Short term plan for Systems

Devansh Agrawal
ICLR Systems

October 26, 2019

1 Summary

There are three major tasks that need to be completed in about the next week or two, and pretty much need to happen in parallel.

1. Define top level requirements in Valispace
2. Preliminary and semi-detailed sizing
3. Standard interface definition

This document provides some guidance on why these are critical next steps, and on how we will go about doing these.

2 Define top level requirements

2.1 Why

1. We can't let any high level requirements slip through the cracks of communication
2. All design decisions should be traceable to to the top level requirements

2.2 How

There are a few types of top level requirements that need to be specified, and the first sub-level requirements can be specified from there. Every requirement must have a:

- *owner* which is the person in charge of ensuring the requirement is met.
- *description* which explains the requirements in words
- *success criteria* which is a specific rule that can be checked to see if the requirement is met (usually numeric)

The verification method for the requirement is needed if relevant.
The top level requirements are derived from the competition:

1. COMP-SAFE-xxx: Competition safety requirements. These also include requirements on which tests need to be performed

2. COMP-PERF-xxx: Competition performance requirements. These include all the rocket performance requirements, including things like the launch rail velocity requirement or stability requirements.

3. COMP-TIME-xxx: Competition timeline requirements. These indicate all the deadlines from the compeititon, internal deadlines will be labelled differently.

The xxx refers to the requirement ID number.

From these competition requirements, all internal requirements should be derived.

We will define these based on which system they are relevant to, and the breakdown of subsystems is defined in the next section.

From here, the responsible engineers list should also be defined, which includes exact specification of which person is incharge of which components and which interfaces, to (again) prevent details from being missed in communication.

3 Preliminary and semi-detailed sizing

3.1 Why

Most teams have not started a detailed component design as they are unaware of parameters from other teams. The initial sizing doc provides a decent estimate on the parameters, but now detailed sizing for each component must be carried out.

Seeing that we don't have much time, this is roughly a mashup of both the preliminary and detailed sizing of Sporadic Impulse.

3.2 How

1. Define most critical parameters and their relationships on valispace based on the initial sizing. At the same time, define these in openrocket

2. Perform a more detailed baseline estimation of the size of each component, working with the team members to produce accurate bounds on system parameters.

3. Continuously feed back into valispace such that the design can stay upto date.

4. Ensure all top level requirements are still being met.

5. Perform more detailed sizing of the components.

Initially, we can start by spec-ing a cardboard body tube and nothing too fancy in general. All the embellishments, like generative design or special manufactured components can either be ignored at this stage, or estimated as some percentage reduction in mass (or whichever relevant parameter you have). Please don't forget which assumptions you have made, and to go back and correct them later.

4 Standard Interfaces

4.1 Why

To reduce design time and communication, we must define the basic standard interfaces between teams. Teams are free to modify them later but must agree upon it. Essentially

we want to avoid scenarios where we are trying to fit an M4 screw into an M5 hole that is 0.2 mm too far to the left.

4.2 How

The key interfaces we need to define are: (1) mechanical, (2) electrical

1. Understand the relevant load paths, and figure out where the mechanical connections should be
2. Design it to allow for easy changes down the line - think lego rather than 3d printing - it should require minimal effort to change the rocket later and 3d printing a new mount isnt really a 'quick' solution - it requires about a day to do, compared to minutes if you can just unscrew something and screw it in somewhere else.
3. reduce the number of different types of screws or mountings there are
4. dont forget about electrical paths

5 Additional Stuff: Fusion-Valispace-OpenRocket-API

5.1 Why

Soon, we will be straddling at least three different environments - Valispace (where we store the parameters of the rocket), Fusion (where the detailed design of the rocket is) and OpenRocket (where much of the simulation and verification tools are.

These need to be in sync, and the best way to do this is probably using an API where we can run a script, and it would sync across the three platforms.

Specifically the interactions we need are (in order of priority):

1. *Valispace* $\rightarrow$ *Fusion*

   Valispace has the dimensions and parameters of some components that are subject to change, we need these parts to get updated in Fusion.

2. *Fusion* $\rightarrow$ *Valispace*

   Estimating the mass, centre of masses, moments of inertia and placement of components is tricky/too much additional work within Valispace. As such, it would be ideal if we can export these parameters from Fusion into Valispace. This requires the materials to be accurately described and correctly selected within Fusion. It probably also requires everyone to agree on a standard coordinate system, which you should decide and stick to. There is no real reason to change this afterwards, and every subcomponent designed in Fusion should stick to this convention. Probably the easiest is to follow Fusions built in config, where z points up the rocket, x points to the right when looking at the rocket, ie. the $x - z$ plane defines the front face of the rocket (see the view cube below). Also important to define (for Valispace integration) is the reference location of each part, so the center of mass is measured relative to something, but this needs some more thought.

Figure 1: Standard orientation in Fusion

3. *OpenRocket → Valispace*

   OpenRocket should provide decent estimates on drag coefficient (need to be careful about the reference area used) the altitude reached and other such simulation results. Perhaps a small closed design loop can be created to select the boosters needed, but I think manual methods here will be sufficient and quite useful for understanding our rocket intuitively.

4. *Fusion and Valispace → OpenRocket*

   Using the parameters from both Fusion and Valispace, we somehow need to find the appropriate open rocket representation to be able to run the simulations. This means we need accurate masses in the right places, the right motor thrust curve. We should then be able to run the simulations. As far as I can tell, it will be tricky to run this simulation automagically unless we create a OpenRocket plugin (which isnt impossible).

The figure below summarises this.

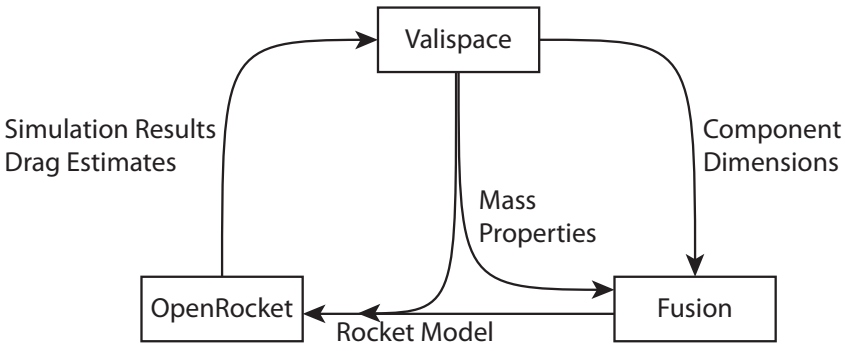

Figure 2: Basic information flow across our three main platforms

5.2 How

Fusion and Valispace both expose an API to input and export parameters, in a fairly (but not completely) trivial manner[1]. OpenRocket on the other hand does not, and the default `*.ork` file is not human-readable. The rocket can however be saved as a rocksim file, `*.rkt` file format, which stores the rocket and all its parameters as a XML format,

[1]Valispace has very poor filtering, and project hierarchy, Fusion's API must be run within their scripting menu

explained in the rocksim documentation[2]. Thus, in theory, we should be able to read parameters and simulation results, and the write straight to the file to see the result. The current best version of the rocket can be stored directly in Github, where the version tracking is done for you.

Some automated way to run the simulations would be very helpful, especially for dispersion analysis, where we can modify each component by $\sim 5\%$ and see how the apogee changes. We may even be able to use this interface to overwrite component mass and cog estimates, straight from valispace (which got it from fusion).

All development of code should be done in Github. In the specific case of the Fusion API which must be stored in some weird directory of your computer, I would recommend linking that file to the Github tracker node.

The exact implementation is to be determined based on what is feasible, but ideally there should be a function/bash script/button in Fusion that I can hit and everything gets updated.

6 Breakdown of System

To avoid confusion, the break down of the entire system is specified here. This is distinct from the team layout,

At the top level, there is either the `Rocket` or `Ground Support`, which contain:

1. Rocket
    (a) Payload
    (b) Avionics
    (c) Airframe
    (d) Recovery
    (e) Propulsion
    (f) Control Algorithm Software[3].
2. Ground Support
    (a) Storage and refuelling system
    (b) Telemetry and tracking equipment
    (c) Test equipment (including any relevant test stands and equipment needed for competition)
    (d) Transportation equipment (eg. boxes to store the rocket and its components, etc)
    (e) Tools and toolboxes
    (f) etc.

The further breakdown is not specified here. In Valispace, we shall keep the `rocket` object not as a parent, but as a separate component. This is simply for convenience, as we don't want to have to refer to valis as `rocket.avionics.mass` but simply as `avionics.mass`. Ground support equipment should still be kept separate.

[2]here is one version I found, there may be more: `https://www.apogeerockets.com/downloads/PDFs/Rocksim.pdf`

[3]need some thinking on whether it is appropriate here