openRocket-modeFrontier Integration

Integration of openRocket dynamic models and modeFrontier MDO software using the EasyDriver node

Nathanael Jenkins
ICL Rocketry Altitude Record Team
Aerodynamics & Simulations Sub-Team

Created: 24 Mar 2023
Last Edited: 27 Mar 2023

Revision 2

Introduction

ICL Rocketry uses openRocket for dynamic simulations of rocket designs to validate performance and stability targets. Optimising rocket performance is critical, especially for the Altitude Record Team's record-breaking work.

Although openRocket provides some optimisation capabilities, they are not sufficient for the advanced optimisation needed to break altitude records. ModeFrontier MDO software is well-suited for multivariable problems and makes it easy to integrate other software.

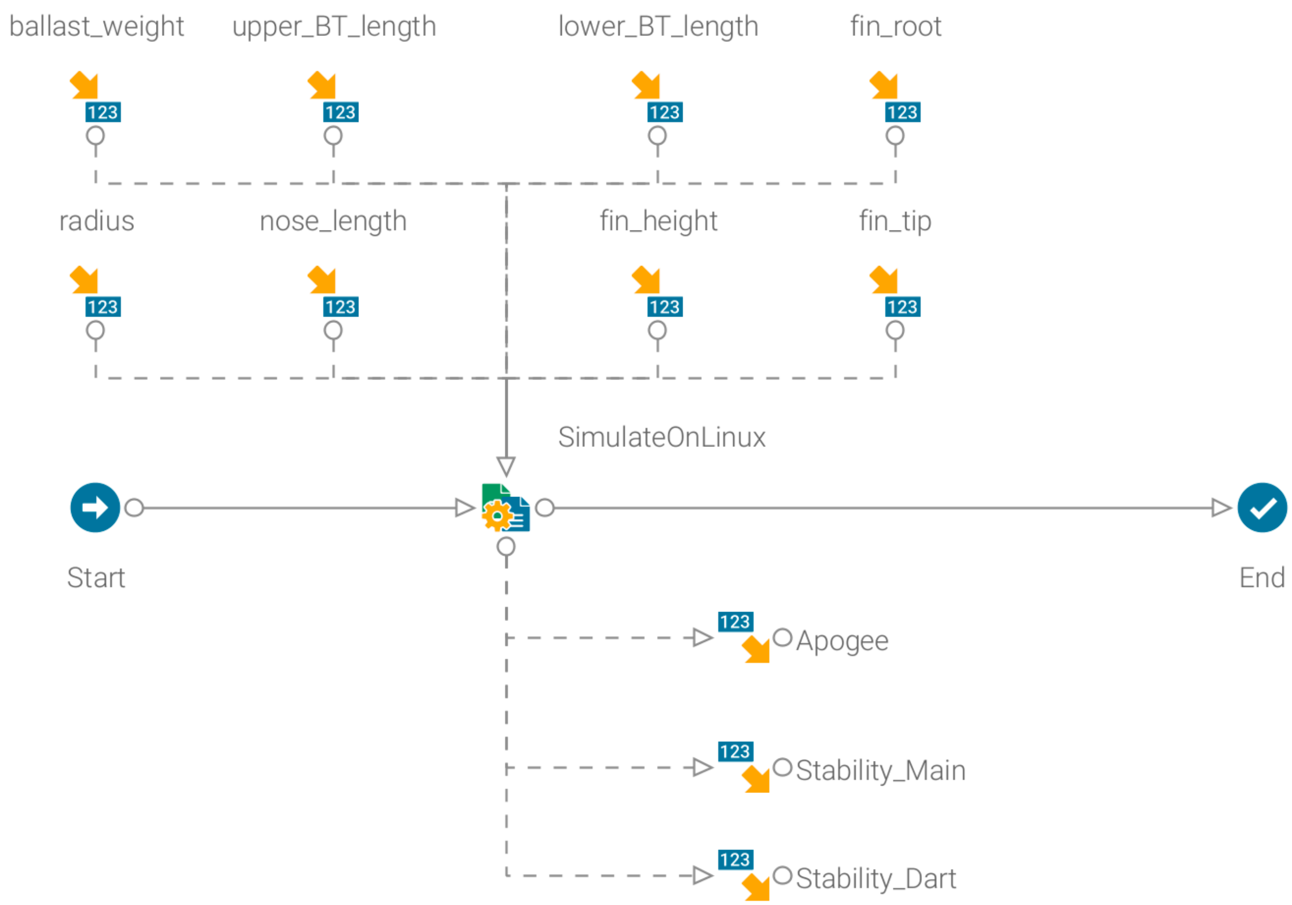
An open-source openRocket API has been developed that interacts with the Java program using a Python script for basic simulation capabilities. This makes it easy to run openRocket simulations using the modeFrontier Python node. However, this method does not provide the ability to change openRocket design parameters, which are stored in a *.ork file in XML format. Additionally, launching the openRocket API on a local machine involves loading several databases in parallel, making local execution of concurrent openRocket instances computationally intensive. Preventing parallelism from being used for this is surprisingly difficult.

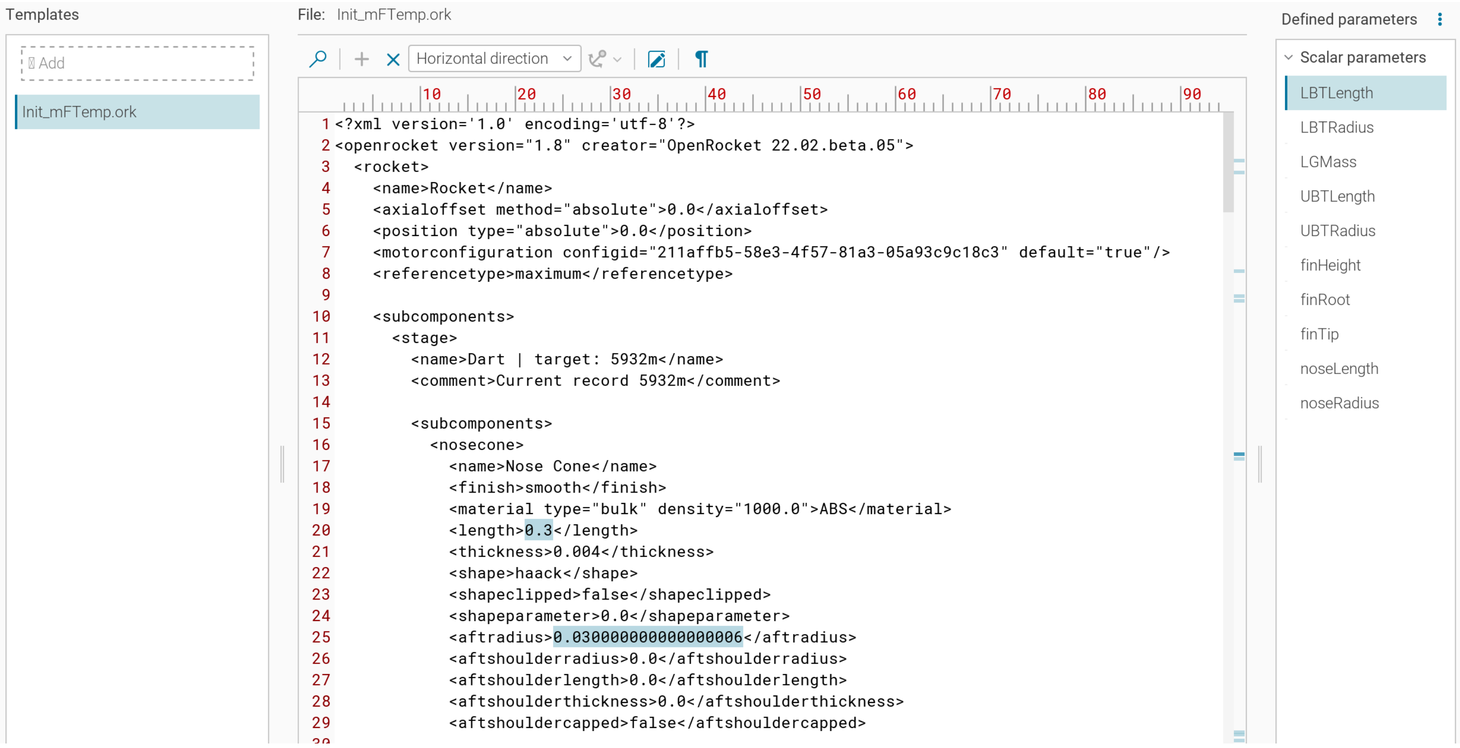
A better approach to integration has been identified using the EasyDriver node to execute simulations on a remote high-performance server. The node provides much greater scope for changing design parameters, and the use of bash scripting allows the user to better limit the number of cores each instance of openRocket may use.

Input Parameters

openRocket saves files in the *.ork format, which is essentially a compressed XML. By unzipping the *.ork file, it is possible to load the uncompressed *.ork file into the 'input template' on the EasyDriver node. Any default rocket design can be used, provided it includes all of the components the user intends to optimise.

The EasyDriver 'input template' makes it easy to highlight parameter values to replace. These can be defined for each input variable and linked to input nodes using the 'introspection' tool. This generates a unique *.ork file in the 'proc' directory of each design evaluation.





Remote Execution

openRocket simulations are conducted using the generated *.ork file through a simple Python script which interacts with the openRocket API. This script is provided below, with explanatory comments where helpful.

```python
import os
import numpy as np

# Load openRocket API
import orhelper
from orhelper import FlightDataType

os.environ['JAVA_HOME'] = '/usr/lib/jvm/java-11-openjdk-amd64'
os.environ['CLASSPATH'] = '<user home directory>/openRocket/OpenRocket-22.02.jar'

with orhelper.OpenRocketInstance('<user home directory>/openRocket/OpenRocket-22.02.jar', 'OFF') as instance:
    orh = orhelper.Helper(instance)

    # Load document, run simulation and get data and events
    doc = orh.load_doc('Init_mFTemp.ork')    # Note this loads from process working directory
    sim = doc.getSimulation(0)

    orh.run_simulation(sim)      # Run simulation
    data = orh.get_timeseries(sim, [FlightDataType.TYPE_ALTITUDE, FlightDataType.TYPE_THRUST_FORCE,
        FlightDataType.TYPE_STABILITY])

    apogee = max(data[FlightDataType.TYPE_ALTITUDE])

    stability = data[FlightDataType.TYPE_STABILITY].tolist()

    thrust = data[FlightDataType.TYPE_THRUST_FORCE].tolist()
    stability_burnout = stability[thrust.index(0)]

    stability = list(filter(lambda v: v==v, stability))
    stability_launch = stability[0]

    if np.isnan(stability_burnout):
        if min(stability) < 0:
            stability_burnout = min(stability)
        else:
            stability_burnout = 0

    # Write data to output file (in working directory)
    f = open("output.txt", "w")
    f.write(str(apogee))
    f.write("\n")
    f.write(str(stability_launch))
    f.write("\n")
    f.write(str(stability_burnout))
    f.close()
```

To reduce local computational workload, simulations are conducted on a remote Linux server. The command to run this from the EasyDriver node (in the 'SSH' commands section) is given below.

```
CORE=$((DESIGN_ID%80))
taskset -c $CORE python3 <user home directory>/mF_oR/mF_oR.py
```

Note that this command makes use of the DESIGN_ID environment variable set by modeFrontier, which allows the program to cycle through every CPU on the Linux server. The 'taskset' command defines which core to execute the openRocket instance on. Although not an elegant solution, this has been identified as a means to reduce computational workload on the servers. The program cycles through all of the cores in order to prevent concurrent evaluations from executing on the same core if one evaluation is delayed for some reason.

The Python file can be saved to a single directory on the server, and it does not then need copying into each working directory, reducing optimisation time. It is important to set up the EasyDriver configuration to execute by SSH (set a hostname and credentials) to ensure this is executed correctly. This implementation can run concurrently without any conflicts. Note that this will generate a large 'ESTECO' file in the specified remote working directory that can be deleted after each run.

Output Processing

The Python script shown above writes an 'output.txt' to the working directory, which is read by the EasyDriver node using similar text parsing to the input template file. These values can then be fed into output variables. The Python script can be modified to extract a much larger set of variables if required.