ValBal Altitude Control and Trajectory Planning

Master document

Joan Creus-Costa John Dean

December 14, 2018

Contents

Nomenclature

h	altitude
v	vertical velocity
ℓ	net lift
k_d	linearized drag constant of altitude; $\Delta v = k_d \Delta \ell$
$w_{\dot{\ell}}$	disturbance on lift
w_v	disturbance on the vertical velocity of the balloon (wind)
k_v	altitude controller velocity gain
k_h	altitude controller altitude gain
h_c	commanded altitude
e_{tol}	allowable steady-state deviation around from set altitude
θ	parameters to differentiable trajectory planner, e.g. (h_c, e_{tol}) pairs
$V(\theta)$	value function to be optimized for trajectory planning
$\hat{V}(\theta; \gamma)$	stochastic evaluation of the value function under randomness γ
(u, v)	wind vector in longitude and latitude, respectively.
ϕ	longitude
λ	latitude

Forward

This paper is not written to be like a conventional conference or journal paper with the typical introduction to high altitude balloons or what the potential applications are. Rather, this document jumps right into technical details, as there was a lot to cover and we felt that at this point in time we should focus on just that, rather than the additional info to make it accessible to a wider audience.

In addition, some of the figures on this report are not of sufficient quality for publication. This is because many of them were taken from screen shots posted in SSI's Slack while developing this project. At this stage, going back and generating good looking versions of them would take an unnecessary amount of time while not providing much benefit, as this is still meant to be a draft.

Ultimately, the goal is to condense and generalize the important parts of this document and make a nicer, smaller, journal paper about it. For now, we were focused on getting raw research and experimentation done.

1 Altitude control

1.1 System dynamics

High Altitude Balloons utilize a lifting gas that is lighter than air to produce a buoyant force to counteract the force of gravity on the balloon membrane and the payload. The buoyant force on the balloon minus the force of gravity on the balloon is refereed to as net lift, ℓ. ValBal uses a valve system to vent lifting gas and a ballast system to drop small mass pellets, to change the net lift of the balloon. This net lift produces an acceleration on the system, causing the balloon to accelerate upwards or downwards until the force of drag on the system equals

the net lift, at terminal velocity. Because the balloon accelerates to terminal velocity quickly after a change in net lift, the balloon's vertical velocity, v, can be approximately modeled as always traveling at terminal velocity.

While the force of drag on the balloon is nonlinear with respect to velocity, for the purposed of modeling and control, it can be linearized within a range of reasonable velocities. After doing so, the system dynamics can be simplified to:

$$v(t) = k_d \ell \qquad \dot{h}(t) = v(t)$$

With k_d being the linearized coefficient relating net lift to velocity. The valve and ballast system of can be seen as changes to the derivative of the net lift of the balloon, $\dot{\ell}$. For example, when the valve is open, $\dot{\ell}$ becomes negative as the balloon looses lift. In addition, the atmosphere is full of disturbances, so in the system dynamics we consider two other terms: $w_{\dot{\ell}}$ as the atmospheric disturbance on the lift rate, and w_v, as the atmospheric disturbance of velocity. For example, at night the balloon cools and looses lift, which is modeled as a negative $w_{\dot{\ell}}$, and vertical winds are modeled as a nonzero w_v. With these additions, the system dynamics become

$$\dot{v}(t) = k_d(\dot{\ell}(t) + w_{\dot{\ell}}(t)) \qquad \dot{h}(t) = v(t) + w_v(t)$$

Using the following substitution we can write the a state-space representation of the system.

$$x = \begin{bmatrix} h \\ v \end{bmatrix} \qquad u = \dot{\ell}$$

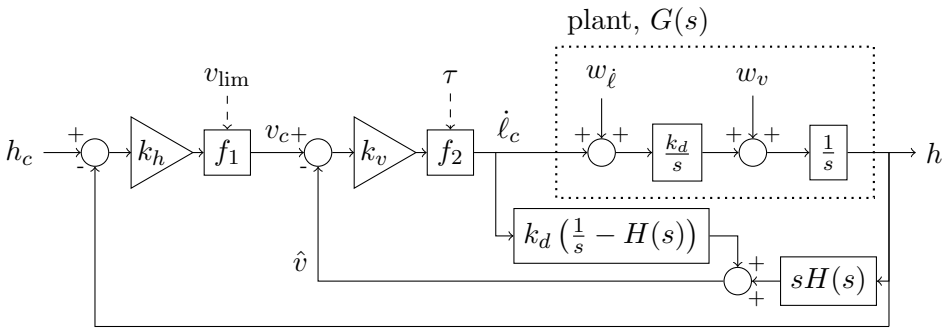
$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ k_d \end{bmatrix} u + \begin{bmatrix} w_v \\ k_d w_{\dot{\ell}} \end{bmatrix}$$

1.2 Control system

1.2.1 Overview

The altitude control system has an unconventional objective for a feedback control system. On every ValBal flight, there is an essential trade-off between tracking closely to a target and system ballast use, as ballast is typically the endurance-limiting factor on a flight. Furthermore, this trade-off can not be usually be conveniently represented as a quadratic relationship between state errors and control inputs to allow for an LQG-style optimal control design. In addition, a more complex SMPC-based optimal control approach would not be feasible due to the constraints that the control law run on a low power embedded processor. Further more, the control objectives are typically given (or well-approximated by) a range of equally equally appealing altitudes, with a lower and upper bound. For instance, on an endurance flight with no location objectives, the lower bound is set by aircraft altitudes and the upper bound is set by latex degradation balloon latex degradation.

As such a linear control law that is of the form $u = K\hat{x}$ for some $K \in \mathbf{R}^{1\times 2}$, and a state estimate $\hat{x}$, would not be sufficient. Instead, a nested feedback loop



$H(s)$	low-pass filter
$f_1(v\,; v_{\text{lim}})$	clamp on the velocity commanded by the altitude loop set by v_{lim}
$f_2(\dot{\ell}\,; \tau)$	deadband on the controller effort set by τ
h_c	commanded altitude (set by Flight Controller)
v_c	commanded velocity (output of position loop)
$\dot{\ell}_c$	commanded change in lift per unit time (output of velocity loop)
$w_{\dot{\ell}}$	atmospheric disturbances that change balloon lift (heating/cooling)
w_v	atmospheric disturbances that change balloon velocity (turbulence)
h	balloon altitude
$\hat{v}$	estimate of velocity

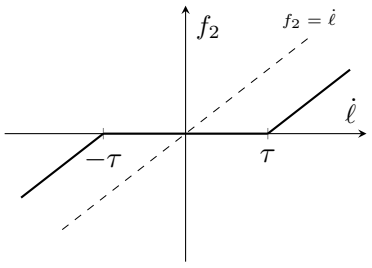
Figure 1: Altitude control system block diagram

structure is used, where an altitude loop commands a velocity loop which commands a change in lift, which allows for the easy addition of two key nonlinearities. The full control system structure is show in Figure 1.

1.2.2 Nonlinearities

The two non-linearities in the controller are given by f_1 and f_2. The first function $f_1(v\,; v_{\text{lim}})$ is simply a clamping function that limits the velocity commanded by the altitude loop to $\pm v_{\text{lim}}$. This prevents the controller from overreacting when it is far from it's command. The second non-linearity $f_2(\dot{\ell}\,; \tau)$ is a deadband on the output of the controller. It is defined by

$$f_2(\dot{\ell}\,; \tau) = \begin{cases} 0 & |\dot{\ell}| < \tau \\ \mathbf{sign}(\dot{\ell})(|\dot{\ell}| - \tau) & |\dot{\ell}| > \tau \end{cases}$$



The parameter τ is set by $\tau = k_h k_v e_{\text{tol}}$, where e_{tol} is the steady-state error in altitude allowable by the controller. This means, that with $v = 0$, the system can

be at an altitude anywhere in the range $h = h_{\text{cmd}} \pm e_{\text{tol}}$, without the controller acting. The controller can then be thought of as having two bounds, $h_{\text{lower}}, h_{\text{upper}}$ that are given by $h_{\text{lower}} = h_{\text{cmd}} - e_{\text{tol}}$ and $h_{\text{upper}} = h_{\text{cmd}} + e_{\text{tol}}$. As such, controller input commands can either be given in the form $(h_{\text{cmd}}, e_{\text{tol}})$ or $(h_{\text{lower}}, h_{\text{upper}})$. The values are either set by the human flight controller commanding the system, or a trajectory planning algorithm (described in section 2).

1.2.3 Velocity estimator

For control, it is important that we have a reasonable estimate of the vertical velocity of the balloon, $\hat{v}$. However, the atmosphere has a large amount of turbulence and waves, which have a dramatic impact on the short-term velocity of the balloon. For control on the altitude ranges that ValBal flies in, we do not care about these effects; what we actually care about is the component the velocity of the balloon that is due to the net lift. For example, it is possible for the balloon to be rising at 1 m/s, and still be neutrally buoyant, as it is simply floating atop a wave in the atmosphere. In this case, we would not want the controller to act to this in the same way it would to the balloon rising steadily at 1 m/s due to a positive net lift.

As a result, an estimator must be designed to calculate $\hat{v}$. A first order estimator such as a Kalman filter would suffice for this, if the noise on the velocity was Gaussian and uncorrelated in time. However, previous flight data and atmospheric literature [1] shows that this noise is dominated by large-amplitude waves at specific frequencies. Therefore, a second order low-pass filter, $H(s)$, is used to achieve better roll-off and reject the high frequency waves. In addition, actions of the controller are be integrated through the plan model to update the $\hat{v}$ estimate immediately after controller actions. The corner frequency of $H(s)$ is typically set around $(15 \text{ minutes})^{-1}$ and is determined by looking at the spectral content of atmospheric waves in prior flight data.

1.2.4 Choosing gains

Choosing gains, k_h and k_v, can be done through a linear second order system analysis of the system dynamic response. While the control system is non-linear, it is piecewise-linear, so for any given operating point a linear approximation remains valid.

The transfer function model for the system without nonlinearities is

$$T(s) = \frac{k_h k_v k_d}{s^2 + k_v k_d s + k_h k_v k_d}.$$

The damping ratio is then given by $\zeta = \frac{1}{2}\sqrt{\frac{k_v k_d}{k_h}}$. Since in general minimal ballast usage is preferred over other performance metrics such as rise time, we want to ensure no overshoot in the step response of the controller. As such, we need to pick a gain ratio such that $\zeta \geq 1$. Assuming that we won't overshoot, we would prefer to have a minimal rise time, so we want the lowest damping ration that is greater than or equal to one, which would be one. However, k_d, is an estimated

parameter we uncertainty associated with it, and since ensuring no overshoot is more important than minimizing rise time, we add in some margin. In practice, $\zeta = 1.2$ works well.

This gives use a ratio between gains, but not the magnitude of $k_h k_v$. However, picking the magnitude of the gains requires consideration of the deadband because, unlike a linear controller, a higher gain will not result in a closer tracking of h_{cmd}, that is set by the e_{tol}. Instead, the magnitude of the gains influences how aggressively the controller acts around the bounds. A high gain means that the controller will wait longer and act more aggressively near the bounds, and a lower gain means that it will act sooner but slower. In the limit of a small $k_h k_v$ and a small e_{tol}, the system will behave like a linear controller tracking h_{cmd}, and in the limit of a large $k_h k_v$, the system will behave like a bang-bang controller to command velocity to zero when the system reaches a boundary.

With no uncertainty in the system, high gain would result better performance as the system would wait until the last possible second to act. However, it would not be robust to noise in the system, measurement errors, or errors in the estimated parameters. While a robust controller analysis could be used to determine the best total gain magnitude, this is subject to realistic estimates of errors in the system which are rigorously defined. Instead, we model these errors in simulation, and tune this parameter using the simulation, described in the next section.

1.3 Simulations

Simulations are critical in the control system designs, as a means of testing new concepts, tuning parameters, checking flight-code for bugs, and for giving general intuition on the system, as each flight tests is expensive and it is not easy to radically change the control algorithm after the balloon is launched. While good control performance in simulation gives little guarantee on real-world performance, poor performance in simulation shows problems with the controller and misunderstandings of the system.

A simulation of the plan model shown in Figure 1 is implemented in C++ (the architecture is described in depth in section 3). Models for $w_{\dot{\ell}}$ and w_v are determined by analysis of previous flight data. The noise models contain white noise, nightfall/sunrise effects, atmospheric waves, and terrain elevation effects.[1]

A plot of an example simulation output is shown in **??**. In this instance, the white noise on $w_{\dot{\ell}}$ was relatively small, as this simulation was a test of the controller's response to nightfall.

1.4 Flight results

The control algorithm is always being tweaked and added to after each flight, so each flight uses a slightly different version of the algorithm. Also, as the platform is in development there are typically a number of anomalies each flight. The

[1] Waves and terrain elevation have not been added yet but are currently being analyzed.

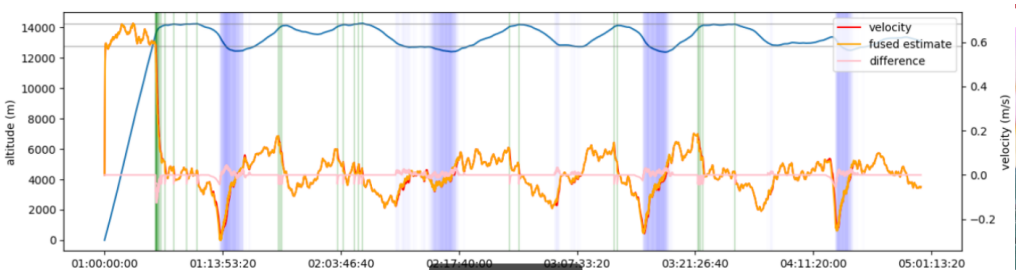


Figure 2: Plot of a simulated ValBal flight, showing altitude (blue, left vertical axis) versus time, along with various velocity terms (right vertical axis). Valve events (green vertical lines), ballast events (blue vertical lines), and controller bounds (gray horizontal lines) are shown.

below plot shows a 1 day segment of flight from June 2018. On this flight, there were sensor issues during the day, causing bad altitude measurements.

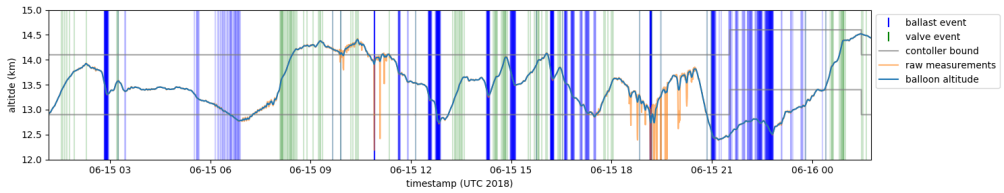


Figure 3: Plot of a real ValBal flight, showing altitude (blue) versus time. Raw measured altitude is shown as well (light red) where it at times is sporadic. Valve events (green vertical lines), ballast events (blue vertical lines), and controller bounds (gray horizontal lines) are shown.

During the period of bad altitude measurements, which can be seen where the light read does not match the blue altitude line, the controller id shut off by an automated system, and no re-enabled until a human flight controller manually turns it back on.

This was one of the earlier flights of the algorithm, and from it, we can notice a few things. First, nightfall occurred at approximately 6-15 20:00, causing the balloon to loose a lot of lift and ballast a lot. It also fell further outside the bounds than desired, and as can be seen, the controller bounds were temporarily manually raised in attempt to get the controller back into the desired range. This issue has since been mitigated by including the estimated effect of nightfall in the controller and adding that estimate into the controller's velocity estimator.

Another thing that we can see from this plot is that during the relatively turbulent period around 6-15 15:00, the controller acted sooner than necessary to disturbances, causing it to waste ballast. this can be mitigated through tuning of the gains.

This is still in progress, and a hefty section to write about, so I decided to leave it out of this document for now

2 Trajectory planning

2.1 Introduction

This section has gotten a bit long, but I wanted to lay out our reasoning for our choices. I only included three key citations to previous paper on high altitude balloon planning, as I only wanted to cite the most relevant work and be able to talk about each in a bit more detail, rather than just listing a bunch of tangentially-related work.

ValBal [2] [3] has no direct control over its lateral movement around the globe, but rather is always drifting with the air currents of Earth's atmosphere. However, by changing the altitude that it flies at, it can choose different wind currents moving in a range of directions in a given column of air. In addition, being a low-cost and small balloon platform, ValBal's control authority is a finite resource that ultimately limits the duration of most missions. The high-level goal of trajectory planning is to come up with a set of altitudes that the balloon should fly at in order to achieve some mission outcome in the lateral trajectory of the balloon, conserving the amount of ballast used such that the flight does not end prematurely. Variations of this problem have been previously tackled in a number of a ways, primarily relying on either discretization of the state space to handle the nonlinear nature of the wind field, or linearizing a portion of of the wind field to simplify the processes of obtaining a control law.

A recent method for station keeping was proposed in [4], which utilized a receding horizon tree search with a discretized state space to find altitude sequences that would keep the balloon in one lateral location. This approach uses Gaussian Processes to model uncertainties in the wind field, and a novel method to discretize altitude ranges into currents of air moving in similar direction. In 2010, this problem was tackled for Montgolfiere balloon navigation on Titan [5], in which the surface of the planet was discretized into regions and a Markov decision process (MDP) was used to solve for minimum Time-to-Goal. Transition probabilities between discrete states were determined through a Monte Carlo method on the wind field distributions.

In both of the above methods, the authors do not worry about minimizing the control action of the balloon. For large platforms with a regenerative method of adjusting lift this is sufficient, but for many objectives to be pursued by a ValBal, running out of ballast is a limiting factor on performance. As such, and trajectory planner for this platform must take ballast use into account. In a paper from 2003 [6], the authors attempt to solve the problem of getting the balloon to a target location while minimizing fuel consumption of the balloon (in this case it's hot air balloon, in which fuel is analogous to ballast on ValBal). They do so by solving the two-point boundary-value problem given by the Hamiltonian system of an optimal control problem that they formulate. However this formulation relies on linearizing the wind field in order to be tractable, which works in their case as they only attempt to solve the problem on instances on the order of kilometers in size, in which a linear approximation for the wind field works. The authors also to not account for uncertainty in the wind field.

For our problem, we must deal with the non-linear wind field as we operate over significant portions of the globe. In addition, we must account for not only uncertainty in the wind field, but also uncertainty in the altitude of the balloon, as it is impossible for the altitude controller to efficiently track a precise commanded altitude. Furthermore, the planner needs to keep track of at least one additional state: remaining ballast, as this effects how long the balloon can stay aloft. We also would like to retain high precision over states so that we can aim to fly the balloon with kilometers of desired destinations. This makes discretization of the state-space much more difficult that previous attempts, as we have more dimensions and desire fine precision.

So, we decided to take a different approach, using two key insights:

1. All dynamics of the system are fundamentally continuous and smooth due to the nature of wind being a continuous vector field.

2. The wind field of the atmosphere is highly structured, as it is the result of macro-scale thermodynamic and inertial effects on the globe. As such, large regions of the globe will have winds all moving in the similar directions with slow changes

The first point means that the dynamics of the system are easily integrable by simple Euler integration, as the second derivative of the dynamics are well-behaved. The second point leads us to hypothesize that most reasonably formed objective functions in the lateral location of the balloon will have much fewer local minima with respect to the altitude parameters than there are possible states of the system. That is to say, while we may care about the position of the balloon to a precision on the order of kilometers, the wind field structure likely results in there being only a relatively small number of locally quasi-convex regions in the space of control parameters. Consider these points in contrast to say a robot motion planning problem in a room with objects, which may be highly non-smooth and unstructured, where a search or value iteration on a discretized system often works well.

I don't know how to say this well, but I think it's an extremely important point and is precisely the reason that it makes sense to solve the problem the way that we are solving it

So, our proposed solution to this problem is to use a fully continuous state and discretized time to perform a gradient-based optimization over the control parameters. We computing trajectories of the system in an analytically differentiable manner (implemented efficiently in software with automatic-differentiation). Expectations of the trajectory under the various forms of uncertainty are obtained through Monte Carlo sampling and average. This formulation is both flexible to a range of different models for various effects on the altitude of the system, as well as computationally tractable for finding locally-optimal trajectories. It's major drawback is that it does not solve for a global optimum, but this mitigated by running the optimization from a set of random initial conditions, in attempt to find all local minima for a given scenario.

We would then apply this in a Stochastic Model Predictive Control fashion, at each planning timestep of the system we take the current state of the balloon, optimize the control parameters entire duration or the mission, and then upload them to the balloon.

2.2 Formulation

The physical system is represented by continuous states $s \in \mathcal{S}$ defined at discrete times $t_k = t_s + k\Delta t$, for $k = 0, 1, \ldots$ In the case of ValBal, we can let $\mathcal{S} = \mathcal{H} \times \Lambda \times \Phi$, if we're considering altitudes, latitudes, and longitudes; we could extend this to include the ballast levels $\mathcal{B}$, or lifts $\mathcal{L}$.

The altitude changes with respect to time according to the setpoint altitudes given to the control algorithm, with some randomness due to atmospheric noise. Latitude and longitude change according to Euler integration of the wind field, evaluated at every given time, altitude, and coordinates. Our goal is to find a control policy policy to maximize some objection function, which might, for instance, be the total horizontal distance traveled.

This results in another advantage that comes from keeping the state space continuous. By picking some fixed set of parameters θ (that might be the setpoint altitudes and tolerances at various points of the flight) and integrating the trajectory as a function of those parameters, we can take the gradient of the cost function (that is a function of the trajectory) with respect to the parameters, and optimize our objective. We expand upon a formulation in the following section.

2.2.1 Differentiable altitude trajectory

Ultimately, our goal is to come up with some continuous and differentiable function $V(\theta)$ that gives the value of the objective function. However, in many instances, the value of the objective function will be related to the integration of the trajectory—e.g. distance traveled, or final longitude. This means that we need to come up with a formulation of the trajectory that depends on the optimization parameters and is differentiable with respect to them.

The parameters that form θ can, for instance, be a set of altitude waypoints defined at various points of the flight, or altitude tolerances at each of those points. If our state consisted only of altitudes, and we were to simply linearly interpolate between altitude waypoints $\theta_0, \theta_1, \ldots$ (defined at $t_0^w, t_1^w, \ldots$) we would find the parameters relevant to the current timestep t_k such that $t_i^w < t_k < t_{i+1}^w$. Then we simply give:

$$h(t_k; \theta) = \theta_i + (t_k - t_i^w)\frac{\theta_{i+1} - \theta_i}{t_{i+1}^w - t_i^w}$$

which has a very simple derivative with respect to the optimization variables θ. However, this assumes that we follow *exactly* the trajectory defined by the optimization parameters. In reality, the value function $V(\theta)$ will be taken over an expectation ensemble of possible trajectories under noise.

In other words, our trajectory integration needs to output a distribution of possible states it might end up, while remaining differentiable. We introduce a parameter γ that can be interpreted as a percentile of the noise distribution that a given trajectory is under. At the beginning of a rollout, a sequence $\gamma_0, \gamma_1, \ldots$ is randomly generated such that the variation results in a noise level consistent with flight data. In the limit where all γ are 0.5, the median trajectory is selected; in a more realistic simulation, the percentile would slowly fluctuate between smaller

and larger numbers such that a particular rollout would be flying low or high respectively with respect to a noise-free simulation. For instance, in the scenario given above, we could introduce $\gamma(t)$ as a random walk bounded between -500 m and 500 m, defined between t_0 and t_f. Then our position is given by:

$$h(t_k;\theta) = \theta_i + (t_k - t_i^w)\frac{\theta_{i+1} - \theta_i}{t_{i+1}^w - t_i^w} + \gamma(t_k)$$

This allows us to consider a stochastic evaluation of our value function that remains differentiable.

$$\tilde{V}(\theta) = \sum_{\text{random } \gamma} F(t_f, \gamma; \theta)$$

2.2.2 Differentiable horizontal trajectory

The section above only handled altitude; however, the objective function for ValBal relies on maximizing spatial parameters, and not altitude. The horizontal integration needs to preserve differentiability to be able to compute a full gradient with respect to every optimization parameter. We use atmospheric data provided by NOAA [7] which, in particular, includes the velocity field $w(h, \lambda, \phi, t) \to (u, v)$, where h is the altitude defined on a set of forecast altitudes;[2] λ and ϕ are the coordinates defined on a (not necessarily regular) grid of latitudes and longitudes;[3] t is a forecast time (usually separated by 1 to 3 hours) and (u, v) are the two horizontal components of wind.

For completely regular grids, bilinear interpolation will suffice. For irregular grids, a scheme such interpolation weighted by inverse distance $(\sum^K \alpha_i w_i)/(\sum^K \alpha_i)$ for the closest K points would do the job. The weights of each nearby point are a function of altitude, latitude, and longitude; therefore, when we take the weighted sum, we preserve a differentiability path through the (differentiable) altitude. In other words, the final velocities (u_k, v_k) are themselves a function of θ, which we use to find the new coordinates:

$$x_{k+1} = x_k + u_k \Delta t \quad y_{k+1} = y_k + v_k \Delta t$$

and the conversion to (λ_k, ϕ_k) follows from a simple computation of spherical coordinates.

The viability of this method, as well as the quality of NOAA data can demonstrated by comparing the integration method and wind model to previous balloon flight data. To do so, we simulate a number of ValBal flights with added noise at the date and time of previous ValBal flights, using the same launch location and altitude profile. We then compare the results with the observed flight, to see that the observed flight matches the simulation. This comparison is shown in figure 4.

[2] The data from NOAA is defined at barometric pressures, which can be mapped monotonically to altitude.

[3] Some models are more regular than others; for instance, NAM is defined on a rather annoying Lambert conformal grid.

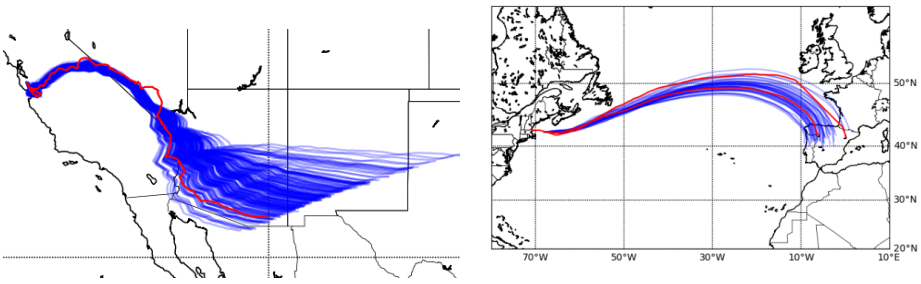


Figure 4: Comparison of real ValBal flights (red) to ValBal flights simulated using GFS data with added noise (blue).

2.2.3 Differentiable value function

We can consider a variety of value functions. If we're trying to maximize the horizontal function, we can use the differentiable longitude computed in Section 2.2.2. For a rollout of length K, the total value is the horizontal distance:

$$V = \sum_{k=0}^{K-1} u_k \Delta t \tag{1}$$

where Δt is a constant that does not affect the result. Other valid objective functions could include the distance to a particular point:

$$V = \sum_{k=0}^{K-1} \|r_k - r_{\text{target}}\|$$

2.2.4 Simple example

To illustrate how the sections above can be built up to a completely differentiable objective as a function of flight parameters we give a simple example. Suppose, as is often the case, that our goal is to maximize longitude traveled eastwards (which corresponds to attempting to circumnavigate as quickly as possible), after N simulation steps (say, $N = 720$, which with Euler timesteps of 10 minutes corresponds to 5 days). Said final longitude will be the result of the Euler integrated rollout, so the value function is given by:

$$V = \lambda_N$$

Let's build up how we would compute λ_N. In a certainty equivalent simulation, altitude as a function of time is given by:

$$h(t;\theta) = \begin{cases} \theta_1 + (t - t_1)\frac{\theta_2 - \theta_1}{t_2 - t_1} & t_1 \le t < t_2 \\ \cdots & \\ \theta_{n-1} + (t - t_{n-1})\frac{\theta_n - \theta_{n-1}}{t_n - t_{n-1}} & t_{n-1} \le t < t_n \end{cases}$$

Here θ are the parameters we are optimizing over, defined at the points $t_1, \ldots, t_n$. We will also need a way to compute the winds at a given point in spacetime.

We can do so by looking at the grid element the balloon is at from the NOAA wind data. We linearly interpolate in altitude at each corner of the grid element (thereby preserving continuity and differentiability), then bilinearly interpolate in latitude and longitude (also continuous in all variables), and finally linearly interpolate in time for the two NOAA prediction times (usually separated by 3 or 6 hours). That gives us two functions $w_{\{\lambda,\phi\}}(\lambda, \phi, h, t)$, since the wind is defined in each coordinate direction.

Having defined these helper functions, we are in a position to trace out the rollout. Say we start at some initial point $(h_0, t_0, \phi_0, \lambda_0)$. For the first timestep, we will compute:

$$\begin{cases} t_1 &= t_0 + \Delta t \\ h_1 &= h(t_1; \theta) \\ \lambda_1 &= \lambda_0 + \Delta t \cdot w_\lambda(\lambda_0, \phi_0, h_0, t_0) \\ \phi_1 &= \phi_0 + \Delta t \cdot w_\phi(\lambda_0, \phi_0, h_0, t_0) \end{cases}$$

Note that, for now, only h_1 is a function of the optimization parameters. At the second timestep, however, we will compute:

$$\begin{cases} t_2 &= t_1 + \Delta t \\ h_2 &= h(t_2; \theta) \\ \lambda_2 &= \lambda_1 + \Delta t \cdot w_\lambda(\lambda_1, \phi_1, h_1, t_1) \\ &= \lambda_0 + \Delta t \cdot w_\lambda(\lambda_0, \phi_0, h_0, t_0) \\ &\qquad + w_\lambda\left(\lambda_0 + w_\lambda(\lambda_0, \phi_0, h_0, t_0)\Delta t, \phi_0 + w_\phi(\lambda_0, \phi_0, h_0, t_0)\Delta t, h(t_1; \theta), t_1\right) \Delta t \end{cases}$$

and similarly for ϕ_2, the latitude at the second timestep. Note, now, that our latitude depends on the optimization parameters, by way of $h(t_1)$, and we can therefore compute $\partial\lambda_2/\partial\theta_i$. In subsequent timesteps, this dependence gets exponentially more complicated, because then ϕ and λ themselves have dependence on θ. Doing so manually would be rather tedious, which motivates the decision explained in Section 3 below to use an automatic differentiation tool to keep track of these gradients in the implementation.

As mentioned, this is the most simple of the possible formulations; however, it is unrealistic in that it does not account for noise and variance—the balloon can't hit linearly interpolated altitude waypoints exactly (not without wasting tonnes of ballast, anyway). It makes sense to reformulate it in terms of the control parameters h_{cmd} (setpoint) and e_{tol} (tolerance). One first formulation would be to approximate the balloon as being somewhere in $h_{\text{cmd}} \pm e_{\text{tol}}$ randomly. However, the position within the tolerance is correlated between timesteps: if the balloon was at $h_{\text{cmd}} - 0.98e_{\text{tol}}$ at $t = 0$, it would be unlikely to be at $h_{\text{cmd}} + 0.5e_{\text{tol}}$ at $t = 10$ min. Our solution is to perform (random) simulations of balloon altitude (as shown in Sec 1.3) with an arbitrary fixed setpoint and tolerance (giving us random walks $\gamma(t, h_{\text{cmd},0}, e_{\text{tol},0})$, and then affinely scale it to the control parameters.

This refined formulation for altitude as a function of time, used in the stochastic

simulations below, is given by:

$$h(t;\theta,\gamma) = \text{LinInterp}\left(t, t_i \to \theta_{2i} + (\gamma(t; h_{\text{cmd},0}, e_{\text{tol},0}) - h_{\text{cmd},0})\frac{\theta_{2i+1}}{e_{\text{tol},0}}\right)$$

Note that now we have two parameters per timestep: θ_{2i}, the setpoint, and θ_{2i+1}, the tolerance.

2.3 Preliminary results

2.3.1 Certainty equivalent

Before running the full stochastic gradient descent on the Monte Carlo problem, we developed and tested the algorithm on certainty-equivalent simulation with no noise. In this experiment, we optimized a single starting trajectory, assuming that we had complete control over 20 altitude way points, with the balloon linearly interpolating between them. The results of such a test are shown in Figure 5.

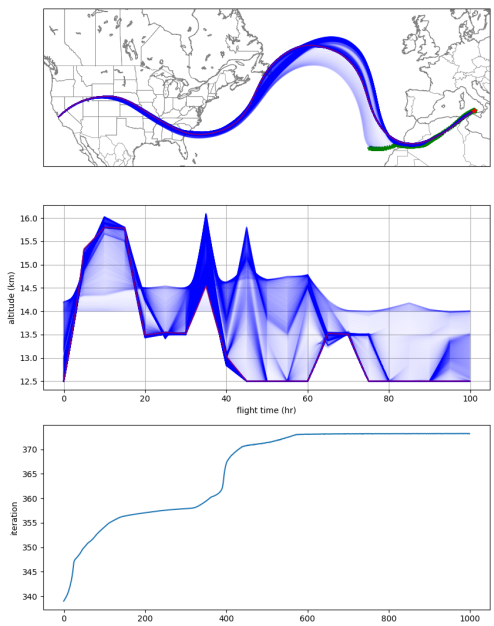


Figure 5: Certainty equivalent optimization from a single trajectory of the total longitude traveled. Trajectories are depicted both in latitude vs longitude over a Mercator projection (top), and altitude vs time (middle). The launch site simulated was Hollister, California and flight endpoints are designated with a green star. Intermediate trajectories are shown in blue, with the final optimized result in red. A convergence plot of the objective function as a function of iteration number is shown (bottom).

We can see from this Figure 5 that the optimization behaves as expected. The final longitude slowly moves forward each iteration, until it reaches a local extremum. The path of green points over the map—the endpoints of each intermediate trajectory—always moves to the right as iteration number increases. Running a thousand iterations of certainty equivalent optimization on a 100 hour flight as in the example takes well under a second on our hardware.

2.3.2 Monte Carlo runs

To run our first test of this formulation, we opted to utilize the traditional Model Predictive Control approach, in which we optimize an open-loop policy over a finite horizon. In flight, this optimization would be run repeatedly as the balloon

flies, using the most recently observed system state to optimize new trajectories, forming a feedback control law.

Results of this method on a sample of NOAA analysis wind data are shown in Figure 6.

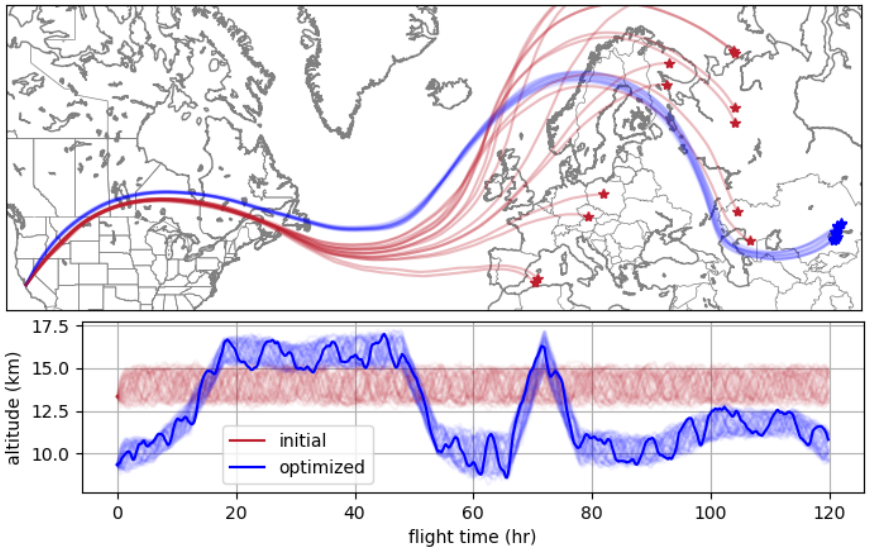


Figure 6: Example result of optimizing an open loop trajectory. The initial settings were a 13.5 km setpoint and a 0.75 km tolerance (red), while the optimized trajectory shows both changing setpoints and tolerances (blue). The dark blue line gives an example possible altitude profile from the many possible altitudes profiles given the command settings.

We notice a few things from this result. First, we note that in this particular scenario the initial commands resulted in a large variation in the resulting flight path. This was due to large changes in wind direction within the command tolerance over the Atlantic Ocean. We see that in the optimized open-loop trajectory, policies are picked such that there is not much variation in the resulting outcome.

2.3.3 Planner simulation

To test performance of a the planner using GFS prediction data, we implemented a simulation of the control system what would run a single trajectory evaluation simulation and at each planning timestep call the planner. The evaluation simulation would be ran with GFS analysis data. This provides us with an expectation of what performance with our planners may look like in real life.

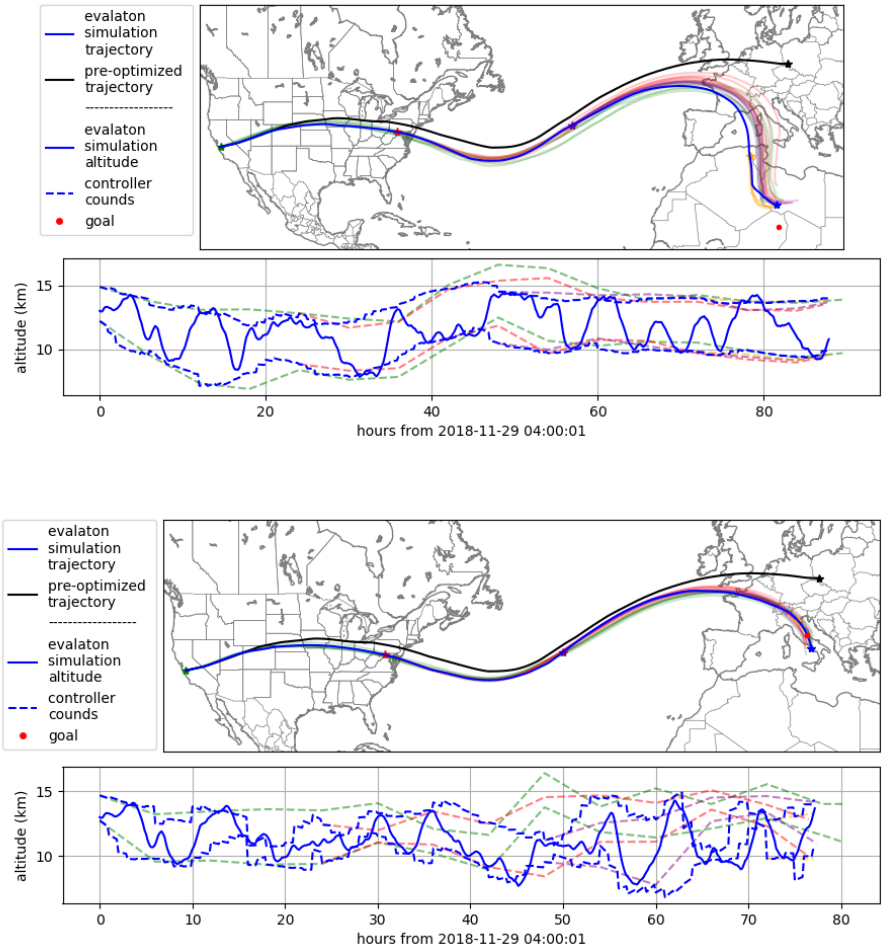


Figure 7: Evaluation simulations for SMPC planners. Each set of plots shows a single simulation ran with GFS analysis data, using control commands generated by SMPC using only prediction data. The non-blue colored lines show MPC simulation runs and optimized controller command plans from the time step they were ran at.

From the simulations shown in Figure 7, we can immediately notice a few things. First, for a point that is outside the ballast-limited range of the system (top plots), we see that the planner commands relatively loose tolerances throughout the flight. This is to be expected, as if the flight is ballast-limited and the

destination is just out of range, any conserved ballast will extend the flight time, thus bringing the balloon closer to the destination. However, if the bounds are too loose, then final endpoints will be too spread out, resulting in a higher average expected distance to the goal. The planner attempts to find the ideal trade-off point to minimize expected minimum distance to the target, and it appears to do so quite well.

Next, in the case where the target is within ballast-limited range, the planner is free to command much tighter tolerances when needed, as there is no benefit to having ballast left over when reaching the desired location. As such, we can see large variations in the commanded tolerance throughout the flight, being very small in places where it needs to hit specific winds to come as close to the target as possible. In this simulation, the evaluation flight comes within a few km of the target location, a promising result future location-sensitive applications of the system.

2.3.4 Location-feedback altitude policy

One limitation of the procedure given above is that we only have a single optimization variable for a given timestep. However, due to the inherent variance in the rollouts of the flights, two rollouts that were once close early on in the flight might wind up taking different wind patterns in the atmosphere, or one might slow down and lag behind the other. And, if their coordinates are very different, the guiding assumption for the algorithm—that the atmosphere is not totally random, and we can therefore take gradient steps to optimize our trajectory—will no longer hold: the algorithm will have to make a compromise between the winds in two potentially very different areas of the globe.

We can see an example of this in Figure 6, in the pre-optimized trajectories: after five days of flight, they are in very different places: the winds in the Mediterranean ocean and in northern Russia will be very different, so it will yield a meaningless unoptimized result. This led the single-variable-per-timestep optimizer to tighten the tolerance, in order to reduce variance and make the optimization meaningful again. However, this is wasteful, since in order to do that, it thought it had to tighten the tolerance and thereby waste ballast late in flight. That variance in the rollouts merely reflects the uncertainty in where the balloon will end up, but in reality, as time advances, our knowledge of the position of the balloon is updated, and we know what trajectory the balloon ends up following. In a sense, we want to be able to discount this future uncertainty, and assume that, as time passes and we know which way the balloon goes, we'll upload the best possible trajectory going down a given path (out of the various possible trajectories).

This motivates the need to incorporate location feedback to our policy. For each of the possible final paths (as they diverge, for instance, by picking the clockwise or counter-clockwise sides of a hurricane) we create a different variable on a given timestep. This way, with more degrees of freedom, the optimizer will make sure that any scenario we possibly end up with will have the most optimized trajectory.

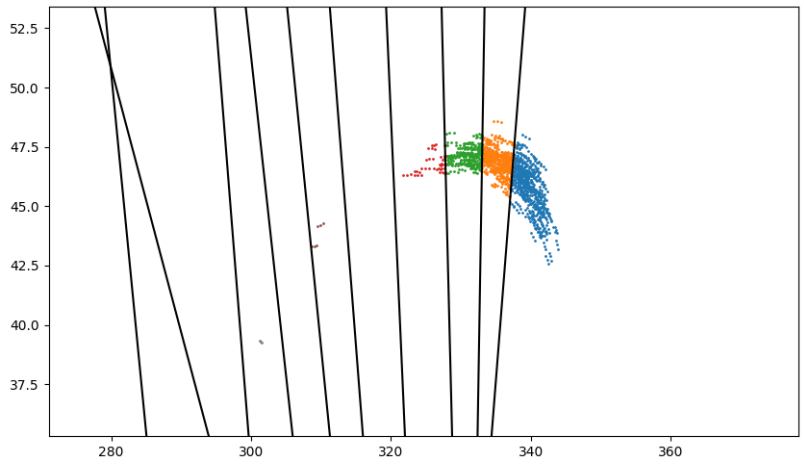


Figure 8: Spatial policy for a given timestep of the simulation. The dots are requests of each variable at that timestep; dots with the same color used the same leaf node. Note that each individual rollout makes three requests (we can see small clusters of three dots); however, we did a hundred rollouts, so they end up having some spread. We see that some rollouts lag behind the main group, and therefore are optimized using a different set of variables.

In theory, given enough rollouts, we could simply assign one latitude and longitude to every one by one degree latitude-longitude grid element. This would reduce to solving the completely discretized Markov Decision Process; however, it would come with an exponential blowup in the number of variables, losing the advantage gained by using gradient descent. We want to create new variables lazily, only as they're needed: if we see that, for a given timestep, all the trajectories are in roughly the same place, there is no need to create new variables. However, if they start to diverge, we'll want to make sure that each branch has its separate variable, so the trajectories don't interfere with each other's optimization.

The algorithm we built to lazily create new variables uses a binary space partitioning of the globe, which creates a tree of halfspaces, with a variable on each halfspace. We start with a single node of the tree, and keep track of all instances of the usage of that variable (i.e. for each rollout, see what coordinates the balloon was at when it made use of that variable); when those requests get too far (measured by the half perimeter of the rectangular hull of the requests), we decide to create a new variable. In order to decide what hyperplane gets used to create two halfspaces, we use k-means to cluster the points into two different clusters: the Voronoi description of the resulting two center points will give us a halfspace from which we create an upper and lower node, initialized with the current value of the parent node. Subsequent variable requests at that given timestep will walk down the tree until they reach a leaf, and use the variable stored at the leaf. At the end of each iteration, if any of the leaves has diverging trajectories, it gets split up according to the procedure above.

Preliminary tests show that more work is needed on deciding how to split up

the space. Figure 8 shows a sample result from using the method above. Note that it is displayed flat, instead of as a tree. Each color corresponds to a different leaf node of the binary space partitioning; the lines correspond to the dividing hyperplanes. For now, in limited testing the performance is not worse than the temporal-only method; however, we have not seen instances of it outperforming it either. Further work will try to improve the splitting method, and try it out in weeks where the divergence of the wind field was bigger (since the method is not appreciably better if the wind field is rather uniform).

Further extensions include further splitting according to other variables, such as ballast usage. That is, if we're at the same time and coordinates, but we have a lot more ballast in one rollout, it would make sense for that rollout to be less conservative about its use of ballast, whereas if it's running low it will want to loosen the tolerance.

2.3.5 Flight tests

As we don't have quality plots from the last flight, we didn't write up things about it in this section as we expect to have a much better opportunity to do so next quarter. For plots, see the presentation slides we made about it.

3 Architecture

In this section we describe the architecture behind the simulations and trajectory optimization. It involves a complex pipeline, that involves fetching atmospheric data, a modular and fast simulation engine, and provisions for simulating with real flight code. It is mostly written more than three thousand lines of C++ and Python.

The core of the simulator is written in C++. This allows for extremely fast simulations; when performing direct rollouts, it can compute hundreds of thousands of three-day ValBal trajectories per second on a 6-core machine. The code is designed to be modular; one can easily use different objective functions (horizontal distance, distance to a given target...), different optimization methods (vanilla gradient descent, Adagrad, Adam...), various differentiable controllers (uniform distribution, approximate Lasagna, differentiable Lasagna...), and so on.

Having been designed with optimization in mind, another key feature of the simulation codebase is the formulation in terms of C++ templates, that allow the compiler to, from a single codebase, generate code that either merely computes rollouts of the trajectories, or keeps track of the gradients leading to the final objective function. The latter uses *automatic differentation*, the same algorithm used in the training of large neural networks. It keeps track of the computation graph and derivative at each node; once the final objective function is computed (say, the final longitude), the gradient is propagated backwards to each parameter, and some first-order optimization method is used to update the parameters. By using the Adept library, the parametrized codebase can be specialized to use Adept's differentiable floating point type, that keeps track of the computation graph. Doing so incurs an overhead, of about 5 to 10×, but is able to provide the exact gradient of the objective with respect to the parameters, in a much easier way than direct numeric differentiation, and without having separate code paths for direct evaluation and gradient evaluation.

We also built several Python scripts to fetch the data and process the results. We can retrieve wind data from either the Global Forecast System (GFS) model from NOAA, or the European ECMWF model. We preprocess the data, that comes in GRIB files, into binary files that are loaded directly by C++. This is key to the speed of the C++ codebase, by using the `mmap` system call from Linux, the data will end up cached in the comparatively faster RAM, and wind table lookups require merely the computation of an offset (as opposed to searching through a database, with various levels of indirection). In particular, we only store relevant variables at the altitudes ValBal normally flies, in order to save space and improve cache performance. Several Python scripts are used to plot the trajectories found by the C++ script (saved to a binary file) using the Matplotlib library.

References

[1] David C Fritts and M Joan Alexander. "Gravity wave dynamics and effects in the middle atmosphere". In: *Reviews of geophysics* 41.1 (2003).

[2] Andrey Sushko et al. "Low cost, high endurance, altitude-controlled latex balloon for near-space research (ValBal)". In: *Aerospace Conference, 2017 IEEE*. IEEE. 2017, pp. 1–9.

[3] Andrey Sushko et al. "Advancements in low-cost, long endurance, altitude controlled latex balloons (ValBal)". In: *2018 IEEE Aerospace Conference*. IEEE. 2018, pp. 1–10.

[4] R. D. Born and M. Schwager. "Riding an Uncertain Wind Field: Receding Horizon Tree Search Planning with Opportunistic Sampling for an Autonomous Weather Balloon". In: *Proc. of the AIAA Science and Technology Forum and Exposition (AIAA SciTech 19)*. Jan. 2019.

[5] Michael T Wolf et al. "Probabilistic motion planning of balloons in strong, uncertain wind fields". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 1123–1129.

[6] Tuhin Das, Ranjan Mukherjee, and Jonathan Cameron. "Optimal trajectory planning for hot-air balloons in linear wind fields". In: *Journal of guidance, control, and dynamics* 26.3 (2003), pp. 416–424.

[7] *NCEP GFS 0.25 Degree Global Forecast Grids Historical Archive*. Boulder CO, 2015. URL: `https://doi.org/10.5065/D65D8PWK`.