Open-Source Visualization of Reusable Rockets Motion: Approaching Simulink - FlightGear Co-simulation

Marco Sagliano*
German Aerospace Center

This paper shows how to approach effective visualization of the motion of reusable rockets by combining Simulink / Matlab modeling with the capabilities of FlightGear, a state-of-the-art open-source tool typically used for aircraft simulation in the gaming community. We describe the entire open-source toolchain and the steps needed for the coupling of the involved software, with detailed code provided as Appendices. Finally, We propose a concrete example, associated with the application to the motion of reusable rockets.

I. Introduction

During the last years the rapid progress of computers' Central Process Units (CPU) and the corresponding reduction of costs has brought a big development in terms of realistic simulators. The easiness of access to both machines and software has sparked the inventiveness of researchers, able to find new ways to test algorithms and numerical methods. A remarkable example was the use of the worldwide famous Rockstar's game GTA made by the machine learning research community. They could exploit what was an excellent reproduction of a urban environment containing people and other cars such that the driving agent could learn how to interact with them[1]. Other well-known commercial simulators, like Flight Simulator give the chance to experience what aircraft piloting means in detail [2], while for what regards the space environment well-known software packages were largely adopted by space enthusiasts to build their own space program [3] for both leisure and space-popularization purposes [4].

Although companies created high-end products the open-source community did not stand and stare, but big collective groups of passionate programmers joined their efforts to create several alternatives that were available to further make experiments and test new ways of using software initially born for home video-gaming. Among these tools a remarkable example is provided by FlightGear [5], a completely open-source project aiming at providing an accurate simulating environment to fly different aircrafts, described at different levels of accuracy, which can fly with or without aided guidance and control. The software does not only provide a high-quality 3-D environment, but can be coupled with physical flight dynamics engines to provide a ultra-realistic experience. An example of such model library is JSBSim [6], which provides the possibility to interact with FlightGear via XML interface. In recent years Mathworks understood the possibilities provided by FlightGear and introduced a demo simulating the landing of the HL-20 [7] to show how to use the recent visualization block implementing the interface between Simulink and Flightgear.

The aircraft GNC community soon realized that FlightGear was a powerful tool to test and visualize their algorithms for traditional aircraft [8, 9], unmanned air vehicles visualization and control [10], new urban air mobility concepts [11], and helicopters' motion visualization [12]. Moreover, given the high degree of detail that can be achieved the software was also used for cockpit-related research and development activities [13], and the implementation of control and identification algorithms [14].

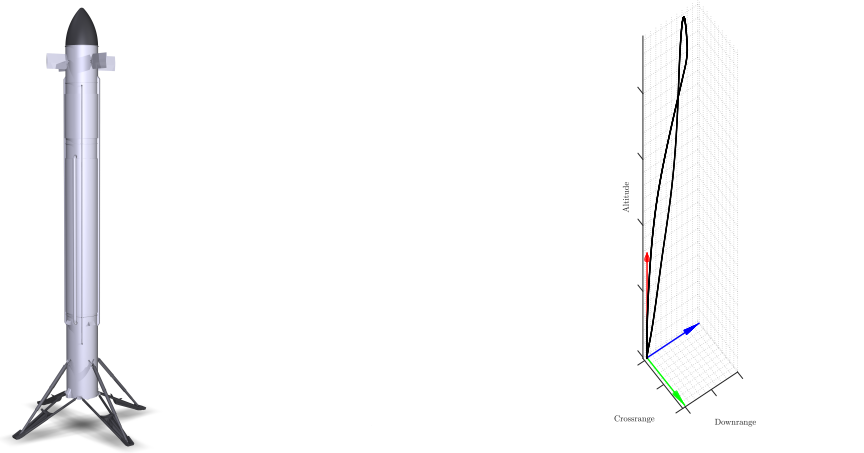
In this paper we focus on the use of FlightGear in combination with Matlab and Simulink for co-simulation and visualization of flight of reusable rockets. We will not rely on any Mathworks predefined toolbox, but rather build the overall environment for co-simulation from scratch, which implies that no further commercial software rather than plain Matlab / Simulink are needed, hoping to further boost the usage of this technology in the guidance and control and flight dynamics communities. All the steps will be illustrated, together with the corresponding open-source tools needed, including Blender for the 3-D modeling of the rocket, and the C-code for establishing the communication between Simulink and FlightGear. The paper is organized as follows: in Sec. II a reference scenario for a reusable rocket is described, while Sec. III will illustrate the preparation steps to be able to perform the co-simulation, including the CAD modeling and the involved software. A concrete example of co-simulation is proposed in Sec. IV, while some concluding remarks will be outlined in Sec. V. Finally, the code required to rebuild the proposed co-simulation environment is embedded in the Appendices.

*Research GNC Engineer, Robert-Hooke Str. 7, 28359, Bremen, Germany, Senior AIAA Member

II. Mission and Vehicle

To show how Simulink-Flightgear co-simulation can be performed we consider a CALLISTO-class rocket [15, 16], able to generate a thrust force up to 40 kN. The engine, that uses liquid propellant, is throttleable, and is mounted on a gimballed system, to be able to generate thrust-vector-control (TVC), and therefore provide pitch and yaw control during the powered phases of the mission. A set of RCS thrusters completes the attitude control system. During the aerodynamic phase the engine is shut down, and the attitude control capability is provided by a set of four steerable fins mounted on top the rocket. The defined set of actuators is therefore able to control the attitude and the position of the rocket during every phase of flight. An illustration of the vehicle used as example, that is, the CALLISTO rocket, is depicted in Fig. 1(a).

For what regards the mission profile we apply co-simulation to a Return-to-Launch-Site (RTLS) profile, visible in Fig. 1(b). For this type of scenario the rocket performs the *boostback* maneuver to invert the direction of its velocity to fly back towards the landing pad, typically very close to the launch-pad, and in this case considered to be exactly in the same spot. After the boostback maneuver the Main Engine Cut-Off (MECO) command is issued, and the vehicle performs an aerodynamic descent, where the motion is controlled through the aerodynamic forces generated by the rocket itself. When a prescribed altitude is reached the Main Engine Re-Ignition (MEIG) command is issued, and the rocket performs what is known as *pinpoint landing* [17], where attitude, position, velocity and thrust are coordinated to ensure safe conditions for the legs deployment and the subsequent touchdown, when the mission is considered completed.



(a) CALLISTO experimental vehicle

(b) Return-to-Launch-Site mission profile

Fig. 1 Mission and Vehicle overview: (a) CALLISTO rocket, (b) Reference Mission Profile.

This type of mission is one of the motivations for the work proposed here. In fact, it is extremely useful to visualize the entire motion of the rocket to verify that the behavior is consistent with the expectations in each phase of the scenario, beyond the traditional use of graphical tools, like plots and telemetry messages. Moreover, we can use it to visualize the correctness of mechanisms more inherently associated with the use of reusable rocket, like fins unfolding and legs deployment.

III. Co-simulation preparation

In this section the steps needed to connect Simulink and Flightgear are explained. First we describe the properties of the CAD model and the software required. Then, we will illustrate how to prepare its use in FlightGear.

A. CAD modeling

First, a set of CAD models is required. Very common formats for modeling the rocket include .stl, .stp, and .wrl among the others, However, FlightGear requires a specific format, with extension *.ac*, associated with the proprietary

software AC3D. To provide an open-source alternative we prefer to use Blender [18], and specifically the version 2.81. Blender does not inherently support .ac files. To overcome this problem the very flexible structure of Blender and its capability to add new functionalities comes in handy. In fact it is possible to prepare python scripts able to extend its native interface. Among these extra scripts the FlightGear community developed a utility able to provide the .ac interface needed between AC3D model and Blender [19]. Although the description of the add-on refers to Blender 2.80, it has been successfully tested and used with Blender 2.81. Figure 2 shows the rocket modeling in Blender based on studies performed for CALLISTO [20].

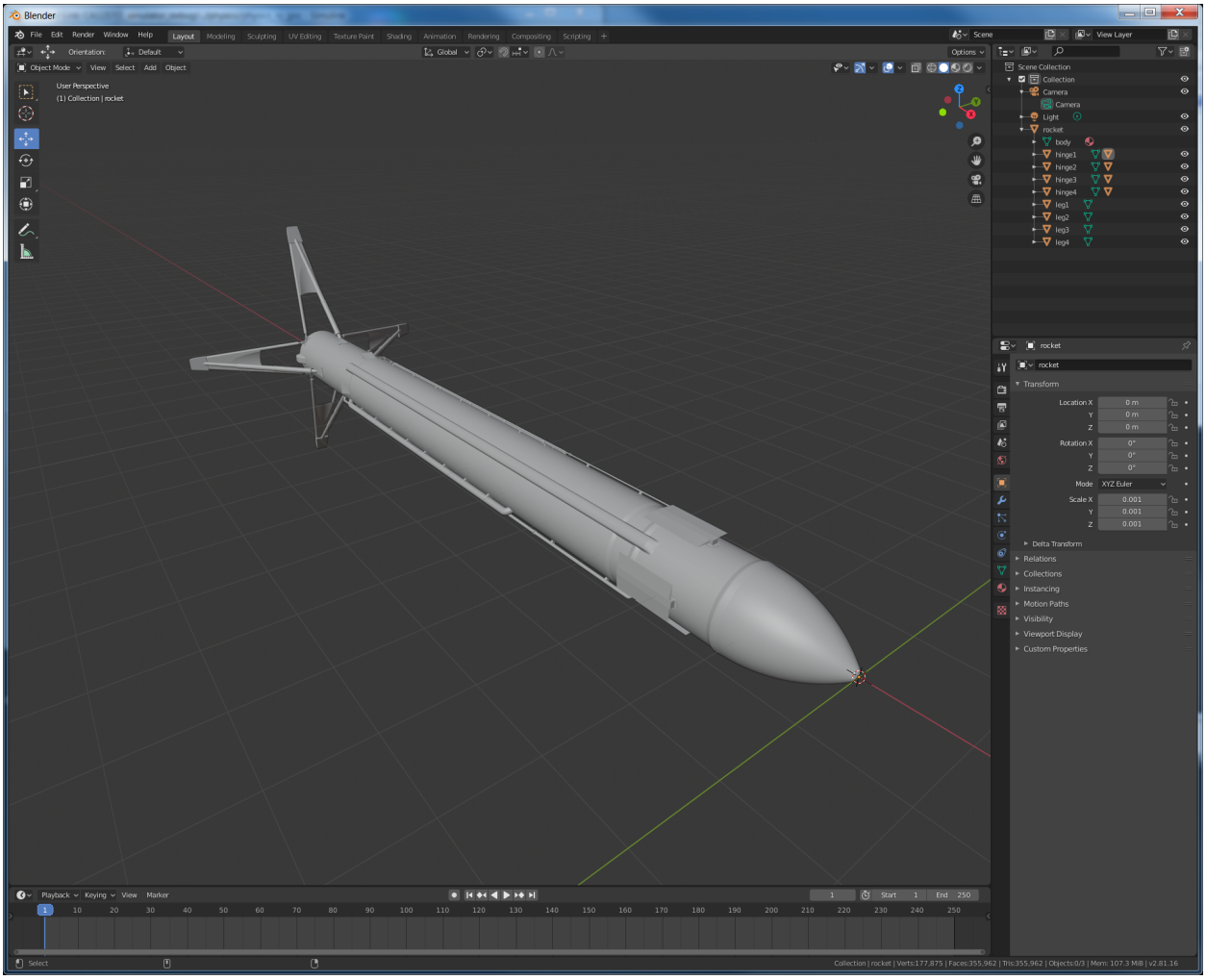


Fig. 2 Rocket modeling in Blender.

Some important points in the modeling are the moving parts, that clearly need to be defined as different objects. The relative motion can then be specified in the FlightGear internal descriptors of the model through the specification of their animation, e.g., relative rotation and translation together with the variables that control the motion. Note while using the 3-D modeling environment it is important to accurately determining the relative motion key parameters, such as the center of rotation or the axis of rotation. This type of information will be used in the next phase, when the model will be imported in FlightGear. The model, with all the related moving parts can be therefore specified according to the prescribed .ac format, and we can consider the CAD modeling phase complete.

B. Preparation of model in FlightGear

When the CAD model is ready and has been exported in .ac format we can prepare the properties of the model according to the Flightgear specifications. Following the structure indicated in Flightgear to import new aircraft [21] we create a folder (e.g., `rocket`) to be placed within the `./data` folder of the FlightGear installation path. The minimal configuration of this folder has at least the following contents:

- `Models`
- `Nasal`
- `rocket-set.xml`
- `rocket-splash.png`
- `thumbnail.png`

The folder `Models` contains the prepared .ac file (e.g., `rocket.ac`), together with an .xml file (typcailly having the same name, e.g., `rocket.xml`) defining the animations and which variables control them. For instance the motion of a leg can be modeled as the following snippet taken from `rocket.xml`.

```
<animation>
 <type>rotate</type>
 <object-name>leg1</object-name>
 <property>/legs-states/leg1-rot</property>
 <factor>1</factor>
 <center>
  <x-m>0</x-m>
  <y-m>0.5</y-m>
  <z-m>0</z-m>
 </center>
 <axis>
  <x>0</x>
  <y>0</y>
  <z>1</z>
 </axis>
</animation>
```

In this example we are imposing that the object `leg1`, controlled by the variable `leg1-rot`, which is defined under the `legs-states` group of objects will perform a rotation around the body Z-axis, and the Center of Rotation (CoR) is specified to be in the point having the following coordinates expressed in the native reference frame of Blender at the moment of exporting the *.ac* file.

$$CoR = \begin{bmatrix} 0 & 0.5 & 0 \end{bmatrix} \tag{1}$$

We will see that these properties (in this case `leg1-rot`) can be controlled through the dedicated communication protocol directly from Simulink.

In the `Nasal` folder we store the `*.nas` files, which are used to model more complex behaviors, based on the the homonym Nasal programming language. In this frame we use it to initialize the properties of the rocket, through the nasal function `init_rocket.nas`, and to place the launch platform at the launch-site location, through the built-in function `geo.put_model`.

The `rocket-set.xml` file is a high-level descriptor, in which the developer can specify the status of the model, the views to be adopted in FlightGear as well as the Nasal files and user-defined Nasal scripts. Finally, the `rocket-splash.png` and `thumbnail.jpg` are pictures associated with the rocket under development used by FlightGear during the load process and the model selection. The Nasal files coded for the rocket example are listed as Appendix IV.

IV. Co-Simulation

With the model prepared and its specifications included in the FlightGear we are ready to perform the co-simulation.

Simulink will send the 6-DOF states of the rocket to FlightGear through the model *Simulink2FG.slx*, visible in Fig. 3. The states are longitude, latitude, and altitude for what regards the position of the center of mass of the rocket, and roll, pitch and yaw angles representing its attitude. Since the model in Simulink works with International System Units

while FlightGear adopts some more intuitive but less formal conventions (e.g., latitude and longitude in degrees) some conversions are needed, and performed in the *Conversion_Block* subsystem. Finally, note that the speed of execution of the animation would be linked to the execution of the Simulink model, and therefore it might be needed to artificially decrease its speed. For this reason a *real-time pacer* block [22] has been included in the model, as visible in the bottom part of the Simulink scheme.

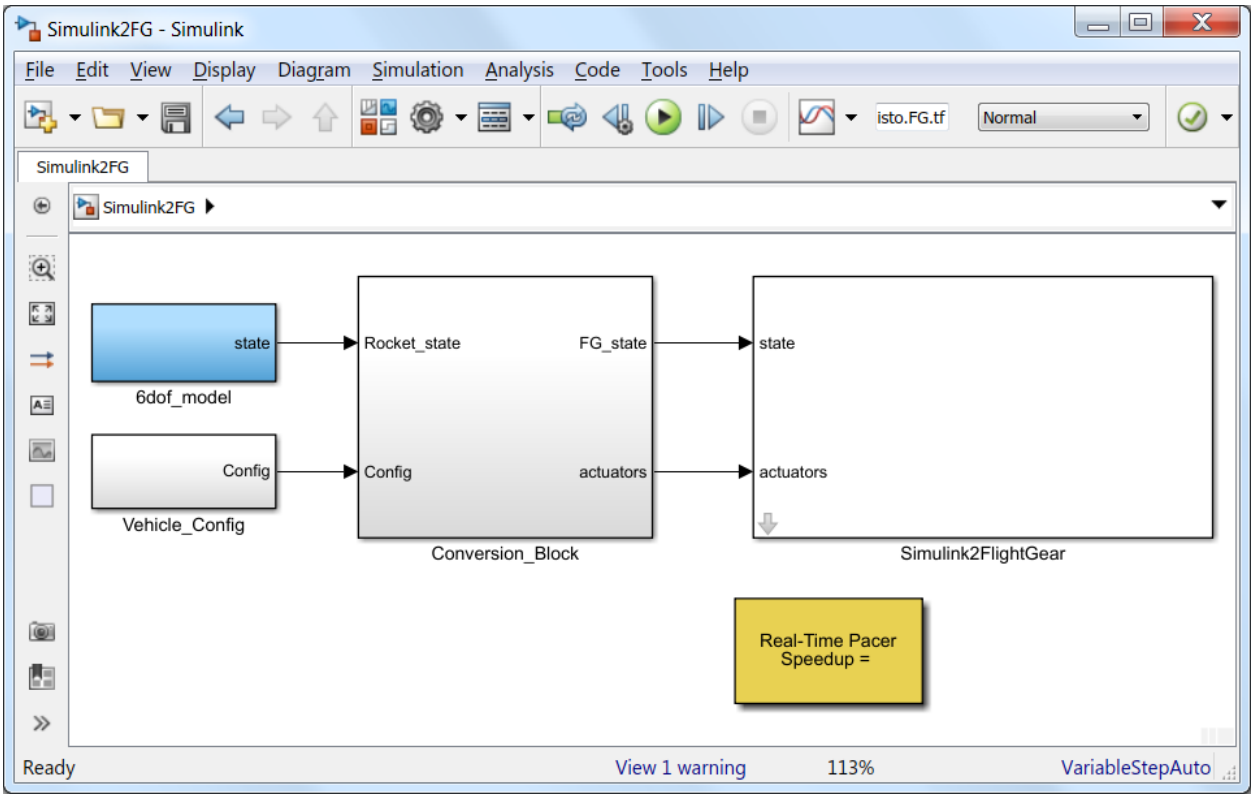


Fig. 3 Simulink-FlightGear co-simulation model.

Despite the capabilities of FlightGear to support physical dynamics engine like JBSim, the purpose here is mainly to verify the behavior of the dynamics and the guidance and control subsystems modeled in Simulink. Therefore the states representing the solutions of the differential equations underlying the motion of the rocket (embedded in the *6dof_model* block) are sent to FlightGear after the conversion performed in the *Conversion_Block* subsystem. When the states and the actuators information opportunely transformed they are sent through UDP communication protocol to FlightGear, through the *Simulink2FlightGear* block, exploded in Fig. 4.

In this subsystem we can see two *UDP Send* blocks. The first transmits information to FlightGear, while the second sends the same information to a UDP receiver coded in C language for visual representation of the send data. The preparation of the messages transmitted through UDP interface is performed in the S-function `state2xml_wrapper.c`, provided in Appendix I. The IP addresses for the communication are set to `127.0.0.1` for both UDP blocks. The ports are `5001` and `5003`, but any other choice would be valid.

The co-simulation is shown in Fig. 5, where we can see the rocket motion controlled by Simulink.

Note that on the bottom-right corner the UDP receiver is open to receive positions back from FlightGear, confirming that the communication protocol works in both directions. On the top-left the telemetry in Simulink is observed. In this specific case we look at altitude, speed, dynamic pressure, Mach number as well as at the roll, pitch, and yaw angles and at the fin deflections (in this case equal to 0 as we are in the initial ascent part). For this specific setup the real-time pacer was set to 2, meaning that the animation time accelerated by a factor 2 with respect to the physical mission time. This choice is a trade-off required to avoid too long animations, while at the same time still allowing to observe the details of interest, and can be easily adjusted according to the needs of any specific mission. For a concrete example the full animation can be seen in Youtube by scanning the QR code depicted in Fig. 6.

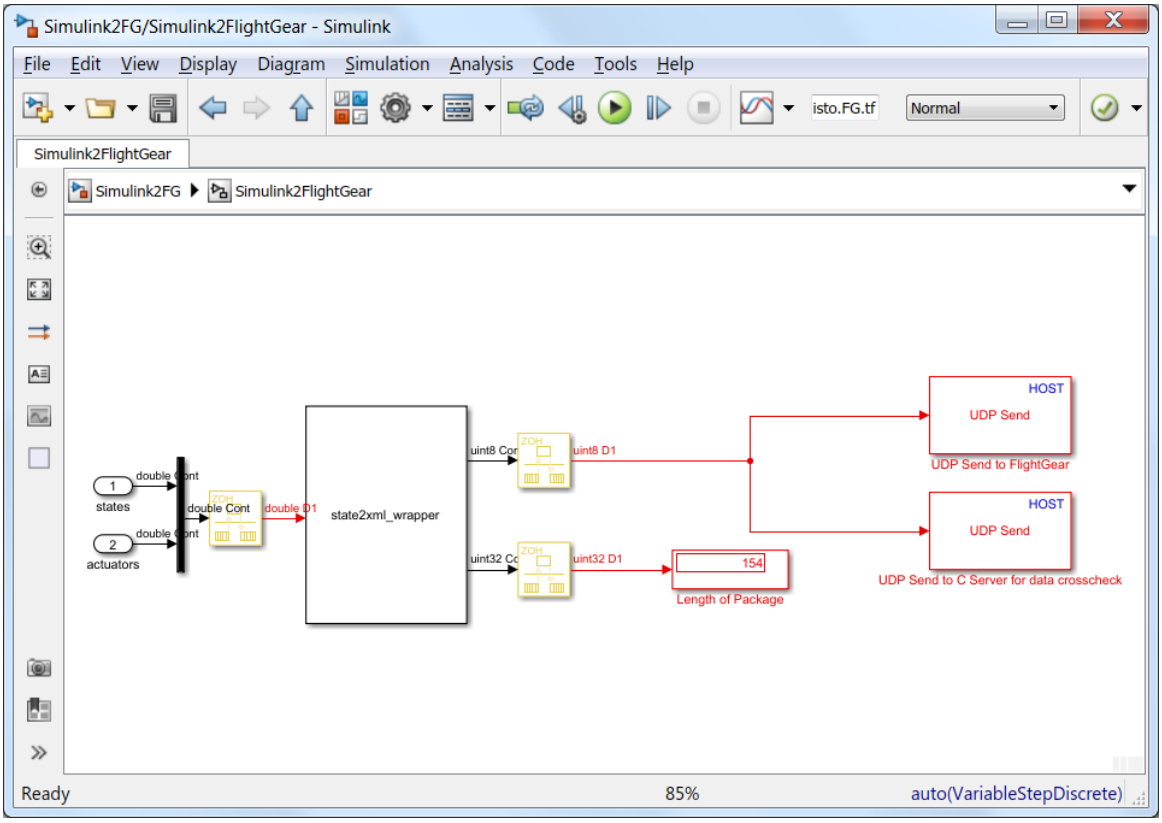


Fig. 4 UDP communication between Simulink and FlightGear.

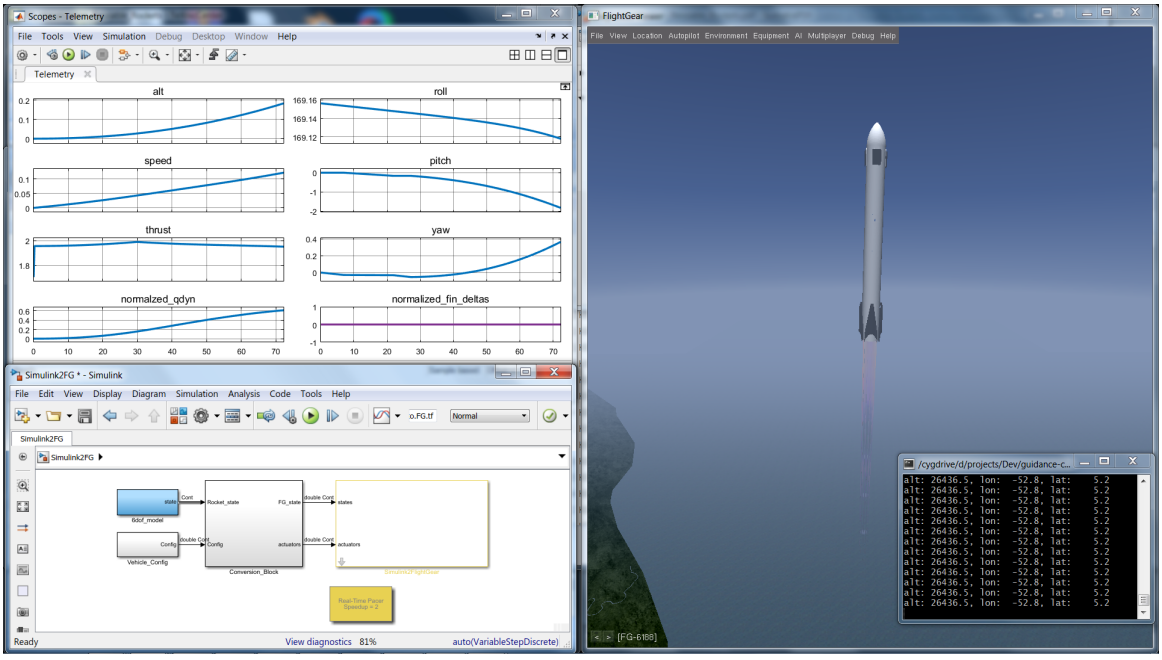


Fig. 5 Simulink-FlightGear Co-simulation.

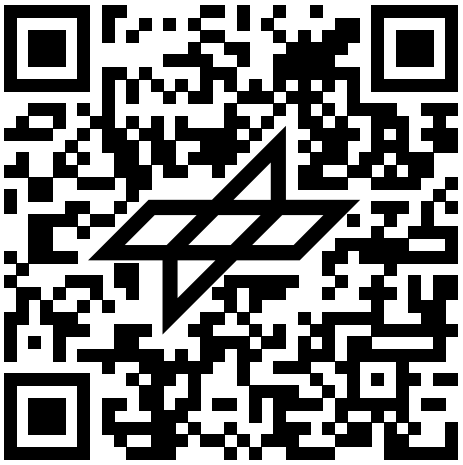


Fig. 6 QR code to see an example of the proposed co-simulation framework applied to the motion of reusable rockets on Youtube.

Although here we limit the number of variables to be transmitted and received to the minimum number required to have a satisfactory visualization of the motion it is possible to extend it in straightforward manner to any desired set of variables. The reader can refer to the appendices provided to see concrete examples of how the variables to be exchanged in both directions during the co-simulation can be defined.

V. Conclusions

This paper proposes a novel use of FlightGear, an open-source software typically adopted to simulate and visualize aircraft motion, for the visualization and the co-simulation of the motion of reusable rockets together with Simulink. This approach enhances the capabilities of Matlab and Simulink themselves, while providing a straightforward visualization tool to verify the correctness of the complex behaviors required to perform demanding missions such as the return-to-launch-site profile.

The steps required to perform the proposed co-simulation have been illustrated, and emphasis has been given to the use of a complete end-to-end open-source toolchain, to maximize its usage within the Guidance, Navigation, and Control research community. The toolchain includes flexible and popular solutions, including the aforementioned FlightGear, Blender, and C code.

The proposed co-simulation approach is especially valuable for the rising research area of reusable rockets, but is completely generic, and can be therefore applied to other applications, such as multi-stage expendable launch vehicles, unmanned aircraft vehicles, and commercial drones.

Appendix I - C code

The C-code used for the communication between Simulink and FlightGear is based on a simple C function embedded within an S-function. Here below the source code, including the header file, and the associated S-function are provided.

A - state2xml.c

```
#include "state2xml.h"
#include <string.h>

/**
 * state2xml sends to FG the data we want to use to control the rocket through generic protocol

 */
void state2xml( real_T* state, /* input */
                  uint8_T* xmlString, uint32_T* stringLength ) /* output */
{
   /* put state parameters into XML string */
   *stringLength = sprintf( xmlString, XML_TEMPLATE ,
               state[0], state[1], state[2],
        state[3], state[4], state[5],
        state[6], state[7], state[8],
        state[9], state[10], state[11],
        state[12], state[13], state[14],
        state[15], state[16], state[17],
        state[18], state[19], state[20],"\n");

}
```

B - state2xml.h

```
#ifndef _STATE2XML_H_
#define _STATE2XML_H_

#include "simstruc.h"

/* #define DEBUG */
#ifdef DEBUG
#include "mex.h"
#endif

/**
 * number of input ports
 * - state
 */
#define INPUT_PORT_NUM 1

/**
 * number of output ports
 * - XML string data in UINT32 array format
 * - string length in bytes
 */
#define OUTPUT_PORT_NUM 2

#define STATE_PARAM_NUM 21

#define MAX_OUT_DATA_SIZE 2048

// only for visualization on the paper - place lines 29 and 30 on the same line
```

```
#define XML_TEMPLATE "\n%6.3f\t%6.3f\t%6.3f\t%6.3f\t%6.3f\t%6.3f\t%6.3f\t%6.3f\t%6.3f\t%6.3f\t%6.3f\t%6.3f
                    \t%6.3f\t%6.3f\t%6.3f\t%6.3f\t%6.3f\t%6.3f\t%6.3f\t%6.3f\t%6.3f\t%6.3f\t"

void state2xml( real_T* state, /* input */
                  uint8_T* xmlString, uint32_T* stringLength ); /* output */

#endif /*_STATE2XML_H_*/
```

C - state2xml_wrapper.c

```
// S-Function name is state2xml_wrapper
#define S_FUNCTION_NAME state2xml_wrapper
#define S_FUNCTION_LEVEL 2

/*
 * Need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"
#include "state2xml.h"

/*====================*
 * S-function methods *
 *====================*/

// Function: mdlInitializeSizes
static void mdlInitializeSizes(SimStruct *S)
{
    int i;
    /* See sfuntmpl_doc.c for more details on the macros below */

    ssSetNumSFcnParams(S, 0); /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        /* Return if number of expected != number of actual parameters */
        return;
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if (!ssSetNumInputPorts(S, INPUT_PORT_NUM)) return;
    /* state parameter */
    ssSetInputPortWidth(S, 0, STATE_PARAM_NUM);
    ssSetInputPortDataType(S, 0, SS_DOUBLE);


    /* all port use the same setting */
    for( i=0; i<INPUT_PORT_NUM; ++i )
    {
        ssSetInputPortRequiredContiguous(S, i, true); /*direct input signal access*/
        ssSetInputPortDirectFeedThrough(S, i, 1);
    }

    if (!ssSetNumOutputPorts(S, OUTPUT_PORT_NUM)) return;
    /* xml string in UINT8 array */
    ssSetOutputPortDataType(S, 0, SS_UINT8);
    ssSetOutputPortWidth(S, 0, MAX_OUT_DATA_SIZE );

    /* string length */
```

```
50     ssSetOutputPortDataType(S, 1, SS_UINT32);
51     ssSetOutputPortWidth(S, 1, 1);
52
53     ssSetNumSampleTimes(S, 1);
54     ssSetNumRWork(S, 0);
55     ssSetNumIWork(S, 0);
56     ssSetNumPWork(S, 0);
57     ssSetNumModes(S, 0);
58     ssSetNumNonsampledZCs(S, 0);
59
60     ssSetOptions(S, 0);
61 }
62
63 // Function: mdlInitializeSampleTimes
64 static void mdlInitializeSampleTimes(SimStruct *S)
65 {
66     ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
67     ssSetOffsetTime(S, 0, 0.0);
68 }
69
70 #undef MDL_INITIALIZE_CONDITIONS
71 #if defined(MDL_INITIALIZE_CONDITIONS)
72   // Function: mdlInitializeConditions
73   static void mdlInitializeConditions(SimStruct *S)
74   {
75   }
76 #endif
77
78 #undef MDL_START /* Change to #undef to remove function */
79 #if defined(MDL_START)
80   // Function: mdlStart
81   static void mdlStart(SimStruct *S)
82   {
83   }
84 #endif
85
86 // Function: mdlOutputs
87 static void mdlOutputs(SimStruct *S, int_T tid)
88 {
89     real_T *state = (real_T*) ssGetInputPortSignal(S,0);
90     uint8_T *xmlString = (uint8_T*)ssGetOutputPortSignal(S,0);
91     uint32_T *stringLength = (uint32_T*)ssGetOutputPortSignal(S,1);
92     state2xml(state, xmlString, stringLength);
93 }
94
95 #undef MDL_UPDATE
96 #if defined(MDL_UPDATE)
97   // Function: mdlUpdate
98
99   static void mdlUpdate(SimStruct *S, int_T tid)
100  {
101  }
102 #endif /* MDL_UPDATE */
103
104 #undef MDL_DERIVATIVES //
105 #if defined(MDL_DERIVATIVES)
106   // Function: mdlDerivatives
107   static void mdlDerivatives(SimStruct *S)
108   {
```

```
109   }
110 #endif /* MDL_DERIVATIVES */
111
112 // Function: mdlTerminate
113 static void mdlTerminate(SimStruct *S)
114 {
115     UNUSED_ARG(S); /* unused input argument */
116 }
117
118 /*=============================*
119  * Required S-function trailer *
120  *=============================*/
121
122 #ifdef MATLAB_MEX_FILE /* Is this file being compiled as a MEX-file? */
123 #include "simulink.c" /* MEX-file interface mechanism */
124 #else
125 #include "cg_sfun.h" /* Code generation registration function */
126 #endif
```

Appendix II - Communication Protocol

The xml protocol used to send variables to FlightGear is listed here below. Note that since the equations of motion are modeled in Simulink no inverse communication would be needed. Nevertheless, the protocol below also allows bi-directional communication. The data sent to FlightGear is specified from line 4 to 154, while lines from 155 to 176 specify the data received by FlightGear. The extension to other variables of interest is straightforward, and the full list of variables that can be manipulated is observable from the html FlightGear Φ Interface, that can be open as web-page, and set in this case to `http://localhost:5400/#Simulator/Properties`. In the example coded here we receive back altitude, longitude, and latitude, that can be listened to by a UDP receiver set at port 5002, and IP address `127.0.0.1`. These settings are all visible in Appendix III, where the call to FlightGear is listed.

rocket_protocol.xml

```
1  <?xml version="1.0"?>
2      <PropertyList>
3      <generic>
4           <input>
5           <line_separator>\n</line_separator>
6           <var_separator>\t</var_separator>
7
8           <chunk>
9              <name>sent altitude </name>
10             <node>/position/altitude-ft</node>
11             <type>float</type>
12             <format>alt: %6.3f ft</format>
13          </chunk>
14
15          <chunk>
16             <name>sent latitude </name>
17             <node>/position/latitude-deg</node>
18             <type>float</type>
19             <format>lat: %6.3f deg</format>
20          </chunk>
21
22          <chunk>
23             <name>sent longitude </name>
24             <node>/position/longitude-deg</node>
25             <type>float</type>
26             <format>lon: %6.3f deg</format>
27          </chunk>
28
29      <chunk>
30             <name>sent roll </name>
31             <node>/orientation/roll-deg</node>
32             <type>float</type>
33             <format>roll: %6.3f deg</format>
34          </chunk>
35
36      <chunk>
37             <name>sent pitch </name>
38             <node>/orientation/pitch-deg</node>
39             <type>float</type>
40             <format>pitch: %6.3f deg</format>
41          </chunk>
42
43      <chunk>
44             <name>sent yaw </name>
45             <node>/orientation/heading-deg</node>
46             <type>float</type>
```

```
47            <format>yaw: %6.3f deg</format>
48          </chunk>
49
50          <chunk>
51            <name>sent fin1 deployment cmd </name>
52            <node>/fins-states/fin1-rot</node>
53            <type>float</type>
54            <format>fin #1 deployment: %6.3f</format>
55          </chunk>
56
57          <chunk>
58            <name>sent fin2 deployment cmd </name>
59            <node>/fins-states/fin2-rot</node>
60            <type>float</type>
61            <format>fin #2 deployment: %6.3f</format>
62          </chunk>
63
64          <chunk>
65            <name>sent fin3 deployment cmd </name>
66            <node>/fins-states/fin3-rot</node>
67            <type>float</type>
68            <format>fin #3 deployment: %6.3f</format>
69          </chunk>
70
71          <chunk>
72            <name>sent fin4 deployment cmd </name>
73            <node>/fins-states/fin4-rot</node>
74            <type>float</type>
75            <format>fin #4 deployment: %6.3f</format>
76          </chunk>
77
78          <chunk>
79            <name>sent fin1 control cmd </name>
80            <node>/hinge-rotations/hinge1-rot</node>
81            <type>float</type>
82            <format>fin #1 angle: %6.3f</format>
83          </chunk>
84
85          <chunk>
86            <name>sent fin2 control cmd </name>
87            <node>/hinge-rotations/hinge2-rot</node>
88            <type>float</type>
89            <format>fin #2 angle: %6.3f</format>
90          </chunk>
91
92          <chunk>
93            <name>sent fin3 control cmd </name>
94            <node>/hinge-rotations/hinge3-rot</node>
95            <type>float</type>
96            <format>fin #3 angle: %6.3f</format>
97          </chunk>
98
99          <chunk>
100           <name>sent fin4 control cmd </name>
101           <node>/hinge-rotations/hinge4-rot</node>
102           <type>float</type>
103           <format>fin #4 angle: %6.3f</format>
104         </chunk>
105
```

```
106     <chunk>
107            <name>sent leg1 state </name>
108            <node>/legs-states/leg1-rot</node>
109            <type>float</type>
110            <format>leg #1 deployment: %6.3f</format>
111          </chunk>
112
113     <chunk>
114            <name>sent leg2 state </name>
115            <node>/legs-states/leg2-rot</node>
116            <type>float</type>
117            <format>leg #2 deployment: %6.3f</format>
118          </chunk>
119
120     <chunk>
121            <name>sent leg3 state </name>
122            <node>/legs-states/leg3-rot</node>
123            <type>float</type>
124            <format>leg #3 deployment: %6.3f</format>
125          </chunk>
126
127     <chunk>
128            <name>sent leg4 state </name>
129            <node>/legs-states/leg4-rot</node>
130            <type>float</type>
131            <format>leg #4 deployment: %6.3f</format>
132          </chunk>
133
134     <chunk>
135            <name>sent TVC state beta1 </name>
136            <node>/TVC/beta1</node>
137            <type>float</type>
138            <format>tvc angle #1: %6.3f deg</format>
139          </chunk>
140
141     <chunk>
142            <name>sent TVC state beta2 </name>
143            <node>/TVC/beta2</node>
144            <type>float</type>
145            <format>tvc angle #2: %6.3f deg</format>
146          </chunk>
147
148     <chunk>
149            <name>sent TVC state Thrust </name>
150            <node>/TVC/thrust</node>
151            <type>float</type>
152            <format>thrust %6.3f </format>
153          </chunk>
154       </input>
155       <output>
156          <line_separator>\n</line_separator>
157          <var_separator>, </var_separator>
158          <chunk>
159            <name>received altitude</name>
160            <node>/position/altitude-ft</node>
161            <type>float</type>
162            <format>alt: %6.1f</format>
163           </chunk>
164          <chunk>
```

```
165             <name>received lon</name>
166             <node>/position/longitude-deg</node>
167             <type>float</type>
168             <format>lon: %6.1f</format>
169           </chunk>
170           <chunk>
171             <name>received lat</name>
172             <node>/position/latitude-deg</node>
173             <type>float</type>
174             <format>lat: %6.1f</format>
175           </chunk>
176       </output>
177     </generic>
178     </PropertyList>
```

Appendix III - FlightGear Call

To perform the co-simulation FlightGear is invoked through the following call. In line 1 we call FlightGear through `fgfs` and we specify the aircraft to be used (in this case `rocket`), while in line 2 we specify the time of the flight (noon for better light conditions). We can observe that the rocket protocol for transmission and reception are specified in lines 3 and 4, together with the desired frame-rate (in this case 30 Hertz), and the corresponding IP address and relative ports for the transmission and the reception of the specified variables (`5001` and `5002`, respectively). Note that we show here a minimal setup (e.g., by disabling multiple features, as visible in lines 5-18) to minimize the hardware specifications sufficient to perform a smooth co-simulation. Finally, in line 19 we can see the port specified for using the Φ interface is 5400, in accordance to what described in Appendix II, whereas we specify the size of the FlightGear window in line 20.

```
1  fgfs --aircraft=rocket
2  --timeofday=noon
3  --generic=socket,in,30,127.0.0.1,5001,udp,rocket_protocol
4  --generic=socket,out,30,127.0.0.1,5002,udp,rocket_protocol
5  --disable-random-objects
6  --disable-ai-traffic
7  --disable-ai-models
8  --disable-specular-highlight
9  --disable-clouds
10 --disable-clouds3d
11 --fog-fastest
12 --visibility=5000
13 --disable-distance-attenuation
14 --disable-real-weather-fetch
15 --disable-random-vegetation
16 --disable-random-buildings
17 --disable-rembrandt
18 --disable-sound
19 --httpd=5400
20 --geometry=1600x900
```

Appendix IV - Nasal files

The Nasal files used in this work are listed here below. They include `rocket.nas` for the initialization of the rocket properties, and `launchpad.nas` to place the launchpad in the scenario.

A - rocket.nas

```
### rocket.nas ###
#startup-properties-----------------------------------------------------------------
###
init_rocket = func {

print ("setting rocket properties...");

#fins
setprop("/fins-states/fin1-rot", 0);
setprop("/fins-states/fin2-rot", 0);
setprop("/fins-states/fin3-rot", 0);
setprop("/fins-states/fin4-rot", 0);

#hinges
setprop("/hinge-rotations/hinge1-rot", 0);
setprop("/hinge-rotations/hinge2-rot", 0);
setprop("/hinge-rotations/hinge3-rot", 0);
setprop("/hinge-rotations/hinge4-rot", 0);

#legs
setprop("/legs-states/leg1-rot", 0);
setprop("/legs-states/leg2-rot", 0);
setprop("/legs-states/leg3-rot", 0);
setprop("/legs-states/leg4-rot", 0);

#thrust
setprop("/TVC/thrust", 1);
setprop("/TVC/beta1", 0);
setprop("/TVC/beta2", 0);
setprop("/TVC/spin", 1);

#orientation
setprop("/orientation/pitch-deg", 0);
setprop("/orientation/heading-deg", 0);
setprop("/orientation/roll-deg", 0);

#pos
setprop("/position/altitude-ft", 0);
setprop("/position/latitude-deg", 5.233868);
setprop("/position/longitude-deg", -52.752);

#vel

print ("rocket ready to lift-off!");

}

### end of rocket.nas ###
```

B - launchpad.nas

```
print ("loading launch platform...");

geo.put_model("Aircraft/rocket/Models/launchpad.xml", 5.233977, -52.7517, 0, 0);

print ("launch plaform loaded.");
```

References

[1] Martinez, M., Sitawarin, C., Finch, K., Meincke, L., Yablonski, A., and Kornhauser, A. L., "Beyond Grand Theft Auto V for Training, Testing and Enhancing Deep Learning in Self Driving Cars," *ArXiv*, Vol. abs/1712.01397, 2017.

[2] Microsoft, "Flight Simulator," , 2020. URL `https://www.flightsimulator.com/`, retrieval Date: 01-Jun-2020.

[3] Squad Private Division, "Kerbal Space Program," , 2020. URL `https://www.kerbalspaceprogram.com/`.

[4] Manley, S., "Kerbal Space Program 101 - Tutorial For Beginners - Construction, Piloting, Orbiting," , 2012. URL `https://www.youtube.com/watch?v=tgPr4q5tj-Q&`.

[5] FlightGear Community, "Flightgear Flight Simulator," , 2020. URL `https://www.flightgear.org/`.

[6] Berndt, J., and De Marco, A., "Progress On and Usage of the Open Source Flight Dynamics Model Software Library, JSBSim," *AIAA Modeling and Simulation Technologies Conference*, American Institute of Aeronautics and Astronautics, 2009. https://doi.org/10.2514/6.2009-5699.

[7] Mathworks, "Working with the Flight Simulator Interface," , 2016. URL https://mathworks.com/help/aeroblks/working-with-the-flight-simulator-interface.html#f3-19773.

[8] Coiro, D., De Marco, A., and Nicolosi, F., "A 6DOF Flight Simulation Environment for General Aviation Aircraft with Control Loading Reproduction," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, American Institute of Aeronautics and Astronautics, 2007. https://doi.org/10.2514/6.2007-6364.

[9] Cao, P., Hu, X., and Zhang, G., "Interface research and flight control based on FlightGear," *2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2017, pp. 397–402.

[10] Lu, P., and Geng, Q., "Real-time simulation system for UAV based on Matlab/Simulink," *2011 IEEE 2nd International Conference on Computing, Control and Industrial Engineering*, IEEE, 2011. https://doi.org/10.1109/ccieng.2011.6008043.

[11] Chakraborty, I., Ahuja, V., Comer, A., and Mulekar, O., "Development of a Modeling, Flight Simulation, and Control Analysis Capability for Novel Vehicle Configurations," *AIAA Aviation 2019 Forum*, American Institute of Aeronautics and Astronautics, 2019. https://doi.org/10.2514/6.2019-3112.

[12] Ying, J., Luc, H., Dai, J., and Pan, H., "Visual flight simulation system based on Matlab/FlightGear," *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, IEEE, 2017. https://doi.org/10.1109/iaeac.2017.8054444.

[13] Feng, T., ZhuLei, F., Yu, L., and Rixu, D., "Certain Type of Aircraft Virtual Cockpit Research and Development Based on FlightGear," *2013 Third International Conference on Instrumentation, Measurement, Computer, Communication and Control*, IEEE, 2013. https://doi.org/10.1109/imccc.2013.102.

[14] Aschauer, G., Schirrer, A., and Kozek, M., "Co-Simulation of Matlab and FlightGear for Identification and Control of Aircraft," *IFAC-PapersOnLine*, Vol. 48, No. 1, 2015, pp. 67–72. https://doi.org/10.1016/j.ifacol.2015.05.071.

[15] Dumont, E., Ishimoto, S., Tatiossian, P., Klevanski, J., Reimann, B., Ecker, T., Witte, L., Riehmer, J., Sagliano, M., Giagkozoglou, S., Petkov, I., Rotärmel, W., Schwarz, R. G., Seelbinder, D., Markgraf, M., Sommer, J., Pfau, D., and Martens, H., "CALLISTO: a Demonstrator for Reusable Launcher Key Technologies," 2019.

[16] Sagliano, M., Tsukamoto, T., Maces-Hernandez, J. A., Seelbinder, D., Ishimoto, S., and Dumont, E., "Guidance and Control Strategy for the CALLISTO Flight Experiment," *8th European Conference for Aeronautics and Aerospace Sciences (EUCASS)*, 2019.

[17] Blackmore, L., "Autonomous Precision Landing of Space Rockets," *National Academy of Engineering: "The Bridge on Frontiers of Engineering"*, Vol. 4, No. 46, 2016, pp. 15–20. URL https://www.nae.edu/164334/Autonomous-Precision-Landing-of-Space-Rockets.

[18] Blender Community, "Blender," , 2020.

[19] FlightGear Community, "Howto:Work with AC3D files in Blender," , 2020. URL http://wiki.flightgear.org/Howto:Work_with_AC3D_files_in_Blender, retrieved on Oct, 27, 2020.

[20] Klevanski, J., Ecker, T., Riehmer, J., Reimann, B., Dumont, E., and Chavagnac, C., "Aerodynamic Studies in Preparation for CALLISTO - Reusable VTVL Launcher First Stage Demonstrator," *69th International Astronautical Congress*, 2018.

[21] FlightGear Community, “How to: Install aircraft,” , 2020. URL http://wiki.flightgear.org/Howto:Install_aircraft.

[22] Mathworks, “RealTime Pacer,” , 2010. URL https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/29107/versions/2/previews/RealTime_Pacer/realtime_pacer_help.html.