performance was assessed by conducting real flight tests in Lubbock, TX. Flown loitering trajectories are compared to baseline trajectories flown at approximately the same altitude and radius. Two of the three optimized trajectories examined were shown to be less efficient than the baseline counterpart. The third loitering trajectory was found to be more efficient by a factor of nearly five.

The lack of an appropriate guidance control layer and significant wind shear are believed to be the causes of low performance by the flown trajectories. Due to the simplistic nature of a waypoint following controller and the aggressiveness of the optimized paths, significant control effort was required, in the form of thrust, to maintain a low cross-track error.

Despite the lack of any significant shear, the trajectory optimization layer generated paths that were able to take advantage of small regions of upward air motion to reduce the impact on thrust usage. Turning maneuvers that utilized the oncoming wind were also common in the generated trajectories and in one instance led to a more efficient path.

The air-relative energy and energy rates were examined for each trajectory case. With each case, the dynamic soaring term played a small role in the overall energy transfer of the trajectory. The static soaring term was significant in two of the three flown trajectories and helped offset the drag force. Thrust for the flown trajectories was calculated using measured on-board power and its relationship to airspeed. However, this calculation of thrust is suspect due to the lack of any inefficiency modeling between the flight battery and the actual thrust produced by the propeller.

The baseline trajectories may have outperformed the flown trajectories due to the control logic governing their paths. The baseline loiters were controlled by maintaining a coordinated turn bank angle and airspeed for a given loiter radius. This considerably simpler controller may have caused less abrupt changes in lift and thrust use, leading to more efficient trajectories.

In summary, the limitations of the prior path planning algorithm were identified and addressed. The trajectory optimization layer and its software improvements have been proven to be capable of running in an online fashion to generate stitched, optimal trajectories for use in a flight system. The lack of ideal conditions and a suitable trajectory following controller may have contributed to less than ideal performance during the flight tests.

6.2 Future Work

Future work for this avenue of research could involve developing an effective trajectory tracking controller that not only tracks state position, but also airspeed, air-relative body angles, and rates. This may improve path following performance as well as reduce the amount of control effort used on reducing position error. The new controller could also take advantage of the optimizer providing temporal information about the aircraft's pose and roll rate.

An adjustment to the focus of the cost function could be made to instead attempt to optimize power by minimizing the product of airspeed and thrust. This would better correlate with the reduced battery-power usage on-board the aircraft, especially once a battery-thrust relationship is modeled. This cost would capture the work done by the thrust upon the air by the propeller.

Future experiments could also include flying the EA-DDDAS framework within a severe storm environment to realize any dynamic soaring or static soaring benefits. It may become easier to discern the benefits in an environment where the baseline trajectory is highly inefficient due to high winds. This further experimentation could be performed in tandem with developing a more accurate energy model relating power usage of the flight battery with the thrust produced by the sUAS propeller. This would enable the optimized trajectories to be better compared with the actual flight performance of the aircraft.

Bibliography

[1] JJ Bird, JW Langelaan, and C Montella. Closing the Loop in Dynamic Soaring. (January):1–19, 2014.

[2] Department of Defense. FY20132038 Unmanned Systems Integrated Roadmap. Technical Report 11, Office of the Secretary of Defense, 2013.

[3] T Flanzer, G Bower, and I Kroo. Robust Trajectory Optimization for Dynamic Soaring. AIAA Guidance, Navigation, and Control . . . , (August):1–22, 2012.

[4] Martin Franklin. Dynamic Soaring.

[5] Eric W. Frew, Brian Argrow, Adam Houston, Chris Weiss, and Jack Elston. An Energy-Aware Airborne Dynamic Data-Driven Application System for Persistent Sampling and Surveillance. Procedia Computer Science, 18:2008–2017, January 2013.

[6] Xian Zhong Gao, Zhong Xi Hou, Xiong Feng Zhu, Jun Tao Zhang, and Xiao Qian Chen. The shortest path planning for manoeuvres of UAV. Acta Polytechnica Hungarica, 10(1):221–239, 2013.

[7] Matthew S. Gilmore, Jerry M. Straka, and Erik N. Rasmussen. Precipitation and Evolution Sensitivity in Simulated Deep Convective Storms: Comparisons between Liquid-Only and Simple Ice and Liquid Phase Microphysics*, 2004.

[8] Matthew S. Gilmore, Jerry M. Straka, and Erik N. Rasmussen. Precipitation Uncertainty Due to Variations in Precipitation Particle Parameters within a Simple Microphysics Scheme, 2004.

[9] Nicholas R. J. LAWRANCE and Salah SUKKARIEH. Autonomous Exploration of a Wind Field with a Gliding Aircraft. Journal of guidance, control, and dynamics, 34(3):719–733.

[10] Nicholas R J Lawrance and Salah Sukkarieh. A guidance and control strategy for dynamic soaring with a gliding UAV. Proceedings - IEEE International Conference on Robotics and Automation, pages 3632–3637, 2009.

[11] Lorenz Meier. MAVLink Micro Air Vehicle Communication Protocol, 2009.

[12] Paul M. Markowski, Jerry M. Straka, and Erik N. Rasmussen. Direct Surface Thermodynamic Observations within the Rear-Flank Downdrafts of Nontornadic and Tornadic Supercells, 2002.

[13] Paul M. Markowski, Jerry M. Straka, and Erik N. Rasmussen. Tornadogenesis Resulting from the Transport of Circulation by a Downdraft: Idealized Numerical Simulations, 2003.

[14] Joseph L Nguyen, Nicholas R J Lawrance, and Salah Sukkarieh. Nonmyopic Planning for Long-Term Information Gathering with an Aerial Glider. 2014.

[15] Technology Quarterly, I T Is, and Helico Aerospace Industries. The robot overhead, 2015.

[16] Wenceslao Shaw-cortez. Energy-Aware Path Planning for UAS Persistent Sampling and Surveillance. 2013.

[17] William Silva and Eric Frew. Path Planning for Persistent Environmental Monitoring Using an Energy-Aware Airborne Dynamic Data-Driven Application System. pages 2–4.

[18] William Silva, Eric W Frew, and Wenceslao Shaw-cortez. Implementing Path Planning and Guidance Layers for Dynamic Soaring and Persistence Missions. ICUAS, 2015.

[19] Jerry M. Straka and John R. Anderson. Numerical Simulations of Microburst-producing Storms: Some Results from Storms Observed during COHMEX, 1993.

[20] Tom Simonite. Air traffic control. MIT Technology Review, page 3, 2014.

[21] Yiyuan J. Zhao. Optimal patterns of glider dynamic soaring. Optimal Control Applications and Methods, 25(2):67–89, March 2004.