Loon Vehicle Sizing Tool

Technical Documentation

Introduction

This is a technical document describing the theoretical aspects of the sizing tool, as well as providing rationale for design decisions made in the development of the sizing tool. Sizing tool is used by Loon's system engineering team to optimize physical parameters of Loon's flight vehicles for a given objective. For example, the objective could be availability, payload power, cost, etc. The physical parameters that are tuned during this process are grouped into the following subsystems: aerodynamics, avionics, battery, down connect, envelope, flight system, LTE, mission, recovery, seahorse, solar, stability and control.

In order to incorporate all subsystems into a consistent, payload carrying vehicle, the sizing tool takes on an optimization approach to designing Loon's flight systems; this is known as **multi-disciplinary design optimization (MDO)**. Techniques from MDO are commonly used in the engineering design cycle of complex and safety-critical systems, in particular, aircrafts and spacecrafts. In addition to allowing designers to systematically consider all relevant disciplines, MDO provides a framework for design engineers to leverage the fast growing body of algorithms available in black-box optimization and machine learning.

Purpose

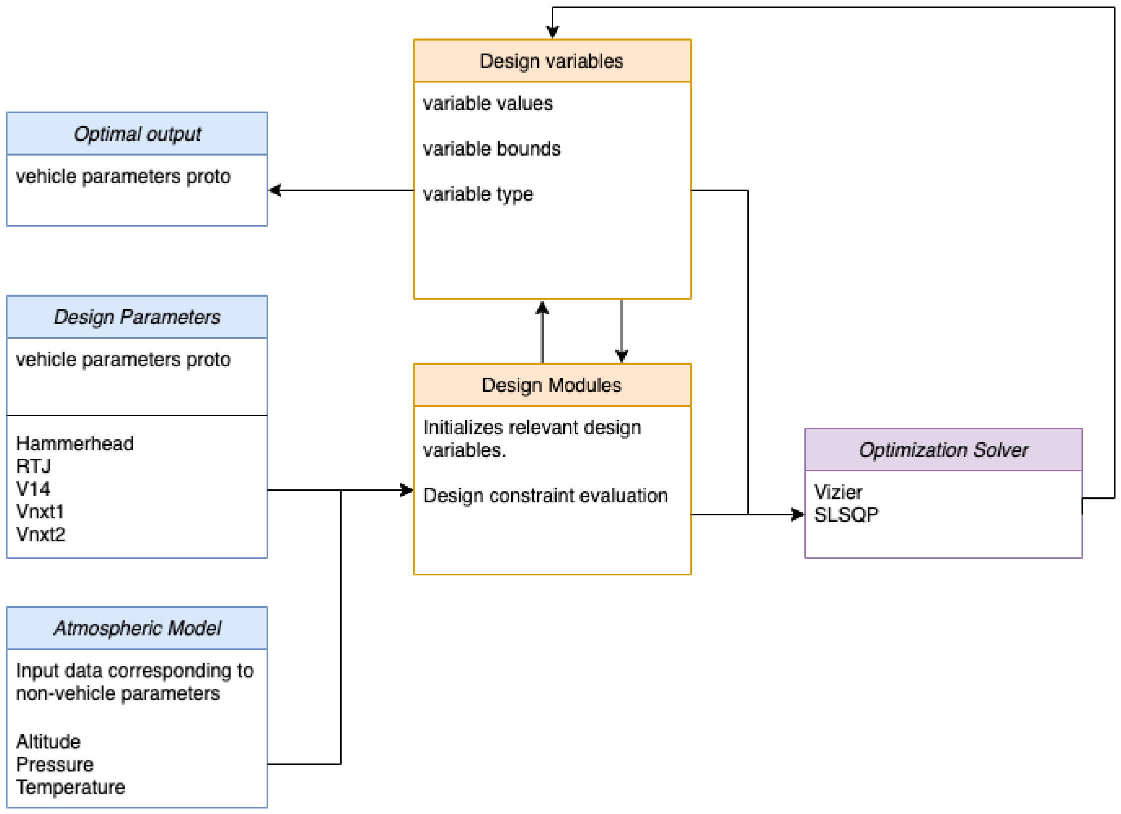
The sizing tool is Loon's flight system design process formulated as a MDO problem and developed as a software package. Three separate perspectives are taken to illustrate how the sizing tool fulfills this purpose:

1. **Software:** how to structure MDO elements into a software package with robustness, extendability and parallelizability in mind.
2. **Subsystem models:** how different components of a flight vehicle are modeled. In particular, we give a comprehensive documentation of subsystems Envelope, Mission, Battery and Avionics.
3. **Optimization**: how to solve MDOs from a mathematical framework.

We start with a detailed discussion on the coding architecture that supports the MDO problem, subsystem models utilized by MDO, and the optimization algorithms utilized to solve the MDO. Finally, a list of future improvements that could be made to improve the sizing tool is included in the appendix.

MDO/Software Architecture

The software structure for Loon’s sizing tool directly reflects the MDO problem formulation and approach taken to solve the MDO. The main components within software architecture are shown below.



The sizing tool imports **design parameters** - a set of parameters defining a flight system candidate, and the **atmosphere model** - a set of altitude/pressure/temperature profiles defining the environment for which a flight system is optimized. These are used to initialize the intermediate objects: design variables and design modules. Both are inputs to an optimization solver of choice, which iteratively modifies the value of each design variable while taking feasibility constraints into consideration, and finally outputs a set of optimal outputs.

Physics/Data-driven models

The sizing tool uses models to approximate subsystems to an accurate enough representation while still maintaining reasonable computational complexity. This trade-off allows designers to see how changing design parameters affect overall objectives without losing too much time in computation.

These models are sorted into **DesignModule** classes, based on the subsystem they belong to. Broadly speaking, most DesignModules have one or two surrogate models which the sizing tool iterates over to derive feasible designs. Details of surrogate models will be explained further under subsection Sizing Subsystems.

Design Variables

Within MDO literature, a design variable is a parameter involved in the design process that is directly controlled by the designer. Within the sizing tool software, we expanded this definition to include parameters that are not directly controlled by the designer but are relevant to design considerations.

1. **Input design variables** - design parameters that are directly controlled by the designer. Each design variable in the sizing tool has a continuous and bounded range. Examples include: length and radius of the envelope, number of solar panels.
2. **Output design variables** - design parameters that are NOT directly controllable by the designer. These are design variables that depend on one or more input design variables, and are either required in calculating the objective or required in representing inter-disciplinary dependencies. Examples include the TWR - which is an important system sizing objective, and system mass - the sum of masses across different subsystems including battery, solar, envelope, etc.

Within the sizing tool, design variables are stored within a single custom type dictionary, **Protected_Type_Dictionary**. Each DesignModule uses its surrogate models to modify current values stored within the design_variables dictionary to ensure that the parameters are consistent across different DesignModules.

Design Constraints

Design constraints are conditions that must be satisfied for a design to be feasible. For example, leak rate of an envelope must not cause the flight system to fall below the zero pressure margin within its designed life span.

Within the sizing tool, design constraints are stored as **DesignModule member functions**, which can be called on to evaluate the design feasibility. We store design constraints within individual DesignModules because most of the time, design constraints only require parameters of a specific module. Constraints that are interdisciplinary - such as power and mass - are stored in DesignModules that do not correspond to a physical subsystem such as Flight System and Mission.

Design Objectives

Design objectives are metrics that evaluate how well a flight system candidate performs. In MDO, design objectives are usually maximized or minimized to achieve the *optimal* flight system candidate. Examples of objectives of the sizing tool include availability, the ability of a flight system candidate to perform basic stationkeeping, and profit, the total cost of a flight system candidate. In addition to being optimized individually, design objectives can be optimized in sets to achieve pareto fronts.

The objectives currently implemented within the sizing tool are:

1. Duration: the lifespan of a flight system candidate.
2. Time
3. Speed: the average flight system speed.
4. Speed-prod:
5. Availability: the flight system’s ability to do basic stationkeeping.
6. Profit: the profit of a flight system, taking into account the amount of bytes it can serve.
7. Cost: total cost of producing the flight system.
8. Cost_per_area: the cost per coverage of a geographical area unit, equivalent to total cost/area_served.
9. Mass_per_area: mass of flight system candidate per coverage of a geographical area unit, equivalent to total mass/area_served.
10. Mass: total mass of flight system candidate.
11. Aeroshellmass: mass of the aeroshell of the envelope.
12. MoverF:
13. Alt: maximum altitude reached by the flight system.
14. Size: total size of the balloon as determined by the flight system’s widest dimension.
15. Payload: weight of flight system’s payload.
16. Steering: the daily altitude range that a flight system must be able to navigate.
17. LTE_hours: total LTE hours the flight system must be able to serve.
18. Fleet_cost: total cost of a fleet of the given flight system candidate.
19. Fleet_cost_day:

Within the sizing tool, design objectives are functions defined within **optimize/objective.py**, where the inputs are the design variables corresponding to the flight system, and the list of DesignModules containing the corresponding subsystem surrogate models and constraints. Design objectives are evaluated by optimization solvers. We discuss the two solvers currently implemented - **SLSQP and Vizier** - in detail in the optimization algorithm section.

Optimization Algorithms

In this section we go over the mathematical formulation of single objective and multi-objective MDO, followed by descriptions of the optimization solvers used.

Single Objective MDO

For a single objective MDO, the sizing tool can be mathematically formulated as the following optimization problem.

$$\begin{aligned} \max_{x_1,\ldots,x_N \in \mathbb{R}} \; & f^k(x_1, \ldots x_N) \\ \text{s.t. } & \ell_i \leq x_i \leq u_i, \quad \forall\, i \in \{1, \ldots, N\} \\ & g_j(x_1, \ldots, x_N) \leq 0, \quad \forall\, j \in \{1, \ldots, M\} \end{aligned} \tag{1}$$

Where $x_1, \ldots, x_N$ are the **input design variables** within the MDO, ℓ_i and u_i are the lower and upper bounds for each design variable, respectively, g_j are all the **design constraints** imposed on the MDO, and f^k is the **design objective** being optimized. In general, the sizing tool has about 100 input design variables and 50 design constraints(TODO check what exactly are the number of design variables used).
The output of (1) is $x^\star$ - the optimal flight system candidate satisfying

$$\begin{aligned} x^\star = \operatorname*{argmax}_{x_1,\ldots,x_N \in \mathbb{R}} \; & f^k(x_1, \ldots x_N) \\ \text{s.t. } & \ell_i \leq x_i \leq u_i, \quad \forall\, i \in \{1, \ldots, N\} \\ & g_j(x_1, \ldots, x_N) \leq 0, \quad \forall\, j \in \{1, \ldots, M\} \end{aligned} \tag{2}$$

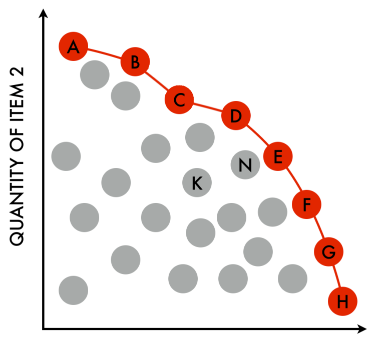
In single objective MDO, the objective is usually highly non-smooth and non-convex, therefore, multiple local optimals exist, and it is very difficult to determine the global optimal solution. The sizing tool is often run multiple times over different sections of the design space to determine a set of local optimal solutions, and the results are compared to achieve a somewhat 'global' solution.

Multi-objective MDO

In multi-objective MDO, the goal is to collect a set of flight system candidates that are at the pareto front, $\mathcal{P}$, of a set of design objectives, $\left\{f^1, \ldots, f^K\right\}$, for a design space domain, $\mathcal{D} = \left\{y \in \mathbb{R}^N \mid \ell_i \leq y_i \leq u_i\right\}$. The pareto front of a set of design objectives is defined by

$$\mathcal{P} = \left\{x \in \mathcal{D} \mid \exists\; k, \;\text{s.t. } f^k(x) > f^k(y), \;\forall\; y \in \mathcal{D}\right\}$$

I.e., a flight system is pareto optimal if it is better than every other flight system candidate in at least one of the objectives f^ks.



For sizing purposes within Loon, determining the set of pareto optimal flight system candidates allows Loon to better study the trade-offs between different objectives such as availability and cost.

Sequential Least Squares Programming (SLSQP)

SLSQP is an iterative gradient-based algorithm that finds locally optimal solutions of a single objective MDO. The sizing tool uses the third party tool scipy.minimize(SLSQP) implementation of SLSQP. Details of its implementation can be found here.

Limitations

Based on the SLSQP documentation, the algorithm is suitable for N<= 200, where N is the number of input design variables, and M' <= N, where M' is the number of active constraints at the optimal flight system. A constraint g_j is active if it evaluates to 0.

Overview

As a high level overview, SLSQP iteratively improves its solution x at step k via

$$x^{k+1} = x^k + \alpha^k d^k$$

Where $\alpha^k \in \mathbb{R}_+$ is the step size at iteration k, and $d^k \in \mathbb{R}^N$ is the search direction. The **step size** α^k is found through a modification to Powell's line search method, described here.
The search direction d^k is found by solving a quadratic approximation to the Lagrangian of (1).
Given an optimization problem (1), it's Lagraingian is given by

$$L(x, \lambda) = f^k(x) + \sum_{j=1}^{M} \lambda_j g_j(x) \qquad (4)$$

The second input λ^k is the Lagrange multiplier and can be found by solving the dual problem corresponding to x^k. The quadratic program solved to determine the search direction is given by

$$d^k = \underset{d_1,\ldots,d_N \in \mathbb{R}}{\operatorname{argmax}} \frac{1}{2} d^\top B^k d + \nabla f(x^k) d$$
$$\text{s.t. } \nabla g_j(x^k) d + g_j(x^k) = 0, \ \forall\ j = 1, \ldots, M \qquad (5)$$

Where $B^k = \nabla_{xx} L(x^k, \lambda^k) \in \mathbb{R}^{N \times N}$ is the Hessian of the Lagrangian.

Vizier

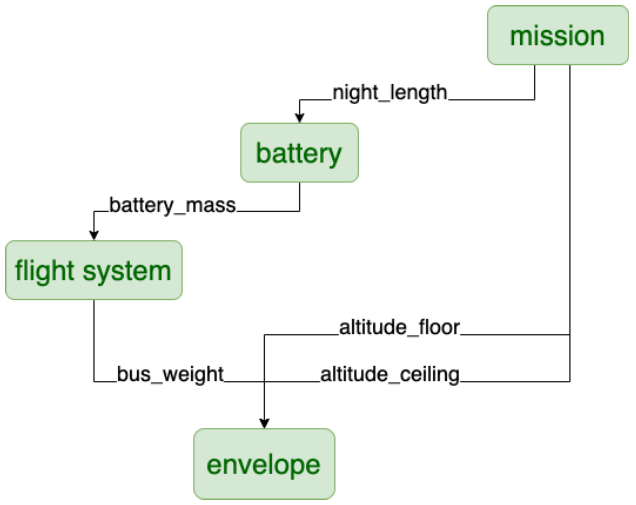
Blackbox optimization also aims to solve (1), but assumes that the gradient and Hessians of functions f and g are **not** accessible.

Vizier is a blackbox optimization solver built by Google and running on Google Cloud Platform. See here for further documentation on vizier.

Sizing Subsystems

The sizing tool contains surrogate models for the following subsystems:

1. Flight System
2. Envelope
3. ACS
4. Aerodynamics
5. Envelope
6. Avionics
7. Battery
8. Solar
9. Cost
10. Mission
11. Stability & Control
12. Downconnect
13. LTE
14. Seahorse
15. Recovery



Each subsystem is connected with one another via shared design variables. For example, design variables calculated by the mission module affect other modules, such as battery and envelope.

Envelope

The Envelope module contains the calculations for the envelope system such as leak rates, lifts, and pressure margins.

Volume

Volume is calculated differently for different envelopes. For the Seahawk module, it's calculated via the Atlas report. For L = length and e = elongation,
Volume = 4 \pi \frac{L^3}{e^2}

Mass

Envelope mass = fin mass + shell mass + ballonet mass
Where all masses are linear functions of their respective surface areas

Fin mass = fin area * fin specific weight
Shell mass = shell area * shell weighting factor * shell specific weight
Ballonet mass = ballonet area * ballonet weighting factor * ballonet specific weight
The surface areas of fin and shell are estimated.
Fin area= fin specific area * volume^{2\3}
Shell area= (0.4616 * elongation + 4.306) * volume^{2\3}
The surface area of the ballonet is all of shell area

\pi r^2 * 2 - ballonets x 2

Membrane holding the ballonet in place (doubled thickness of ballonet) = 2 \pi r^2

- Ballonet won’t slosh when the balloon tilts.

Ballonet area = shell area + 4 \pi r^2

Leak rate

Leak rate is interpolated based on shell permeability data (function of temperature), where permeability has units barrer.
Then converted from barrer to moles per day:
leak_{material} = (barrer * surface area / material thickness * stratosphere pressure * 3600 * 1e-11 * hydrogen density / hydrogen molecular weight)
Final leak rate = leak_shell + leak_ballonet + miscellaneous leak

ZP margin

Zero pressure margin verifies that the balloon does not zero pressure at its lowest pressure point.
N = Total moles of gas = ballast_gas + fill_gas + ballast_weight_equivalent - leak
T_super_min = interpolated min super temperature from data profile at altitude ceiling
P_super_min = nR(T_atm_vec + T_super_min) / V - P_atm
Zp_margin = interpolated(altitude, P_super_min) at (ceiling + floor)/2
Constraint = zp_margin - zp_threshold

Burst margin

Burst margin verifies that the maximum envelope pressure does not make the envelope burst.
T_super_max = interpolated max super temperature from data profile at altitude at noon
N = Total moles of gas (initial before leak) = ballast_gas + fill_gas
P_super_max = nR(T_atm_vec + T_super_max)/V - P_atm
Burst_margin = interpolate(altitude, P_super_max) at altitude at noon
Constraint = max_burst_margin_threshold - burst_margin

Lift Delta

Verifies that the balloon has enough lift force for the envelope.
Net weight = envelope_weight + bus+weight + ballast * 1.05

Gross lift = (volume* density at altitude ceiling * (1 - (lift gas molecular weight) / m_air))
Constraint = Gross_lift - net weight = (100 * (gross lift / net weight - 1) - 0) / (net weight)

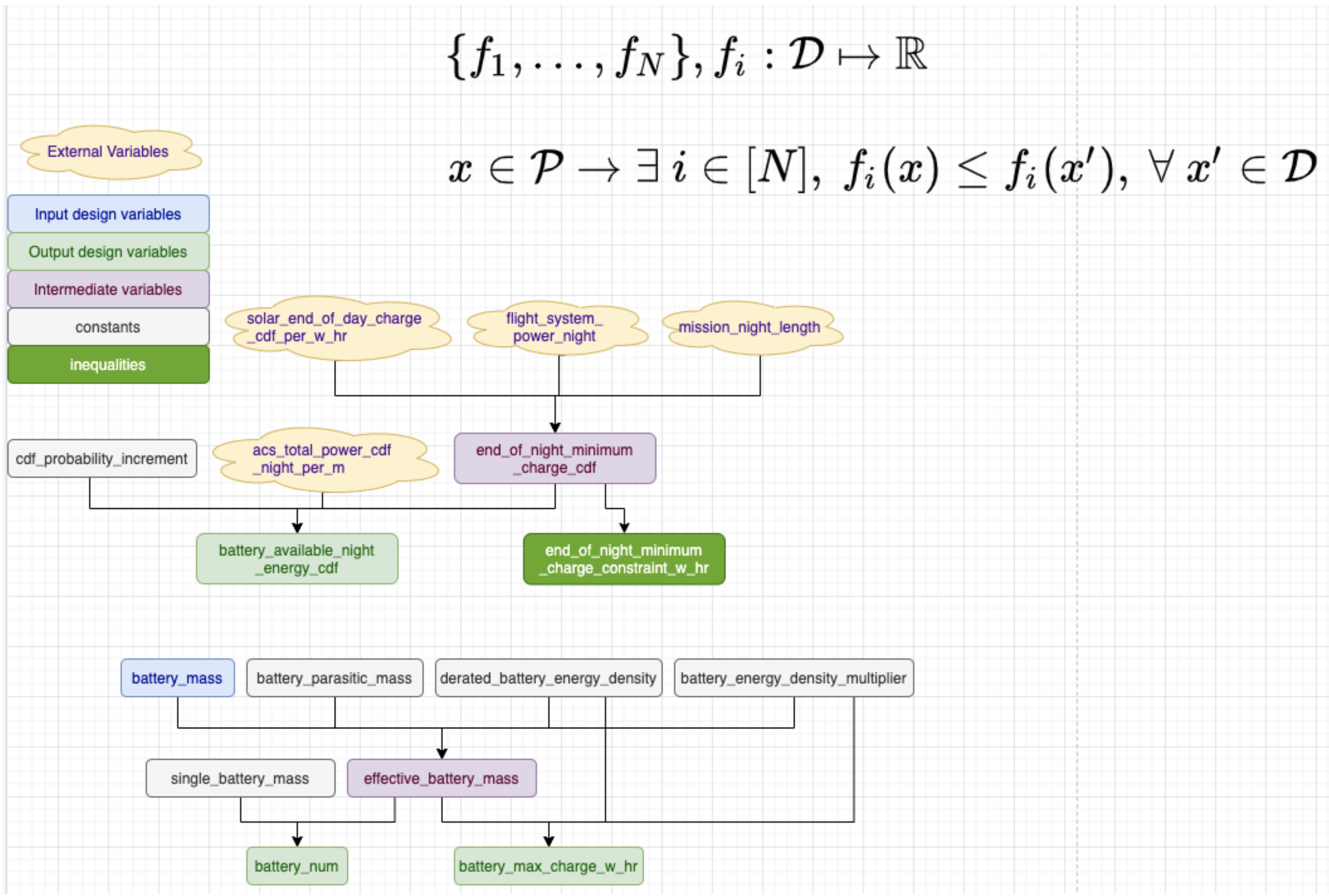
Free Lift

Constraint = 25 - 100 * (gross lift / net weight - 1)

Free Lift Min

Constraint = (100 * (gross lift / net weight - 1) - free_lift_min_thershold)

Battery



$\{f_1, \ldots, f_N\}, f_i : \mathcal{D} \mapsto \mathbb{R}$
$x \in \mathcal{P} \to \exists\, i \in [N],\ f_i(x) \le f_i(x'),\ \forall\, x' \in \mathcal{D}$
External Variables
Input design variables
Output design variables
Intermediate variables
constants
inequalities
solar_end_of_day_charge_cdf_per_w_hr
flight_system_power_night
mission_night_length
cdf_probability_increment
acs_total_power_cdf_night_per_m
end_of_night_minimum_charge_cdf
battery_available_night_energy_cdf
end_of_night_minimum_charge_constraint_w_hr
battery_mass
battery_parasitic_mass
derated_battery_energy_density
battery_energy_density_multiplier
single_battery_mass
effective_battery_mass
battery_num
battery_max_charge_w_hr

Figure 1: Battery System Diagram

The battery module calculates the number of batteries needed based on the ACS power consumptions, solar panel charging rates, and the target mission night power.

The optimizer can choose the size of the battery mass as **battery_mass_kg**. Given a chosen battery mass, the **effective battery mass [kg]** available for usage is given by
Effective_battery_mass = battery_mass_kg / (1 + parasitic_mass + derated_battery_energy_density * energy_density_multiplier)

Where parasitic_mass [kg/W-hr] includes:

1) The percentage of battery mass whose power goes towards maintaining insulation,
2) The non-power, structural portion of the battery.

derated_battery_energy_density [kg/W-hr] includes:

1) Derating batteries refers to operating the battery at less than full capacity to prolong its life. Energy density refers to the amount of energy [W-hr] stored in a unit of the battery mass.
2) 7% of reserved battery capacity for vehicle's end of life.

Energy_density_multiplier (Unit-less): multiplier used to study the effects of improving battery life (default:keep at 1).

Number of batteries

Given an effective_battery_mass (see above), **battery_num** is calculated as

Battery_num = effective_battery_mass / single_battery_mass

Where a single_battery_mass [kg] is the mass of a single battery. **Battery_num** corresponds to the quantity of batteries carried onboard the balloon.

Maximum battery charge

The maximum battery charge corresponds to the total maximum charge carried by all the batteries and is given by
Battery_max_charge_w_hr = effective_battery_mass * derated_battery_energy_density * energy_density_multiplier
See effective battery mass calculation for these variable definitions

Available energy probability distribution at night

Taking into account the power used by ACS, the available_night_energy_cdf [Whr] computes the cumulative distribution function of available battery power at night after taking ACS power usage and solar panel charging into consideration.

At the end of the day, the total amount of charge available is given by solar_end_of_day_charge_cdf [Whr] - the probability distribution of the battery charge remaining at the end of the day. Taking out the balloon's night time operation requirements, the **energy at the end of the night** [Whr] as a probability distribution is given by
End_of_night_minimum_charge_cdf = solar_end_of_day_charge_cdf - flight_system_power_night * mission_night_length

Where flight_system_power_night [W] is the nominal power draw per night time operations, and mission_night_length [hr] is the length of nights.

The other major energy usage source is ACS, so the amount of available energy during the night as a probability distribution is given by
Available_night_energy_cdf = end_of_night_minimum_charge_cdf - acs_total_power_cdf_night

Where the subtraction is a random variable operation, that corresponds to a convolution.

The available_night_energy_cdf is constrained to be non-negative by the optimizer cia **end_of_night_minimum_charge_constraint_whr**.

ACS

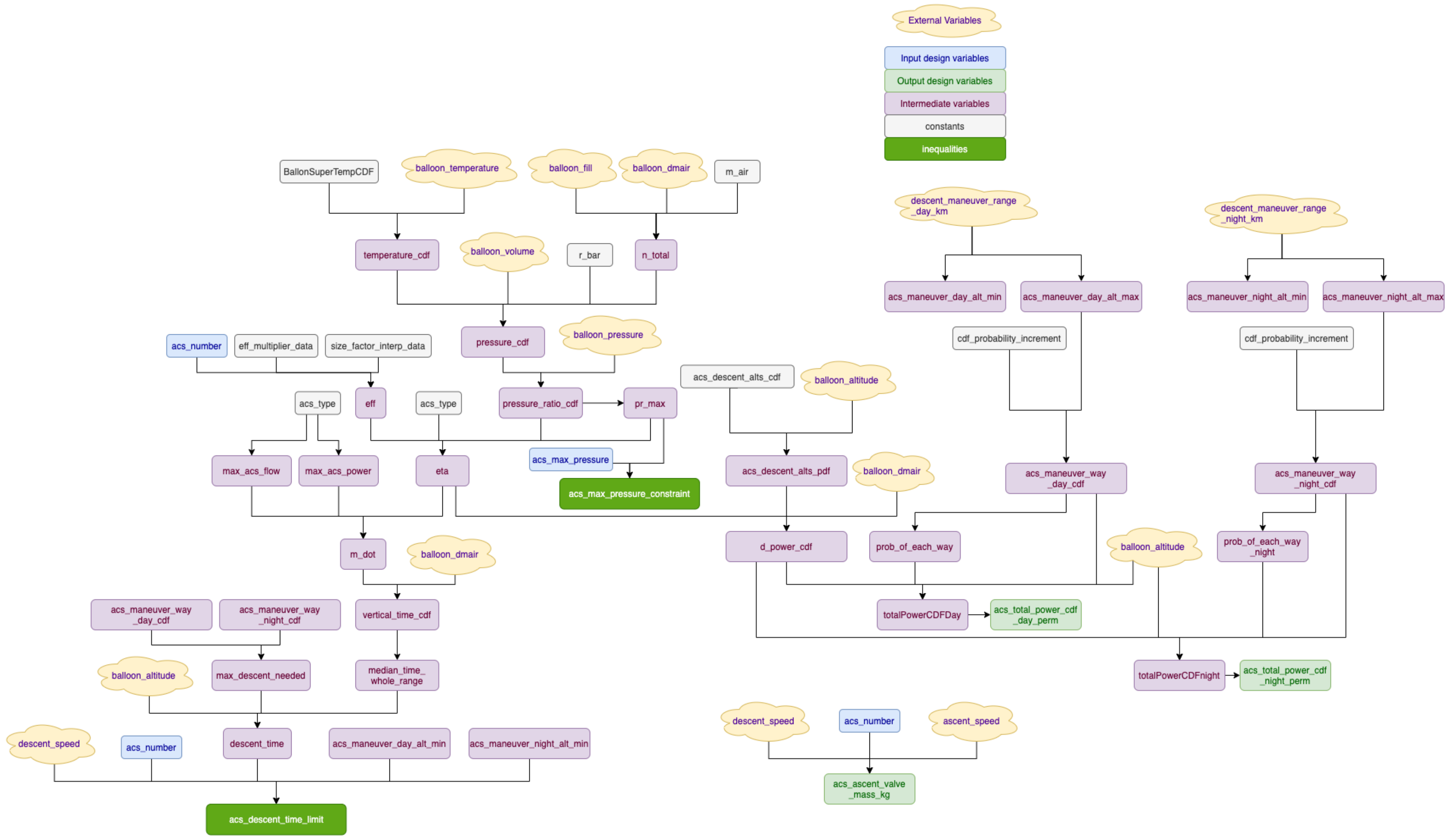

External Variables
Input design variables
Output design variables
Intermediate variables
constants
inequalities
BallonSuperTempCDF
balloon_temperature
balloon_fill
balloon_dmair
m_air
temperature_cdf
balloon_volume
r_bar
n_total
pressure_cdf
balloon_pressure
acs_number
eff_multiplier_data
size_factor_interp_data
acs_type
eff
acs_type
pressure_ratio_cdf
pr_max
acs_descent_alts_cdf
balloon_altitude
max_acs_flow
max_acs_power
eta
acs_max_pressure
acs_max_pressure_constraint
acs_descent_alts_pdf
balloon_dmair
m_dot
balloon_dmair
acs_maneuver_way_day_cdf
acs_maneuver_way_night_cdf
vertical_time_cdf
balloon_altitude
max_descent_needed
median_time_whole_range
descent_speed
acs_number
descent_time
acs_maneuver_day_alt_min
acs_maneuver_night_alt_min
acs_descent_time_limit
descent_maneuver_range_day_km
descent_maneuver_range_night_km
acs_maneuver_day_alt_min
acs_maneuver_day_alt_max
acs_maneuver_night_alt_min
acs_maneuver_night_alt_max
cdf_probability_increment
cdf_probability_increment
acs_maneuver_way_day_cdf
acs_maneuver_way_night_cdf
d_power_cdf
prob_of_each_way
balloon_altitude
prob_of_each_way_night
totalPowerCDFDay
acs_total_power_cdf_day_perm
totalPowerCDFnight
acs_total_power_cdf_night_perm
descent_speed
acs_number
ascent_speed
acs_ascent_valve_mass_kg

Figure 2: ACS System Diagram

The ACS module models the Altitude Control System and computes the pressure and power required to keep the ballonet at target pressures. It computes the power usage of ACS as a probability distribution for both daytime and night time. It also computes the time limit for the vehicle to descend a designated maneuver range.

Total power CDF (day/night)

The total power cdfs are computed as the total amount of power required to navigate the entire altitude maneuver range. To compute the power required to go to minimum altitude from a given altitude, we compute the **pressure ratio** at each altitude using the ideal gas law as

Pressure_ratio = (temperature_atm + super_temperature) *r_bar * n_total/volume/pressure_atm

Where at a given altitude, temperature_atm is the atmospheric temperature, super_temerature is the temperature difference between the balloon and the atmospheric temperature, R_bar is the universal gas constant, n_total is the total moles of gas in the balloon, volume is the volume of the envelope, and pressure_atm is the atmospheric pressure.
These pressure ratios determine the **efficiency [g/(W hr)]** of ACS. This is an interpolated value based on existing data.

Efficiency = (pressure_ratio - pressure_ratio_min)/(pressure_ratio_max - pressure_ratio_min) * (acs_flow_max - acs_flow_min) + acs_flow_min

Where pressure_ratio_max/min and acs_flow_max/min is from data(2 points) and acs_flow_max/min scales linearly with the number of ACS units.

Efficiency is then protected from negative efficiency by replacing low efficiency values with 0.01:
Efficiency = max(efficiency, 0.1)

The efficiency is used to derive the **power** at each altitude as
Power = delta_delta_air /efficiency * (altitude - altitude_min)/(altitude_max - altitude_min)

The **possible power** at each altitude is given by

Possible_power = power * prob_each_way*maneuver_size

Then the total power is the sum of all possible powers at each altitude.

Where **delta_delta_air** [kg] is the addition in ballonet air from changing a unit altitude.

Descent time

The descent time is computed as the following. First the time it takes to descend at each altitude is given by the mass_flow_rate. The ACS **mass flow rate** at each altitude as:
Mass_flow_rate = min(efficiency * acs_power_max, acs_flow_max)

Mass flow rate is then used to compute the amount of time it takes to pass the current delta altitude as:
Vertical_time = delta_delta_air /flow_rate * 1000
The descent time is then given by
Descent_time = vertical_time / 3600 * max_descent_needed / (altitude_max - altitude_min)

This value is constrained to be equal or less than the time it takes to descent the whole maneuver range as:
(maneuver_range_day + maneuver_range_night)/descent_speed >= descent_time / acs_number

Where maneuver_range_day/maneuver_range_night/descent_speed are all values set by the optimizer.