Trajectory Optimization of Solar-Powered High-Altitude Long Endurance Aircraft

Jack Marriott, Birce Tezel, Zhang Liu and Nicolas E. Stier-Moses
Facebook Connectivity
Facebook, Inc.
Los Angeles, CA, USA
marriott@fb.com

***Abstract* — Solar-powered aircraft can fly perpetually if they are able to fly a 24-hour mission in which they collect more energy than they expend. The energy collected is determined by the orientation and length of time that the aircraft's solar panels are exposed to the sun. The energy spent is determined by flight maneuvers such as changes in airspeed, altitude and heading. Therefore, a perpetual energy balance can be maintained when the aircraft flies along a trajectory that maximizes solar exposure and minimizes flight maneuvers. This paper presents a faster than real-time dynamic programming strategy that computes the optimal trajectory that contains a loitering aircraft inside a cylindrical volume placed about a reference waypoint.**

Keywords—solar powered aircraft, trajectory planning, perpetual endurance, geostationary station keeping, greedy path planning with buffering, multi-path Dijkstra.

I. INTRODUCTION

Starting in 1994, NASA's ERAST program "was a multiyear effort to develop the aeronautical and sensor technologies for a new family of remotely piloted unmanned aircraft intended for upper atmospheric science missions. Designed to cruise at slow speeds for long durations at altitudes of 60,000 to 100,000 ft, such aircraft could be used to collect, identify, and monitor environmental data to assess global climate change and assist in weather monitoring and forecasting. They also could serve as airborne telecommunications platforms, performing functions similar to communications satellites at a fraction of the cost." [7]

AeroVironment's Pathfinder was a notable solar-powered high-altitude long endurance (SPHALE) unmanned air vehicle (UAV) that predated ERAST. Pathfinder demonstrated the promise of this technology, but it was unable to overcome the propulsion system's limitations at that time [2]. Under its auspices, ERAST demonstrated new SPHALE airframes. The lesson learned was that the combined energy storage and propulsion system are the Achilles heel in the SPHALE UAV technology. More recently, Airbus' Zephyr S High Altitude Platform Station (HAPS) solar-powered aircraft just flew for nearly 26 days straight. The Zephyr T is under development as of this writing, [1].

The HAPS/SPHALE concept is namely a low weight and high aspect ratio airframe with solar panels on the upper surfaces. The panels collect solar energy during the daytime. This energy is stored in batteries, which power the avionics and propulsion system (electric motors with propellers). The lack of solar energy between sunset and sunrise means that daytime collection net of consumption must provide enough charge to make it through the night. Aircraft that can close a daily energy cycle without a deficit are known as perpetual endurance vehicles.

The magnitude of solar energy available varies with time of year, latitude, and solar elevation angle (which is a function of time of day). At a latitude of 35 degrees during the winter solstice, there are only about 10 hours of sunlight, of which the periods near sunrise and sunset are too feeble to provide any useful irradiance.

In practical terms, the aircraft is required to autonomously program a trajectory that maximizes energy storage at the end of the day. This in turn means finding the orientation that maximizes exposure to the sun, while consuming the least power in doing so. For aircraft used as a communications platform, this imposes the constraint of remaining within a geostationary containment volume, as shown in Fig. 1. In summary, a practical optimal trajectory must have the following characteristics:

- account for radial station keeping constraints;
- account for vertical station keeping constraints;
- store excess energy as gravitational energy; and,
- be computable in-flight.

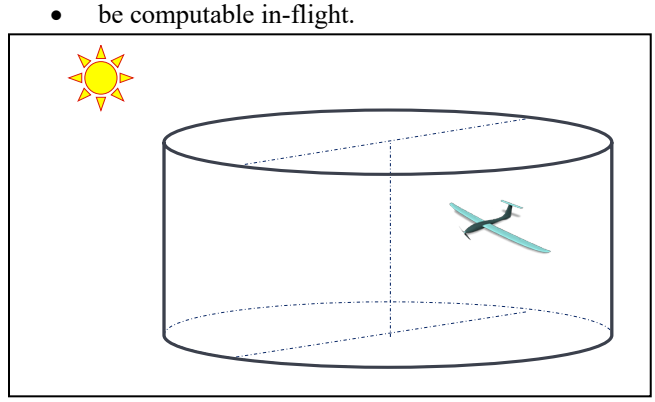

Figure 1. Containment area.

II. RECENT DEVELOPMENTS

Autonomous navigation algorithms have mostly focused on ground vehicles, e.g. [12]. These compute a trajectory between desired discrete endpoints, with the trajectory itself being the objective. There is no consideration of the energetic cost of the displacement. The literature lacks a dedicated set of solutions for flight vehicles.

In their work on energy optimal flight path planning, Klesh & Kabamba formulate the problem of perpetual loitering, propose a solution for the case of level flight [11], and derive the necessary conditions for energy optimal flight [8].

Martin et al. pose the trajectory optimization problem as a nonlinear model predictive control problem (MPC), [4]. Solving for a 24-hour flight period takes about 10 hours, when using 10 second time steps. Their solution identifies several repeatable maneuvers that depend on solar elevation and time of day. They suggest that these maneuvers could be parameterized into a state machine.

Bolandhemmat et al. compute a solution to the optimal trajectory problem using both interior point (gradient based) and nonlinear simplex optimization, [3]. These solutions "provide no formal guarantees on convergence to an optimal (or even feasible) solution. Additionally, the computational burden and associated power draw makes it impractical to perform the trajectory optimization using an onboard computer." The authors ameliorate this shortcoming by using their computed optimal trajectory to train an adaptive neuro-fuzzy inference system which can then plan the trajectory in real-time.

Recent work has focused on optimal control methods to compute trajectories. Though these methods produce good results, execution time is too slow to be useful in-flight. A preferred solution method would be one that can both solve and plan the trajectory in real-time.

III. Original Contributions

This paper makes three original contributions. First, it solves the optimal trajectory problem as a search problem using a dynamic programming (DP) approach. Second, it develops a solution method that runs much faster than published solutions. These developments are unique contributions that together can enable autonomous perpetual flight. Finally, we enumerate the repeatable trajectory behaviors observed in our results and assemble them into a state machine.

IV. Trajectory Optimization

The trajectory optimization problem for the solar powered aircraft in station keeping mode is to maximize the total energy stored in the aircraft, subject to constraints that capture aircraft dynamics, station keeping, and input command limits. Mathematically, we can formulate this as a non-convex constrained optimization problem:

$$\begin{aligned} max \quad & E_b + \kappa E_p \\ \text{s.t.} \quad & \dot{x}(t) = f(x(t), u(t), w(t)) \\ & g(x(t), u(t)) \leq 0, \end{aligned} \tag{1}$$

where $x(t)$, $u(t)$, $w(t)$ are time-domain signals for the states of the system, the inputs to the system, and the external disturbances, respectively. The total system energy is stored in the batteries, E_b, and as potential E_p energy. Battery energy is typically referred to as the state-of-charge. The parameter κ is a weighting constant that amplifies the potential energy during the daytime.

The state vector x includes the aircraft's position, velocity, attitude, and battery state-of-charge. The input vector u contains the optimization variables, including the yaw rate command, R_{cmd}, the vertical velocity (altitude rate) command, $V_{v,cmd}$, and the airspeed command. The disturbance vector w can include wind, gust, turbulence, sensor noise, and modeling uncertainties. The inequality constraint $g(\cdot) \leq 0$ is used to capture the input command upper and lower limits and the station-keeping requirement. Section V provides details on the difference equations $\dot{x}(t) = f(\cdot)$ that capture the aircraft kinematics, dynamics, and solar/battery/propulsion energy dynamics. Section VI.*E* presents the discretized version of (1) that our solution procedure uses.

V. System Model

Aircraft are designed to display specified dynamic responses to a given set of control inputs. A closed loop autopilot responds to these inputs, maintains the airframe stable, and guarantees a response akin to that of a first order differential equation. It is thus possible to design a kinematic fixed wing aircraft simulation that captures this high-level behavior with a minimum number of states and without the requisite small step size, ΔT. Henceforth, the use of a superscript i indicates the time-period during which the referenced variable is updated.

A. Input Commands

An airframe is controlled through a combination of longitudinal, lateral/directional, and vertical input commands. Collectively, our inputs are equivalent airspeed, V_e; yaw rate, R_{cmd}; and vertical velocity, $V_{v,cmd}$, respectively. We decided to consider a constant V_e. The motivation for this decision was twofold: first, this airframe operates in a narrow range of airspeeds (i.e., +/- 1 m/s); second, there are analytical solutions to different speed objectives. E.g., best climb speed, endurance speed, and best descent rate are all known and can be computed online.

B. Autopilot

The updated altitude, h, and heading, ψ, are respectively

$$h^i = h^{i-1} + \Delta h^i \Delta T, \text{ and} \tag{2}$$

$$\psi^i = \psi^{i-1} + \Delta\psi^i \Delta T. \tag{3}$$

The change in altitude per time step, Δh, and change in heading per time step, $\Delta\psi$, are given by

$$\Delta h^i = k_h V_{v,cmd}^i, \text{ and} \tag{4}$$

$$\Delta\psi^i = k_\psi R_{cmd}^i. \tag{5}$$

The terms k_h and k_ψ are non-dimensional factors that account for the dynamics of the vertical and directional response, respectively. For example, in the presence of a step command, a stable first order dynamic system representative of the altitude response rises 33.6% in a 10 second time step.

Though V_e is constant in this implementation, the true airspeed, V_a, does vary with changing altitude. From [13], we know that

$$V_a^i = V_e^i \sqrt{\rho_0/\rho^i}. \tag{6}$$

The air density, ρ, varies with altitude as described in [17], but is constant at sea level, ρ_0.

C. Kinematics

The inertial flight path and bank angles, described in [15], are respectively given by

$$\sin\gamma^i = \Delta h^i / V_a^i, \text{ and} \tag{7}$$

$$\tan\phi^i = \Delta\psi^i V_a^i / g. \tag{8}$$

When $\phi = 0$, the North and East positions on the local horizontal plane (the computed flat Earth positions), p_n and p_e, respectively, are computed as follows

$$\begin{bmatrix} p_e^i \\ p_n^i \end{bmatrix} = \begin{bmatrix} p_e^{i-1} \\ p_n^{i-1} \end{bmatrix} + V_a^i \Delta T \begin{bmatrix} \sin\psi^i \\ \cos\psi^i \end{bmatrix}. \tag{9}$$

When $\phi \neq 0$, the trajectory is curved with radius ρ_c and instantaneous center of rotation at position P_c. North and East positions are computed as follows

$$\rho_c^i = (V_a^i)^2 / \tan\phi^i, \tag{10a}$$

$$P_c^i = \begin{bmatrix} p_e^{i-1} \\ p_n^{i-1} \end{bmatrix} + \rho_c^i \begin{bmatrix} \cos\psi^i \\ -\sin\psi^i \end{bmatrix}, \tag{10b}$$

$$\begin{bmatrix} p_e^i \\ p_n^i \end{bmatrix} = P_c^i - \rho_c^i \begin{bmatrix} \cos\Delta\psi^i & \sin\Delta\psi^i \\ -\sin\Delta\psi^i & \cos\Delta\psi^i \end{bmatrix} \begin{bmatrix} \cos\psi^i \\ -\sin\psi^i \end{bmatrix}. \tag{10c}$$

D. Navigation

Navigation consists of estimating geodetic latitude, φ, and longitude, λ, from the computed flat Earth positions on a World Geodetic System 1984 reference. These are used to estimate the solar angles. Navigation in this implementation was performed using Python's module Pyproj [5].

E. Aerodynamics

The sum of perpendicular forces [13] about the center of gravity for an aircraft in translational flight with center line thrust is given by

$$L/\cos\phi - W\cos\gamma = m V_a^2/(r_e + h), \tag{11}$$

where W is the aircraft weight, m is its mass, and r_e is Earth's local radius. Lift can be solved directly from (11), and nondimensionalized as follows

$$c_L = L/(\bar{q}S),\ c_D = D/(\bar{q}S), \tag{12a,b}$$

$$\bar{q} = \rho V_a^2/2, \tag{13}$$

where $\bar{q}$ is the dynamic pressure, and S is the reference wing area.

With c_L known, the angle of attack is computed in a reverse look-up from the $c_L(\alpha, h)$ curve, Fig. 2. With α known, c_D is computed in a forward look-up from the $c_D(\alpha, h)$ curve, Fig. 3, and the drag force from (12b). Thrust can be computed directly from the sum of parallel forces [13] about the center of gravity for an aircraft in translational flight with center line thrust, as given by

$$T - D - W\sin\gamma = m\,(V_a^i - V_a^{i-1})/\Delta T. \tag{14}$$

Knowledge of α also allows computing the pitch angle, θ, as in

$$\theta = \alpha + \gamma. \tag{15}$$

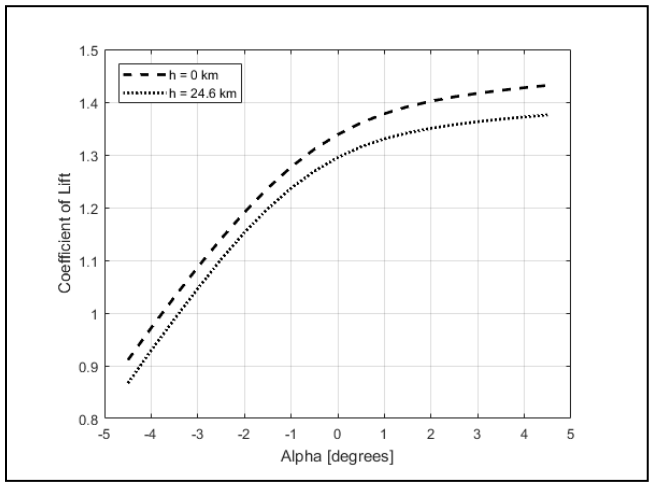

Figure 2. Lift curve.

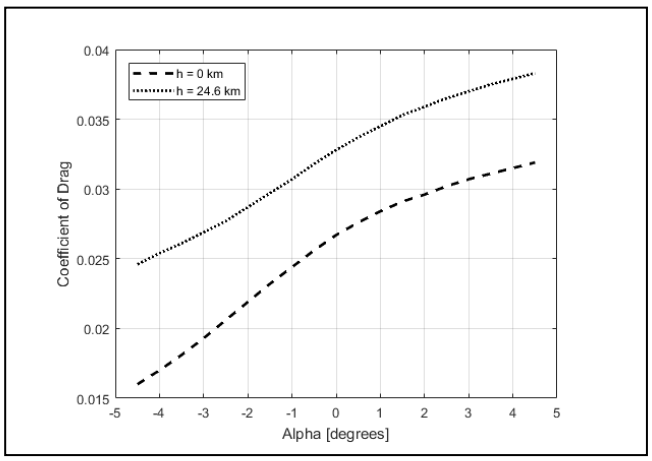

Figure 3. Drag curve.

F. Solar Model

The solar model consists of three parts: irradiance, solar vector, and solar collection. The irradiance is the average power per unit area received from the sun on a specific day, altitude, and solar elevation angle. It is described in [14] as

$$I = I_0\left(1 + 0.034\cos\frac{2\pi n_d}{365}\right) f(h, \varepsilon_s), \tag{16}$$

where n_d is the Julian day index, and I_0 is the solar constant and $f(h, \varepsilon_s)$ is an atmospheric absorption factor described in [9]. The solar angles, ε_s and ζ_s, [10], represent solar elevation and azimuth, respectively. Solar position computations were performed using Python's module Pysolar [6].

The aircraft is fitted with k solar panels, each with area $A_{s,k}$, and position $[x_k \quad y_k \quad z_k]$ in the vehicle's forward-right wing-down coordinate system. The total solar panel area projected perpendicularly at the sun, A_p, is given by

$$A_p = \sum_k A_{s,k} \begin{bmatrix} -\cos\zeta_s\cos\varepsilon_s \\ -\sin\zeta_s\cos\varepsilon_s \\ \sin\varepsilon_s \end{bmatrix}^T R_b^s \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix}, \tag{17a}$$

$$R_b^s = R_\psi R_\theta R_\phi, \tag{17b}$$

$$R_\psi = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{17c}$$

$$R_\theta = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \tag{17d}$$

$$R_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}. \tag{17e}$$

G. Power and Propulsion

Power is the rate of energy, and solar energy is the only source harnessed by the aircraft. Input power, P_{in}, power output from motors and accessories, P_{out}, and net power, P_{net}, are given by

$$P_{in} = \eta_s I A_p, \tag{18a}$$

$$P_{out} = P_{acc} + \frac{1}{2} T V_a \left[\left(\frac{T}{N A_d \bar{q}} + 1 \right)^{1/2} + 1 \right], \tag{18b}$$

$$P_{net} = P_{in} - P_{out}, \tag{18c}$$

where η_s is the solar panel efficiency, P_{acc} is the accessory power (e.g., avionics), N is the number of rotors, and A_d is the propeller disk area. The second term of the right-hand side of (18b) is the efficiency-adjusted propulsive power required to deliver the aerodynamic thrust.

H. Energy

Electrical energy is stored in and drawn from the rechargeable batteries, where charging and discharging occurs at different efficiencies. The energy stored in all batteries, E_b, is

$$E_b^i = E_b^{i-1} + \eta_b P_{net}^i \Delta T, \tag{19a}$$

$$\eta_b = \begin{cases} \eta_{in} & if\ P_{net}^i \geq 0 \\ \eta_{out} & if\ P_{net}^i < 0 \end{cases}, \tag{19b}$$

where η_b is either the charging, η_{in}, or discharging, η_{out}, efficiency, depending on mode.

The aircraft can also store gravitational potential energy in the form of higher altitude. There is a tradeoff with climbing, in that it requires higher thrust, which means that higher power is drawn from the batteries. Therefore, an increase in gravitational energy is only an increase in total energy if the aircraft can climb when $P_{net} > 0$. For purposes of energy optimization, we only account for the gravitational energy above the altitude floor h_{lo}:

$$E_p^i = (h^i - h_{lo}) mg. \tag{20}$$

I. Winds

As was done in [4], the effect of winds was excluded to reduce simulation complexity without loss of generality. Significant winds aloft would invalidate (7) – (10). Winds could be accounted for with the following modifications: track course instead of heading, compute heading from wind triangle, use air-mass-referenced flight path angle. A derivation of these changes is presented in [3]. The resulting equations are similar to those stated herein, with little impact in computation time.

VI. Greedy Path Planning With Buffering

This section outlines an efficient greedy algorithm for finding a trajectory throughout the day for the solar-powered aircraft to have enough exposure to the sun such that the stored energy is sufficient to keep the aircraft in the air.

A. Assumptions, Initial Conditions and Restrictions

We make some simplifying assumptions on the trajectory planning:

- time is discretized into ΔT intervals;
- equivalent airspeed is constant;
- winds are not considered (discussed in Section V.*I*);
- the controls are limited to changes in heading angle and altitude.

Initial conditions that should be specified:

- stored battery energy;
- latitude, longitude and altitude;
- east and north offsets;
- date and time of the flight;
- maximum energy that the battery can store.

The aircraft must remain within:

- three kilometers radius from the center,
- a lower and upper altitude limits.

Implementation details:

- Where memory states are used, e.g. h^{i-1}, these are initialized per the initial conditions.
- The upper limit of Δh^i is the lower of 3 degrees γ and 0.8 m/s. The lower limit is symmetric, but further restricted to be higher than the value which results in negative thrust.
- $\Delta\psi$ is limited to not exceed the bank angle limit
- The bank angle limit is scheduled with altitude, ranging from 5 degrees at sea level, up to 10 degrees at 23 km and higher.
- Battery charging is disabled when the battery is full; discharging is disabled when the battery is empty.

B. States and Commands

We now outline a greedy algorithm that returns a set of commands to take at the beginning of each 10-second period such that results in sufficient gravitational and battery energy with minimal violations of the positional restrictions.

Recall that $x(t)$ and $u(t)$, are the continuous-time domain state and input vectors of the system defined in (1). Let S^i represent the discrete state vector of the aircraft at time i; let C^i represent the discrete input vector, also at time i. S^i contains all the information necessary to evaluate the change in total energy and the positional feasibility associated with each command. For purposes of trajectory planning, S^i consists of: east and north offsets, altitude, heading angle, stored battery energy and true airspeed:

$$S^i = \left(p_e^i, p_n^i, h^i, \psi^i, E_b^i, V_a^i\right). \tag{21}$$

C^i consists of the commands: change in heading angle and change in altitude:

$$C^i = (\Delta\psi^i, \Delta h^i). \tag{22}$$

Note that, given the state S^i and the commands C^i, we do not need any information on previous states to continue the trajectory planning.

C. Tree Representation and Complexity

We model the trajectory planning problem as a tree-structure where each node represents a state of the aircraft and the depth of the tree represents time. The root node corresponds to the initial state. There exists an edge between two states S^i and S^{i+1} only if there is at least one command vector C^i that would transition S^i into S^{i+1}. Moreover, each edge is associated with a value that captures the change in total energy caused by this transition, penalized by the violations in state S^{i+1}, if any. Consequently, the trajectory planning problem is equivalent to finding the longest path from the root node to all the leaf nodes.

Even with all the simplifying assumptions outlined in the previous section, this is a challenging problem since the number of nodes increases exponentially as the depth of the tree increases. Let $\bar{S}^i$ be the set of all potential states we can reach and $\bar{C}^i$ be the set of all possible commands that can be taken at time i. Letting $c \coloneqq |\bar{C}^i|$, there are roughly c^{i+1} possible states $s \in \bar{S}^{i+1}$ to evaluate. Suppose that we are planning the trajectory for N time periods, then the total number of nodes and edges in the tree representation would be in the order of $\mathcal{O}(c^N)$.

Consequently, even solving the simplification of the trajectory planning problem is challenging due to the exponential number of nodes and edges. Note that the longest path problem can be solved in linear time on the number of nodes via topological sorting [16, 18]; however, the main challenge stems from the size of the tree.

D. Greedy algorithm with buffering

Our main challenge is the exponential number of states to be evaluated. To overcome this problem, we consider a greedy algorithm, where we only store the top k states at each time-period, starting from the root node, Fig. 4. We refer to this storage process as *buffering*. Note that if $k = 1$, then the algorithm is equivalent to a simple greedy algorithm where, at each node, we select the edge with the largest weight. The algorithm terminates when a leaf node is reached.

Buffering significantly reduces the complexity of this algorithm: it stores k nodes at each time-period i. From these nodes, we compute the weights of edges to $k \times c$ states. Among these states, we pick the top k with largest total weight from the root node. Letting N be the number of time periods, the overall complexity of this algorithm is $\mathcal{O}(Nkc\log(kc))$.

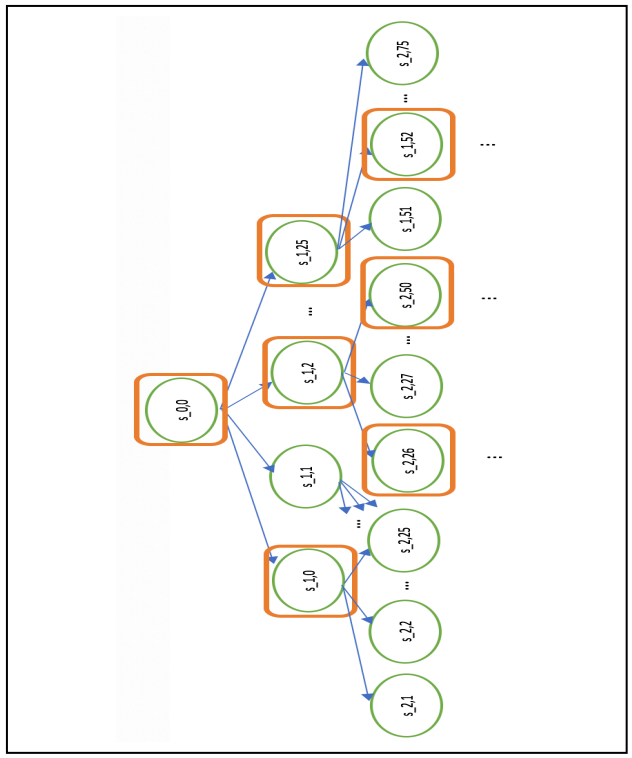

Figure 4. Greedy algorithm with $k = 3$.

E. Edge Weights: Adjusted change in Energy

We compute a trajectory time history by selecting the best path along a succession of energy states. Given multiple alternative states, or nodes, the transition between nodes is evaluated by measuring the change in adjusted energy, ΔE, as given by,

$$\Delta E^i = E^i - E^{i-1}. \tag{23}$$

The adjusted energy, E, is a modified version of the objective function (1). In particular, it is defined as the sum of battery energy and weighted gravitational energy, penalized by the violation of the containment region, q:

$$E^i = \left(E_b^i + \kappa E_p^i\right)/q^i. \tag{24}$$

We found that the best strategy was to set the potential energy weight, κ, to 0 at nighttime (as indicated by $\varepsilon_s < 0$). In other words, the nighttime strategy is to optimize for the battery only.

The infeasibility factor, q, represents a penalty for being outside the containment region, either vertically or horizontally. It is given by

$$q^i = \begin{cases} 1 & q_v^i = q_h^i = 0 \\ q_0 + q_v^i + q_h^i & otherwise, \end{cases} \tag{25}$$

where q_0 is a constant infeasibility factor that enables faster adaptation; q_v is the vertical penalty, and q_h is the horizontal penalty. These penalties are defined as

$$q_v^i = \begin{cases} \left|h^i - h_{lo}\right|/(h_{hi} - h_{lo}) & h^i < h_{lo} \\ \left|h^i - h_{hi}\right|/(h_{hi} - h_{lo}) & h^i > h_{hi} \\ 0 & otherwise, \end{cases} \tag{26}$$

$$q_h^i = \begin{cases} |d^i - r_c + r_b|/r_c & d^i > (r_c - r_b) \\ 0 & otherwise. \end{cases} \quad (27)$$

In the above, h_{hi} is the ceiling altitude; h_{lo} is the floor altitude; r_c is the containment volume radius; r_b is the containment radius buffer; and d is the horizontal distance from vehicle to center of containment area,

$$d^2 = p_e^2 + p_n^2. \quad (28)$$

F. Implementation

We compute the trajectory by solving forward from the current state over a thirty-minute horizon in ten-second time steps. At the end of each horizon, we pick the best solution among all candidate end-states in the buffer. Algorithm 1 outlines the main steps of this solution.

Let $f^i(S^i, C^i)$ represent the set of equations that transitions state S^i to S^{i+1} at time i, if the commands C^i are selected, $g^*(S^i)$ be the largest adjusted energy (24) that can be achieved for state S^i at time $i+1$ and C^{*i} be the command that achieves that energy. Also let $g^i(S^i, C^i)$ be the total adjusted energy collected at state S^i at time i via command C^i. Then,

$$g^*(S^{i+1}) = \max_{S^i \in \bar{S}^i} \{g^*(S^i) + \Delta E^i\}. \quad (29)$$

After computing values $g^*(S^i)$ for $i = 1, \ldots, N$, we find the final state that collected the largest adjusted energy. Using this information, the algorithm backtracks the set of commands that had to be taken to achieve this final state.

G. Computation Time

Table I compares this work to the gradient and MPC methods using ten-second time steps to solve over a thirty-minute horizon. Our resulting execution time was ninety seconds on average. Our development environment was entirely Python based, [5, 6].

Method	Complexity	Compute Time [min] 30 min Horizon
Gradient, [3]	O(nTN)	37.50
MPC, [4]	O(mN)	12.03
Greedy with Buffering, this paper	O(Nkclog(kc))	1.50

TABLE I: Computational complexity and single-CPU time of solving a 30-minute-horizon trajectory optimization.

VII. Trajectory Results

This section presents a case study in which we consider a 24-hour flight in Dehra Dun, India on December 22, 2017 (Winter solstice). We allow five values for the change in heading angle and five values for the change in altitude. This leads to twenty-five possible command pairs ($c = 25$). Time is discretized into ten-second intervals ($\Delta T = 10$ seconds). In the following results, the buffer size is chosen as ten ($k = 10$).

Note that all the values presented in this section are for a fictitious aircraft.

```
Initialization
t # First time period
k # Buffer size
state_0 # Initial state
N # Number of iterations
state_dictionary = []

Algorithm
for i in 1 to N:
 temp = []
 # Explore all states
 for S^{i-1}  in state_dictionary[i - 1]:
  for C^{i-1} in allowed_commands:
   S^i = f^{i-1}(S^{i-1}, C^{i-1})
   ΔE^i = g(S^i, C^i) − g*(S^{i-1})
   Record S^i, S^{i-1}, C^{i-1} and ΔE^i at temp

 top_k_states =
    state_with_largest_energy(temp, k)
 state_dictionary[i] = temp

# Extract the sequence of states and commands
# at time N.
best_final_state =
   state_with_largest_energy_at_N
best_policy = [best_final_state]
next_state = best_final_state
for i in N-1 to 1:
 current_state = get_previous_state(
   state_dictionary[i + 1][next_state])
 add current_state to best_policy
 next_state = current_state
```

Algorithm 1. Greedy algorithm with buffering.

A. State Machine

An analysis of the resulting trajectories in this and other test cases reveals that the charging process can be represented as a state machine, Fig. 5, as had been observed in [4]. States are labeled in the ovals. The bracketed terms indicate the events that trigger transitions between states. The following sections provide additional details on the states of the machine.

B. Dwell

Starting before sunrise and with the battery nearly depleted at 5 kW-hr, the aircraft will dwell in circles at the low limit of its containment altitude, Fig. 6. The aircraft would be expected to settle at the lowest allowable altitude since lower altitude requires lower power at level flight.

C. Charge

Sun rise is declared when $\varepsilon_s > 0$ and the aircraft transitions into charging. The vehicle flies a D-ring pattern, with the straight segment aligned along the solar azimuth, Fig. 7. There is a small change in flight path angle along each direction: a slight climb when flying away from and a slight descent when flying to the sun. This change in altitude corresponds to the aircraft trying to achieve a more perpendicular orientation, which maximizes exposed area and

increases the charge. The aircraft will have a net climb during this stage when there is enough excess energy.

D. Climb

The aircraft will either slowly climb during the charging stage, or explicitly climb at high power once the battery is full, Fig. 8. The climb will generally be at a steep bank angle and continue so long as the irradiance provides for excess energy.

E. Sunset

The sunset state does not encode an explicit type of maneuver, but rather a premature loss in energy, Fig. 9. As shown by the pattern, the battery starts draining at t = 9 hr, which is 1.3 hr prior to nighttime. This loss is due to the low intensity of the irradiance, where the power absorbed is insufficient to maintain both altitude and a full charge.

F. Glide

The aircraft descends at nearly idle power starting at nighttime, and until reaching the low limit containment altitude, Fig. 10. This descent is at the smallest possible bank angle. Upon reaching the floor altitude, the aircraft proceeds to dwell until the next sunrise.

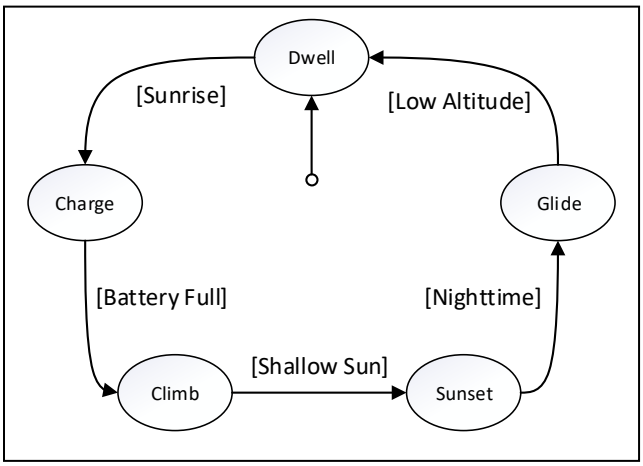

Figure 5. Trajectory state machine.

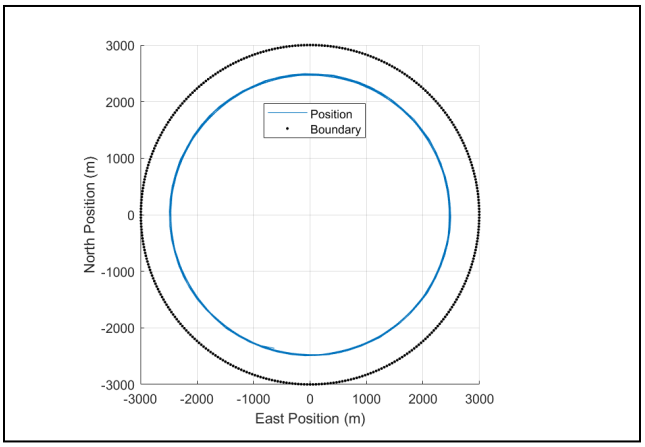

Figure 6. Dwell pattern.

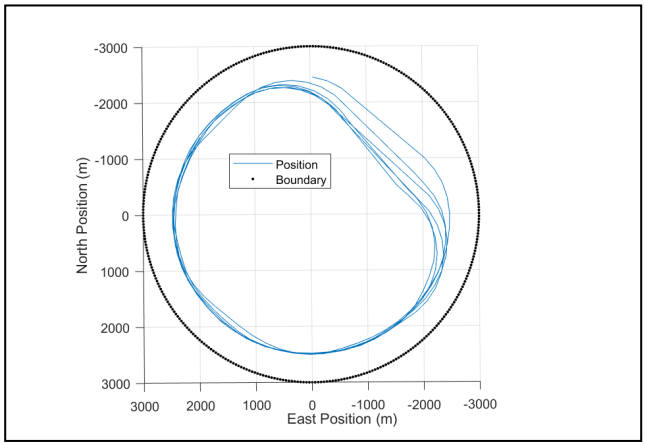

Figure 7. Charge pattern.

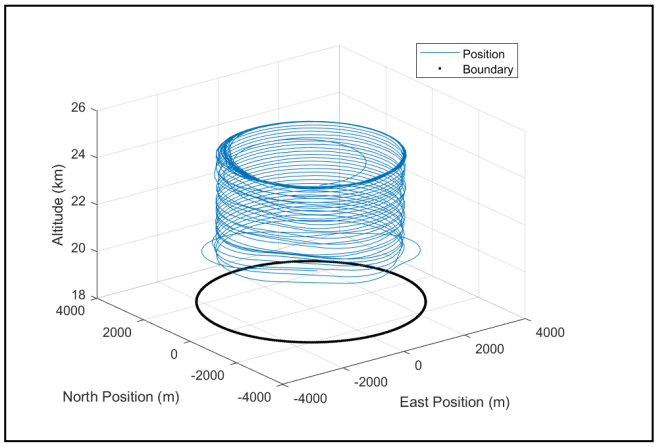

Figure 8. Climb pattern.

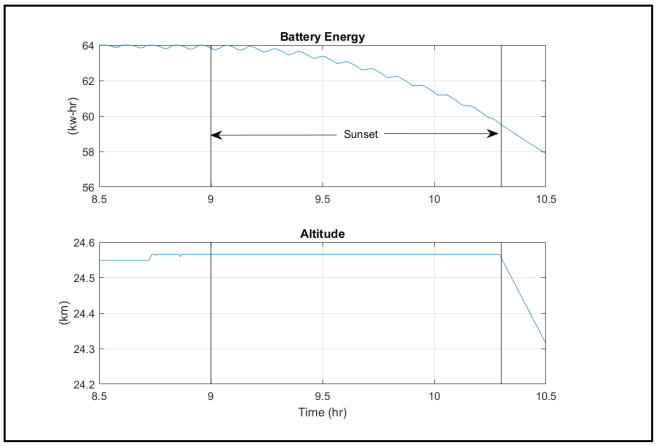

Figure 9. Sunset pattern.

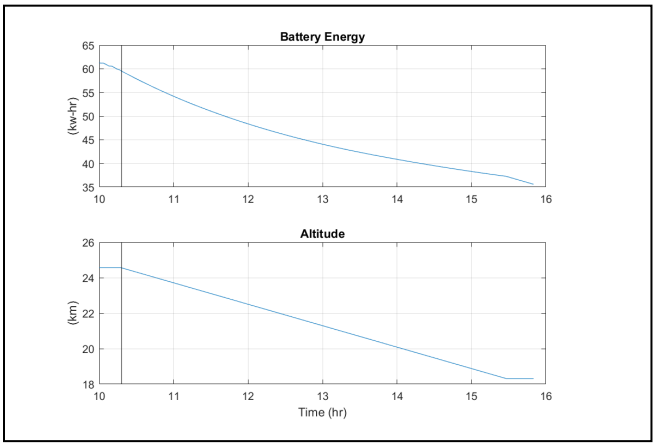

Figure 10. Glide pattern.

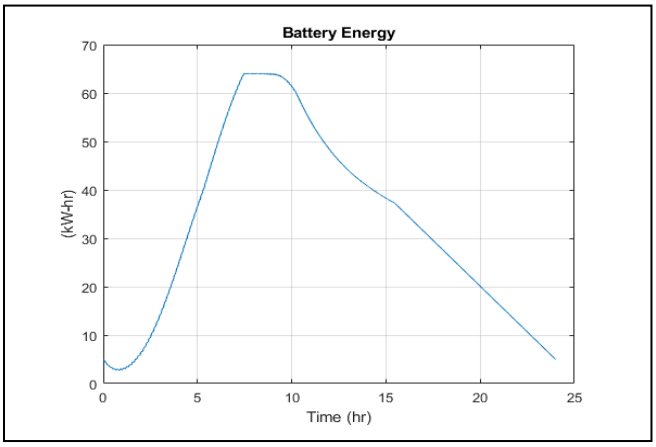

Figure 11. Energy Balance

G. Energy Balance

Our purpose is to design an optimal trajectory for perpetual flight, which is demonstrated by an energy balance greater than or equal to zero. Fig. 11 shows that this is the case over a 24-hour period at this latitude and time of year.

VIII. Conclusion

We have introduced a greedy method with buffering to compute optimal flight trajectories that achieve perpetual flight in a solar powered aircraft. This heuristic solution is a multi-path variation of Dijkstra's shortest path algorithm. Our method offers much faster running times than known alternatives.

Our results have also found agreement with flight maneuvers observed in related work, [3, 4]. These have been formalized into a state machine. This behavior suggests that the optimal trajectory is not a geometric solution but rather a policy.

The field of *Reinforcement Learning* offers algorithms that learn policies that maximize a return. In this case, the policy is the set of guidance actions and the return is the state-of-charge. We believe that the fast computation time of this method is ideally suited to train a reinforcement learner over a wide range of conditions. This is a potential avenue for further research.

A final note on the effect of winds is that winds aloft are typically constant, which would not affect the solution time. Though [3] considers constant winds, variable winds were not considered by any of the surveyed methods. A solution with variable winds would require solving over shorter horizons.

Acknowledgment

The authors wish to thank Julian Mestre of the University of Sydney. His participation was instrumental in getting the project started and providing the oversight to ensure its success.

References

[1] Airbus, viewed Dec. 2019 <www.airbus.com/defence/uav/zephyr.html>

[2] Aerovironment, viewed Oct. 2019 <www.avinc.com/innovative-solutions/hale-uas.html>

[3] H. Bolandhemmat, B. Thomsen and J. Marriott, "Energy-Optimized Trajectory Planning for High Altitude Long Endurance (HALE) Aircraft," 2019 18th European Control Conference (ECC), Naples, Italy, Jun. 2019, pp. 1486-1493.

[4] R.A. Martin, N.S. Gates, A. Ning, and J.D. Hedengren, "Dynamic Optimization of High-Altitude Solar Aircraft Trajectories Under Station-Keeping Constraints," Journal of Guidance, Control, and Dynamics, Vol. 42, No. 3, pp. 538-552, Mar. 2019.

[5] Pyproj 1.9.5.1, released Jan. 2016 <pypi.org/project/pyproj/>

[6] Pysolar 0.7, released Apr. 2015 <pysolar.readthedocs.io/>

[7] Y. Gibbs, *NASA Dryden Fact Sheet: ALTUS II,* viewed Mar. 2015, <www.nasa.gov/centers/armstrong/news/FactSheets/FS-058-DFRC.html >

[8] A.T. Klesh and P.T. Kabamba, "Solar-Powered Aircraft: Energy-Optimal Path Planning and Perpetual Endurance," Journal of Guidance, Control, and Dynamics, Vol. 32, No. 4, Jul. – Aug. 2009.

[9] G.S. Aglietti, S. Redi, A.R. Tatnall, and T. Markvart, "Harnessing High-Altitude Solar Power," IEEE Transactions on Energy Conversion, Vol. 24, No. 2, Jun. 2009.

[10] I. Reda and A. Andreas, "Solar Position Algorithm for Solar Radiation Applications", NREL/TP-560-34302, 2008.

[11] A.T. Klesh and P.T. Kabamba, "Energy-Optimal Path Planning for Solar-Powered Aircraft in Level Flight," AIAA Guidance, Navigation and Control Conference and Exhibit, AIAA 2007-6655.

[12] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics, The MIT Press, 2005.

[13] J.D. Anderson Jr., Introduction to Flight. 5th ed., McGraw Hill, New York, 2005, ch. 4, 6.

[14] T. Markvart, Solar Electricity. 2nd ed., New York: John Wiley & Sons, 2000, ch. 2.

[15] B. Etkin, and L.D. Reid, Dynamics of Flight: Stability and Control. 3rd ed., New York: John Wiley & Sons, 1996, ch. 7, App. A.

[16] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, Network flows, Prentice Hall, 1993.

[17] NASA, "U.S. Standard Atmosphere, 1976" NASA TM-X-74335, 1976.

[18] E.W. Dijkstra, "A note on two problems in connexion with graphs", Numerische Mathematik, Vol. 1, No. 1, pg. 269–271, 1959.