Towards VR-Based Early Design Interaction for electric Vertical Take-Off & Landing (eVTOL) Cyber-Physical Models

Meaghan Podlaski, Luigi Vanfretti, Jacob Monteneiri, Abhijit Khare, James Lewin, Eric Segall
Rensselaer Polytechnic Institute (USA)

Abstract

In the development of complex engineered systems, such as eVTOL, it is difficult to integrate human interaction in early design stages of the engineering design process, for both designer and end-user. This opens the research questions: (1) can the design improve if designers interact with the virtual prototype during early design phases? and (2) can the end-user interaction at early phases fulfil unperceived end-user needs and requirements? To address such questions, a “virtual reality” environment is needed allowing for designers/end-users to interact with the “virtual prototypes”. This paper discusses foundational work towards the vision of such environments. The paper aims to address how virtual reality (VR)-based technologies can enable interaction with eVTOL models early in the design process, providing more flexibility to study models with additional human feedback. The proposed methodology leverages open access and open source standards for modelling cyber-physical systems and for model portability. To illustrate the methodology, an eVTOL model is expanded to incorporate user-interaction functionalities and VR within a Modelica tool. Then, proof-of-concept on portability and interoperability is shown by exporting the drone model using the FMI standard into two different game development environments. The results aim to highlight the importance of the use of open access and open source modeling, simulation and model exchange standards, for model portability into new VR-ready interaction environments that weren’t initially conceived for simulation purposes.

1. Notation and abbreviations

eVTOL - electric Vertical Take-Off and Landing
FMI - Functional mock-up interface
FMU - Functional mock-up unit
VR - Virtual reality

2. Introduction

a. Motivation

With the development of complex, multiple-domain systems such as eVTOL, physical prototypes can be costly and difficult to build. Simulation-based studies using well-understood physics-based models are extremely valuable to determine which designs best comply with specifications and requirements before building them. One aspect that is hard to integrate within the typical engineering design process, is the study of how humans will interact with the system, which can be either the designers themselves or the end-user. This is harder to quantify than when dealing with typical engineering requirements, which largely do not depend on human factors in early design phases. This opens the following research questions: (1) can the design improve if designers can interact with the virtual prototype during early design phases, and (2) can the end-user interaction at early phases fulfill unperceived end-user needs and requirements as compared to existing approaches. To address such questions, a “virtual reality" environment is necessary, allowing for designers/end-users to interact with the “virtual prototypes”. This paper discusses preliminary work to address how virtual reality-based technologies can enable interaction with eVTOL models early in the design process, providing more flexibility to study and simulate models with additional human feedback and prior to building a physical prototype and testing the physical system. The proposed systems engineering method can also provide a basis for an eVTOL virtual reality-based flight training simulator.

The proposed method proposed herein consists of implementing a drone model, which has been created using the object-oriented modeling language, Modelica. The choice of this language is to leverage interoperability for model-reusability in multiple modeling and simulation tools, as well as model export through the Functional Mock-up Interface (FMI). The multi-domain models were used to create each aspect of the drone, specifically focusing on the mechanical and electrical domains. This paper expands the work in (Podlaski et al., 2020) to show methods to simulate and interact with the drone in VR using various visualization tools and game development environments. Firstly, this has been achieved in this paper using the Modelica-based DLR Visualization library to demonstrate interaction and animation with Modelica specific tools that do not require model export. Secondly, to exploit VR-ready environments not originally conceived for engineering, the paper shows how the model can be exported into the Unity and Unreal game development environments using the FMI standard to demonstrate interaction with virtual reality environments through model export methods. By integrating models with a wide set of virtual reality environments and simulation tools, cases such as multiple drone model instances can be controlled concurrently, dealing with external factors, and providing collision detection and response.

b. Previous Works

Previous research have provided the present work with a foundation for human and hardware-in-the-loop methodologies through software developments. This includes developing models to connect external devices to interact with the library (Thiele et al., 2017; Bellman, 2009). The Modelica External Device Drivers library allows for hardware such as keyboards and joysticks to provide interactive inputs to communicate with and control models written in the same modeling language the drone model was made. This provides a foundation for the drone model described in this paper to be controlled in real time simulations and interactive inputs.

Early design interaction also relies heavily on visualization of these models and systems we wish to study, develop, and interact with. A library for visualization using Modelica has also been developed for uses such as this eVTOL application (Hellerer et al., 2014). This provides one method of visualization and simulation of models, where code or model export into another tool is not required. This allows us to explore one method of visualization and interaction with the model.

The Modelica programming language has also been used for the development of models for virtual reality training simulators (Martin-Villalba et al., 2010). Models created using the Modelica programming language can be exported into other simulation tools using the functional mock-up interface (FMI) standard (Modelica Association, n.d.). The equations, parameters, and functions of the model are compiled into C code and header functions in a functional mock-up unit (FMU) structure to be imported into another tool; in this paper, FMUs are exported from the Modelica development environment and imported into a gaming engine that is compliant with the FMI standard. This allows for the model to be simulated and controlled from the new tool, so the features of the gaming engine can be utilized for simulation. This method expands the usability of the model, so one model can be used across multiple tools.

c. Paper Contributions

The proposed methods, models and simulation studies presented in this paper utilizes these previous efforts to develop human and hardware-in-the-loop studies for virtual reality-based interaction of a small eVTOL. This paper primarily focuses on how to expand existing models for visualization, animation and VR-based interaction with the virtual prototype of the quad-copter. This visualization and VR-based interaction has been explored using VR-based game engines and a Modelica-specific animation tool.

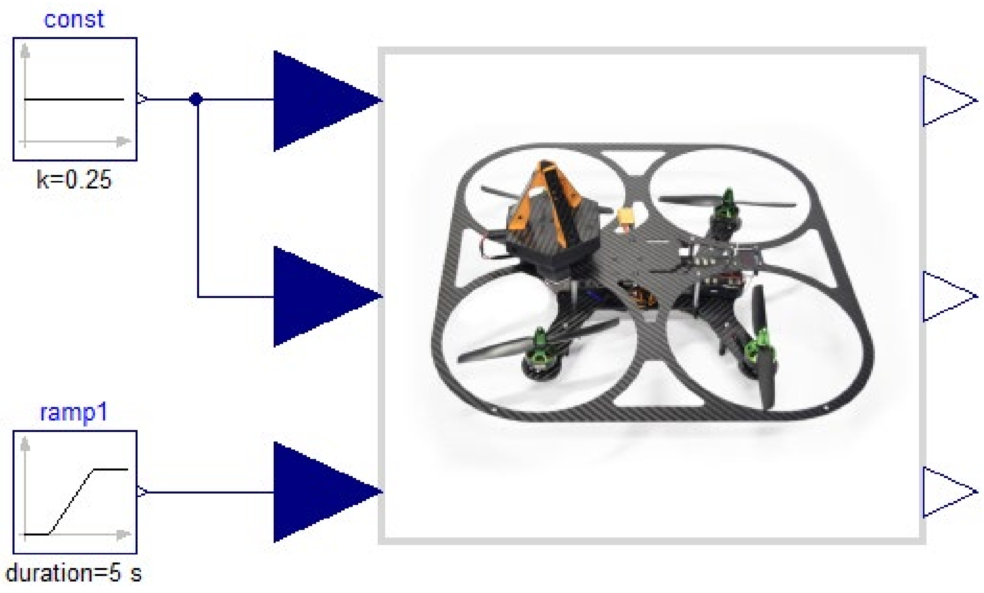


Figure 1: Drone model configured to simulate a 1 m/s 5 second ramp input in the Z direction.

3. Multi-Domain Drone Model

The library and models were implemented in the Dymola software and using the Modelica language (Modelica Association). The library, developed by the authors of this paper, is open source and can be found at: https://github.com/ALSETLab/Modelica-Drone-3D-FMI/. The drone models are multi-domain models, where each engineering domain is modeled and coupled together through physics-based interfaces. This includes the electrical, mechanical, aerodynamic and control domains (Podlaski et al., 2020). The top level representation of the drone model is shown in Figure 1, where the drone is receiving a command to ramp at a rate of 1 m/s in the Z-direction and remain in its current position in the X and Y-directions.

Figure 2 shows the drone model in Figure 1 from a component level. The model is multi-domain, where the dark blue lines represent the control domain, the light blue lines represent the electrical domain, and the mechanical domain is denoted by the grey lines. The drone uses inputs `xcoord`, `ycoord`, and `zcoord` to command the drone to a specific XYZ coordinate location. The coordinate location of the drone is provided by outputs `xgps`, `ygps`, and `zgps`. The desired coordinates `xcoord`, `ycoord`, and `zcoord` feed into a controller, labeled `MCU` in Figure 2. The `MCU` then calculates the voltage and current applied to each of the propellers to move the drone to the desired location.

Each propeller in the drone uses the battery voltage as a reference, then scales the voltage accordingly to deliver the power needed to move the drone to its commanded position, as shown in Figure 3. This model utilizes both a control signal, which is the `position` input into the model, and the electrical domain to scale the voltage applied to the motor.

The propeller models in Figure 4 consist of a speed controller, a DC motor, and the mechanical model of the blades. The model uses a control signal input (`position`) and an electrical input (`p1`) with a rotational mechanical output to connect the drone propeller dynamics to the airframe (`Airframe`). The grey lines represent the rotational mechanical signals, and the blue lines represent the electrical signals.

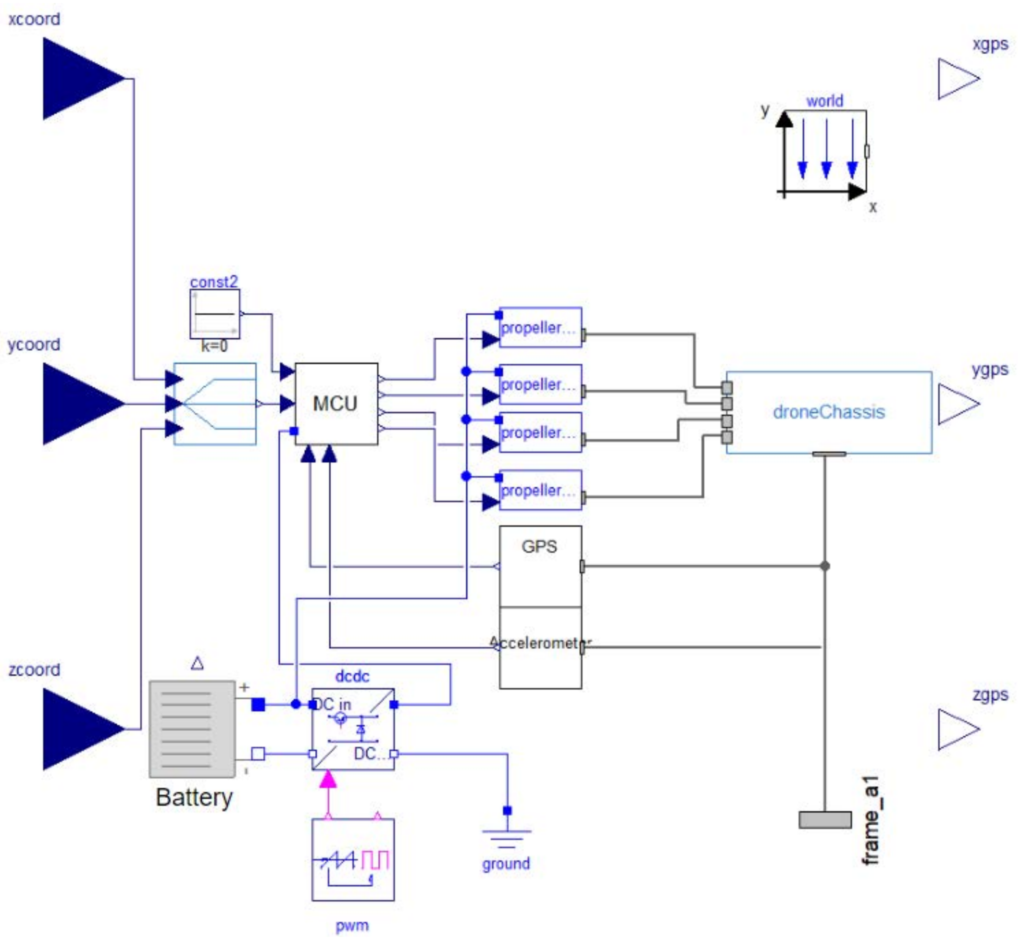


Figure 2: Complete drone model consisting of propellers,motor, controller, and chassis with battery power system.Inputs come from x, y, and z coordinate location.

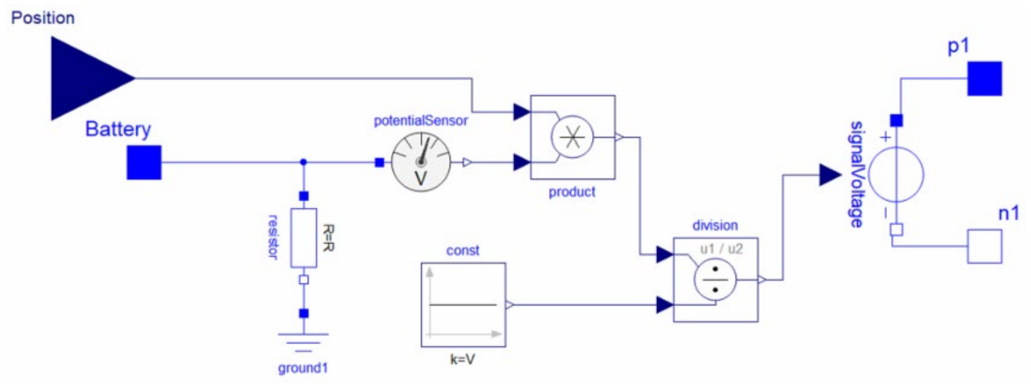


Figure 3: Speed controller using positional input from the MCU and the battery voltage to adjust the power delivered by the motor to follow a command.

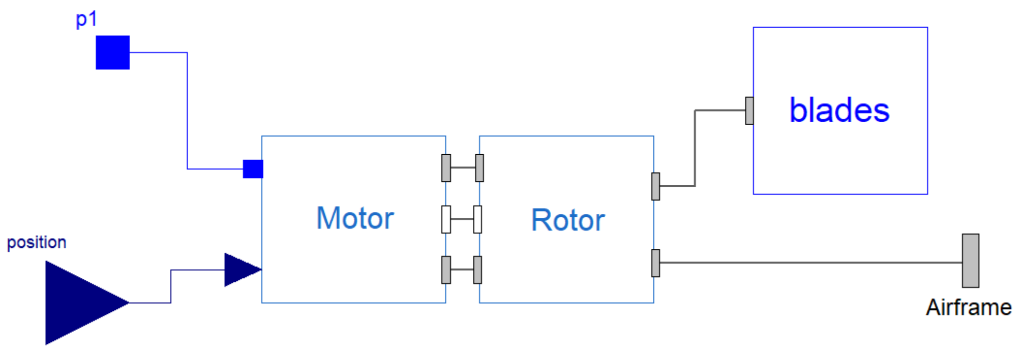


Figure 4: Propeller model consisting of motor, rotor, and blade sub-models. The model uses a control signal input (`position`) and an electrical input (`p1`) with a rotational mechanical output to connect the drone propeller dynamics to the airframe (`Airframe`).

4. Animation and Visualization in Dymola

For visualization purposes, when the drone is simulated, the behavior can be observed as an animation if configured with the resources to do so. The drone has been configured to use 3D Object (.STL) files to represent the propellers and body of the drone in animation, which appears when the drone is simulated. To link the objects to the simulation model variables, the 3D Object files are defined in the chassis and blade models of the drone as `fixedShape` components from MultiBody library (Otter et al., 2003) from the Modelica Standard Library (MSL).

The drone configuration for animation in Dymola (a Modelica-compliant tool from Dassault Systems) on the graphical model level is shown in Figure 1. The drone has three inputs: X, Y, and Z direction, which are labeled from top to bottom respectively. The drone model consists of multiple engineering domains, where each component considers behaviors and equations from the mechanical, electrical, and controls domains.

The initial position of the drone animation is shown in Figure 5, where it is positioned at (0,0,0). The propellers move over time, as shown in Figure 6. The propellers spinning are also shown in Figure 7, where the drone is steadily moving to a height of 5 m over a 5 second period.

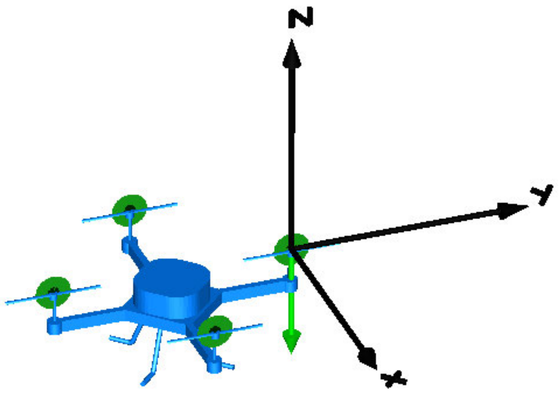


Figure 5: Drone animation in Dymola at t=0 seconds.

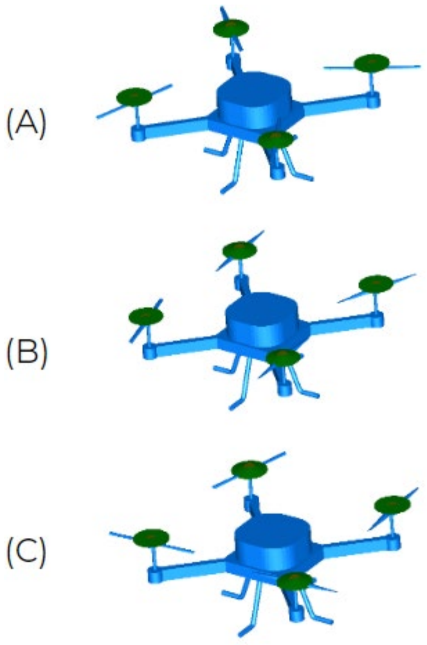


Figure 6: Drone animation in Dymola at t=0 seconds

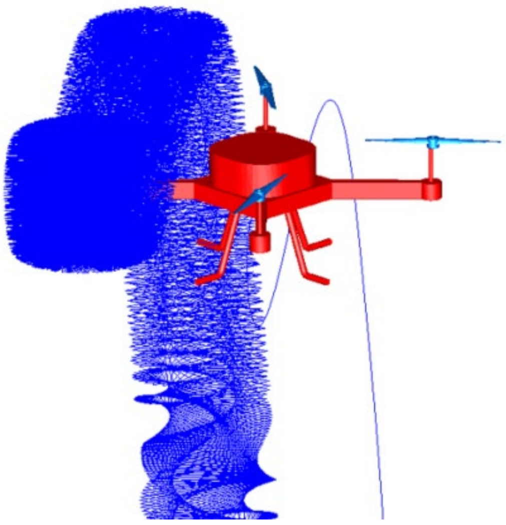


Figure 7: Drone animation with the path of the propeller shown as a trace of the flight path (flying up to 5m).

5. Virtual Reality Integration

The model configuration in Figure 1 is modified where the x, y, and z directional inputs `const` and `ramp1` are replaced with components to allow for interaction with game controllers via the Modelica Device Drivers library (Thiele et al., 2017). The Modelica Device Drivers library enables hardware-in-the-loop (HIL) simulation of models by allowing for the interaction of generic game controllers (e.g. joystick) and keyboard input with models. Figures 8 and 9 show the model used to connect the joystick controllers and keyboard to the drone model for HIL simulation.

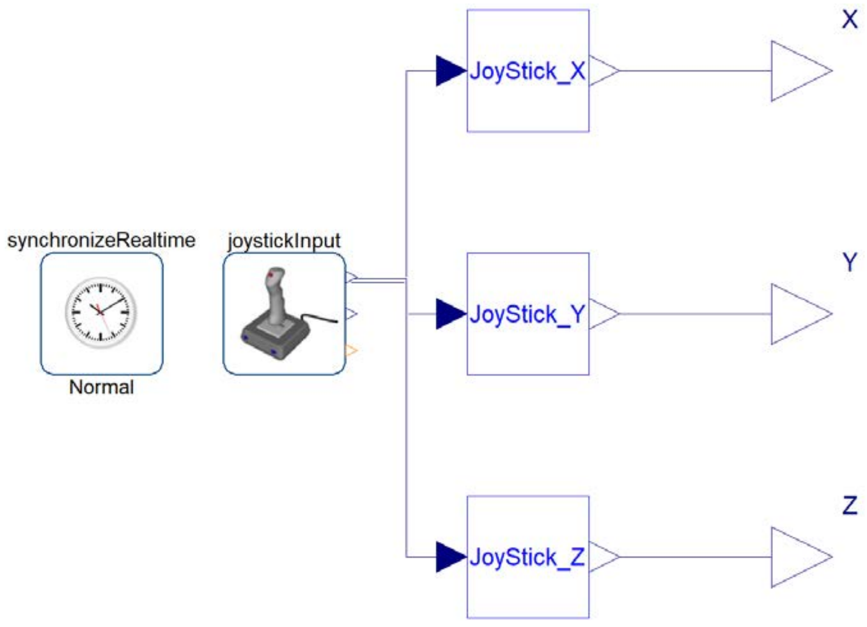


Figure 8: Joystick input converted to X, Y, and Z directional input to connect to the drone.

The `synchronizeRealtime` component synchronizes the simulation time of the simulation process with the operating system's real time clock. The Modelica Device Drivers library was implemented to provide multi-platform support through "soft" synchronization, which means that latency is restricted to a maximum value and that there are no guarantees on the deadlines. However, it can meet today's requirements for human-computer-interaction of computer games. The only important consideration for this application is that the command deadlines from the controller will be met to match the system clock if the simulation is expensive. For the drone example that is studied here, the synchronization capability provided by the library is more than sufficient. The `joystickInput` and `keyboardInput` components in Figures 8 and 9 detect input from the external device and retrieve data to interact with the model according to a specified sampling time.

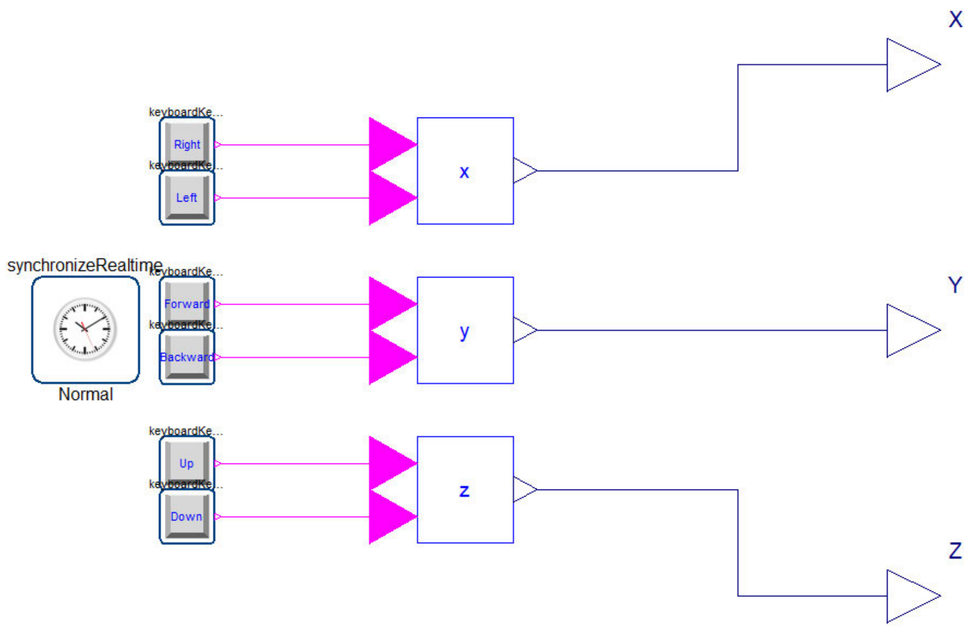


Figure 9: Keyboard input converted to X, Y, and Z directional inputs to connect to the drone.

The data retrieved by the `joystickInput` and `keyboardInput` components are then translated into X, Y, and Z directional control using a PI controller. Figure 10 shows the PI controller of the joystick input. The control system from the keyboard input is similar to the joystick, except the keyboard input explicitly defines the positive and negative direction as different key inputs. This requires an additional step of the control system to compute the direction signal of the drone after integration, as shown in Figure 11.

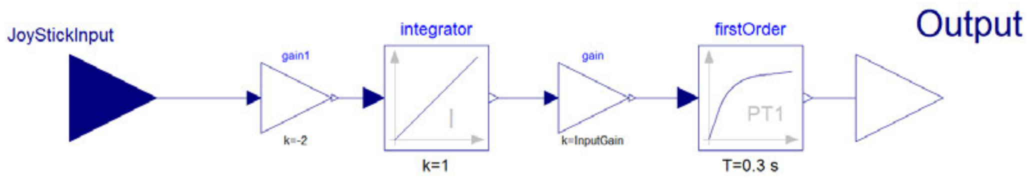


Figure 10: Control system for the joystick HIL input.

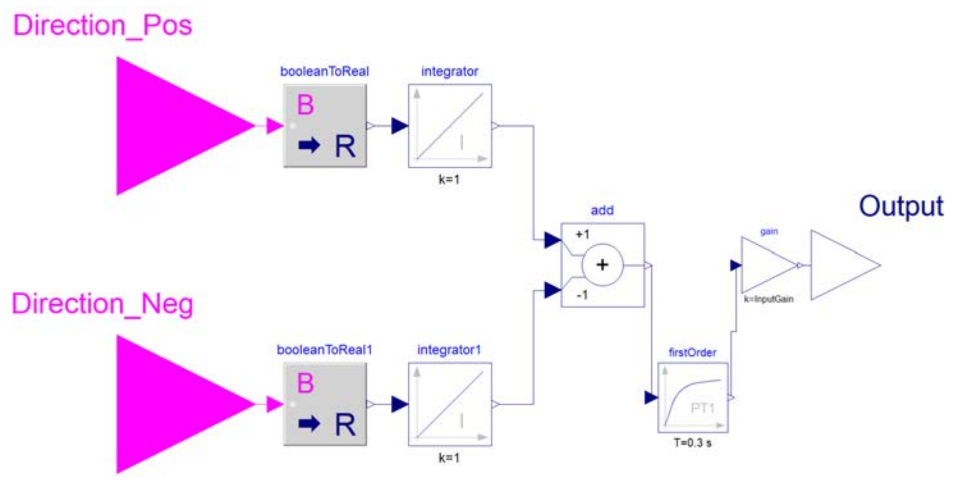


Figure 11: Control system for keyboard HIL input.

After coupling the controllers, the next step is to interface the VR hardware, define the VR scene, camera locations, etc. The full paper will discuss how this was achieved using the DLR Visualization Library (Hellerer et al., 2014) and the HTC Vive Headset, for which results are shown below.

6. Simulation in VR environments

The model structure of the drone allows it to be integrated into multiple gaming engines and virtual reality environments for early design interaction. In this paper, the drone model is used in three different animation tools and game development environments to show the flexibility of the modeling and integration into these VR tools when using open access standards, as discussed below.

a. Functional Mock-Up Interface (FMI) Standard

The FMI standard is crucial in providing a means to communicate the models between the modeling environment and virtual reality environments. It is an open source and open access standard where models are communicated between different software tools as a functional mock-up units (FMU) (FMI Standard, n.d.). The standard defines a container and an interface to exchange dynamic models through a combination of XML files, binaries, and C code. These models can be communicated via two methods: (1) model exchange (2) co-simulation. When models are exchanged between software through model exchange, the attributes of the dynamic model are sent to the new software tool via FMU, where the solver in the new tool is used to simulate the model. In co-simulation, the FMU is sent to the new tool and the solver from the tool the FMU was generated in is used to simulate the model. When the drone models are imported into Unity and Unreal Engine with an FMU, the models are imported using co-simulation.

b. DLR Visualization Library

Using the drone configuration outlined in (Podlaski et al., 2020) and the HIL components outlined in the previous sections, the model in Figure 12 is configured to simulate the model in a VR environment using the DLR Visualization Library (Hellerer et al., 2014). This example does not require to export the model as an FMU, everything is configured and executed within the Dymola tool. In this example, the drone is connected to a keyboard input. The `world` component applies gravity and a reference frame to all moving components in the environment. It is also attached to `shape1`, which defines the physical terrain (i.e. the scene) that the drone is interacting with.

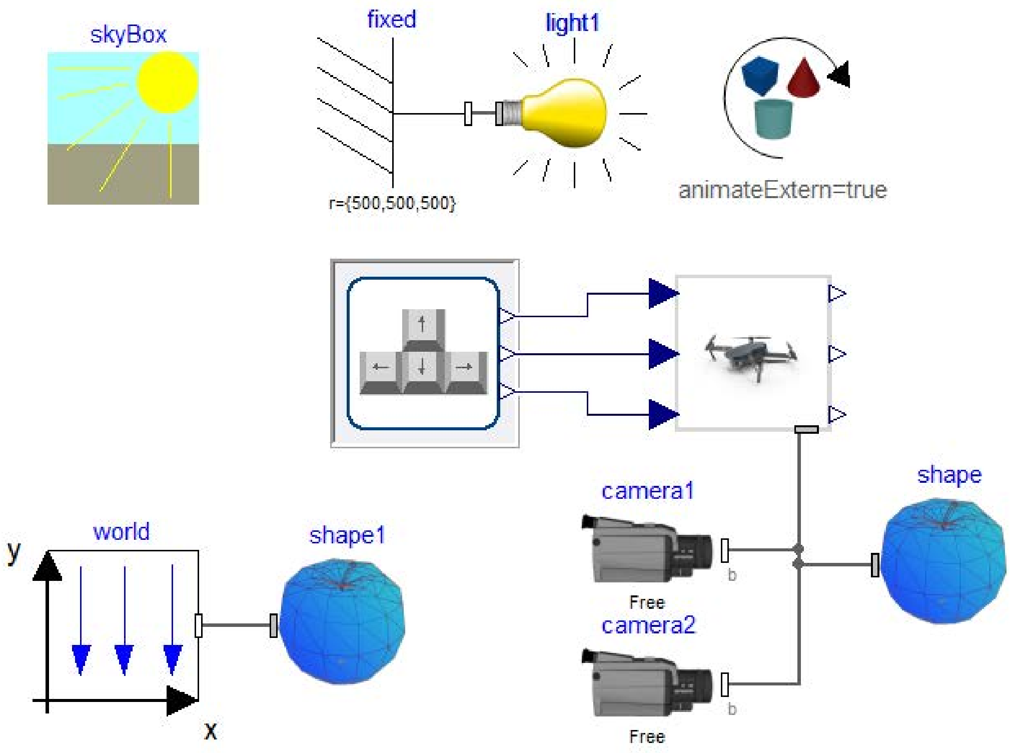


Figure 12: Drone model configured HIL simulation using the DLR Visualization Library

The `camera` and `camera1` components are connected to the chassis of the drone so that the user can follow the drone during the simulation. These allow for the user to follow the drone in the simulation, both on the computer screen or on the HTC Vive headset, which can be controlled through the parameter options. The library also supports the Oculus Rift VR headset, making the library helpful since it is compatible with widely available VR headsets (Hellerer et al., 2014).

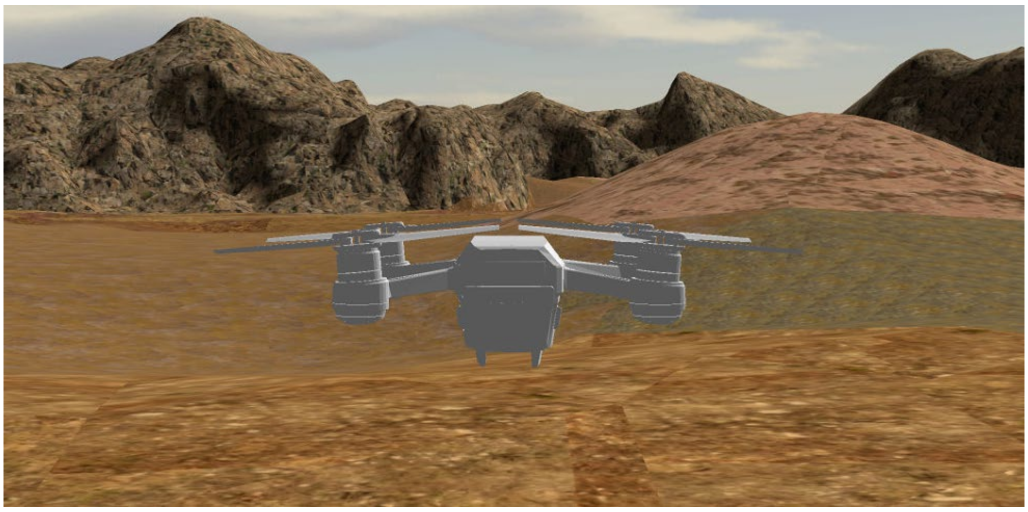


Figure 13: Drone in simulated terrain environmltent using the DLR Visualization Library

When the model is simulated, the drone appears in the Visualization Library's SimVis tool that interfaces with Dymola, shown in Figure 13. The drone can then be controlled by either the keyboard input or the joystick. Currently, the drone is not configured to have the propellers independently spinning from the body of the drone as it is one 3D component. The drone also does not have collision detection, so it can fly through the ground of the simulation terrain. Finally, to demonstrate the end-to-end user interaction, Figure 14 shows a picture of the user interacting with the model through the joystick.

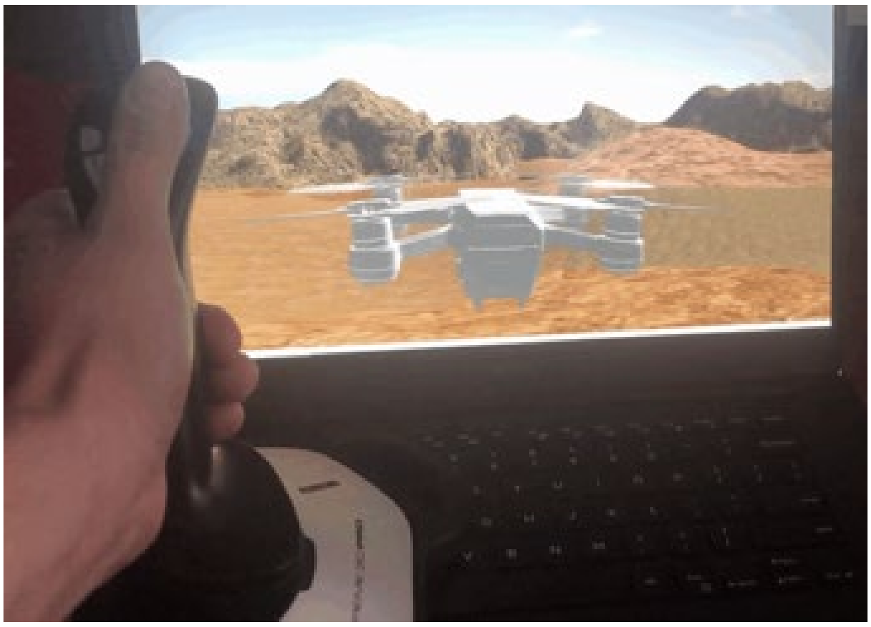
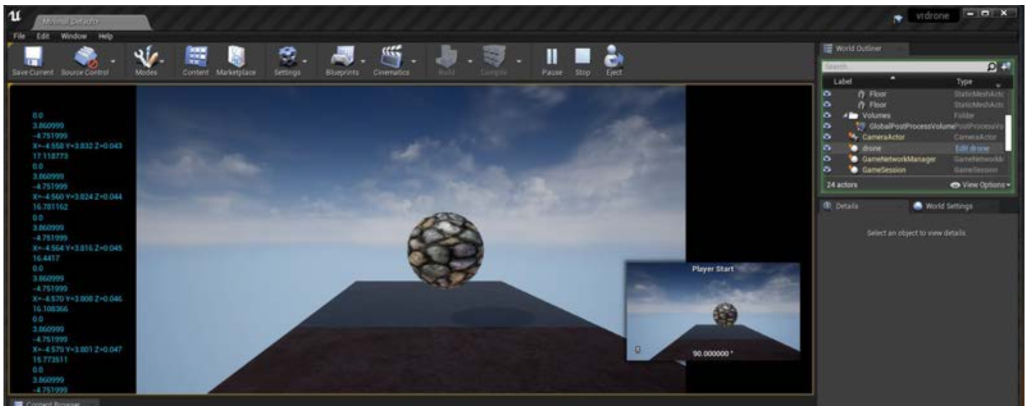


Figure 14: Drone model in simulated terrain environment using the DLR Visualization Library controlled by joystick input

c. Unreal Engine

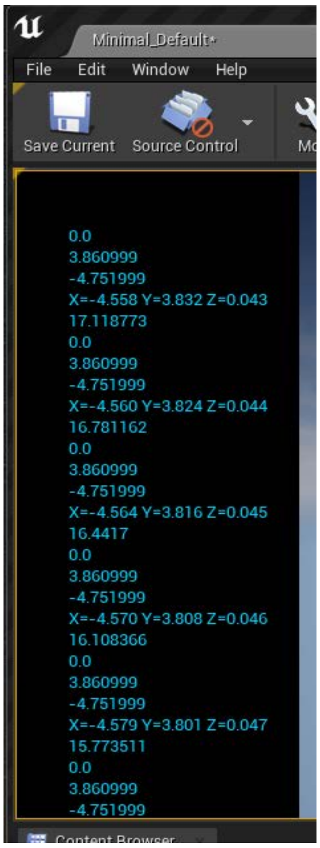
The drone can also be imported into gaming engines for simulation and interaction in addition to using Modelica-specific libraries. The drone is imported into the Unreal Engine, which is a platform for 3D game development (Epic Games, 2021). Just as it has been adopted in the film and television industry, the Unreal Engine can be used for 3D simulation and real-time interaction using C++ to develop environments and simulate models, however, that entails substantial efforts, which cannot be justified for initial design phases. As an alternative, in order to facilitate early design interaction, the model that was developed using Modelica can be imported into Unreal Engine using an open-source plug-in for FMUs (Greenwood, 2020). This provides a method such that the models developed and simulated using Modelica can be re-used in other tools, substantially reducing effort thanks to the underlying FMI standard.

As proof of concept, the drone FMU in the Unreal Engine is shown in Figure 15, where the drone is represented as a spherical object. Since the FMU transfers all the mathematical modeling attributes of the drone into the Unreal Engine, the visualization of the drone as a sphere does not impact the simulation. In Unreal Engine, the drone would be visualized using 3D CAD files, which will be completed and added in future work. Figure 16 shows the drone FMU's coordinates changing over time as the user commands XYZ coordinate changes using the keyboard.



```
0.0
3.860999
-4.751999
X=-4.558 Y=3.832 Z=0.043
17.118773
0.0
3.860999
-4.751999
X=-4.560 Y=3.824 Z=0.044
16.781162
0.0
3.860999
-4.751999
X=-4.564 Y=3.816 Z=0.045
16.4417
0.0
3.860999
-4.751999
X=-4.570 Y=3.808 Z=0.046
16.108366
0.0
3.860999
-4.751999
X=-4.570 Y=3.801 Z=0.047
15.773611
0.0
3.860999
-4.751999
```

Figure 15: Drone model in simulated terrain environment using the Unreal Engine controlled by keyboard input



```
0.0
3.860999
-4.751999
X=-4.558 Y=3.832 Z=0.043
17.118773
0.0
3.860999
-4.751999
X=-4.560 Y=3.824 Z=0.044
16.781162
0.0
3.860999
-4.751999
X=-4.564 Y=3.816 Z=0.045
16.4417
0.0
3.860999
-4.751999
X=-4.570 Y=3.808 Z=0.046
16.108366
0.0
3.860999
-4.751999
X=-4.579 Y=3.801 Z=0.047
15.773511
0.0
3.860999
-4.751999
```

Figure 16: Changes in drone model XYZ coordinates in the Unreal Engine controlled by keyboard input

d. Unity

The Unity game engine provides another option for VR simulation and interaction with the drone models by exploiting the FMI standard. Similar to the Unreal Engine, the drone can be imported into Unity using an FMU using an open-source FMI plug-in (CATIA Systems, 2018).

The drone is simulated and controlled in the Unity environment in Figures 17 and 18, providing another method to use the same model for virtual reality interaction and simulation with one model. The drone can be controlled using either keyboard commands in a XYZ plane orientation or a joystick gaming controller. The ‘W’ and ‘A’ keys control the X direction, the ‘S’ and ‘D’ keys control the Y direction, and the ‘O’ and ‘P’ keys control the drone in the Z direction. The joystick gaming controller has two joystick inputs on it, where one joystick controls the X and Y movement of the drone and the other controls the Z directional movements.

The biggest obstacle with this method of simulation was to attain consistent time synchronization between the game engine and the FMU. Since the FMU requires real-time simulation to properly control the drone due to the HIL inputs, the solver time in the FMU needs to be synchronized to prevent the model from reaching an unstable state. In the future, the visualization will be developed in more detail to include a richer model of the drone, e.g. separately controlling the propellers from the body.

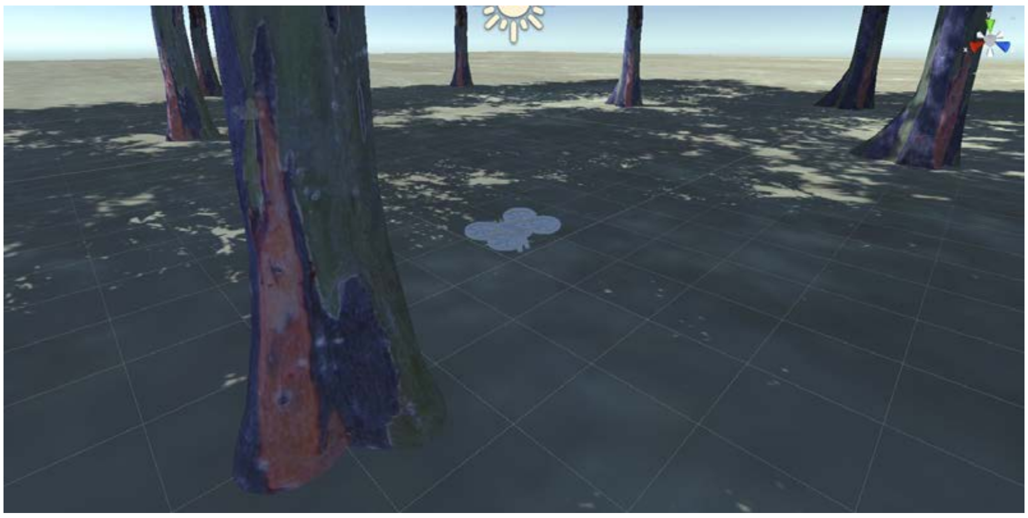


Figure 17: Drone model in simulated terrain environment using Unity with model imported using FMU

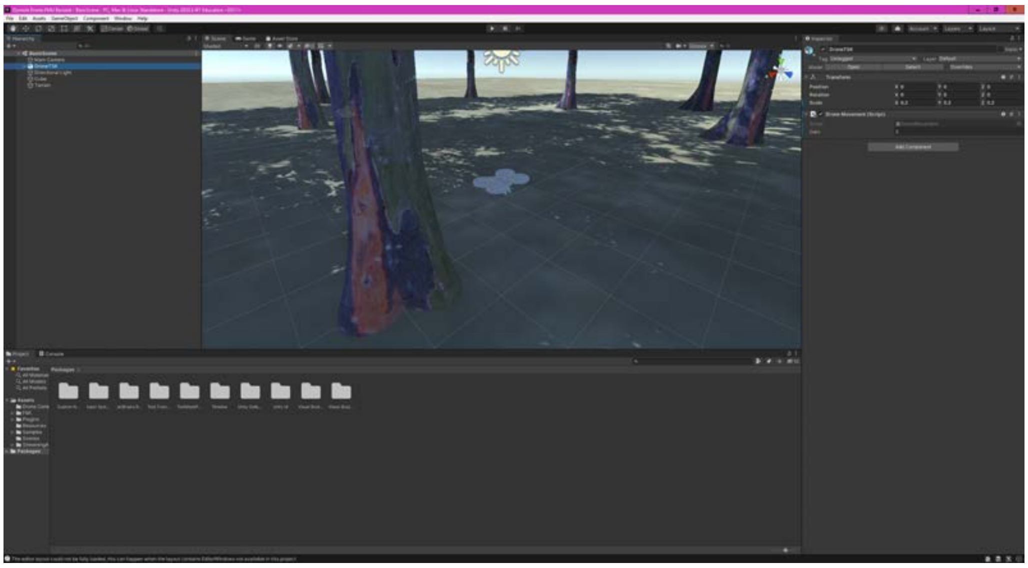


Figure 18: Drone model in simulated terrain environment using Unity with controls shown

7. Conclusions and Future Work

This work provides examples for different methods of simulation and visualization of a quadcopter in three different virtual reality environments. The same model is used for virtual reality interaction for all three methods because of the flexibility provided by the open access Modelica and FMI standards. In addition, this work lays down the proof-of-concept of how models can be integrated into VR environments at early design phases without substantial investments. This can be leveraged in the future to facilitate design methodologies where the designer and end-user gather feedback on requirements and specifications related to human interaction that are difficult to capture with today's

In the future, the visualization will be improved for all three methods. The drone model itself will be enhanced with more dynamism, i.e. the rotors should move separately from the body, so it is necessary to determine how to assign outputs from the FMU to control the 3D objects on the drone. It is also necessary to improve the drone visuals for the Unreal Engine example to represent the drone as it would look in real life. Other goals for improvement include collision detection and the corresponding physics for when the drone runs into an obstacle. The drone will also be simulated and interacted with multiple drones in one environment in future developments.

8. Acknowledgements

This work was supported in whole or in part by the National Aeronautics and Space Administration under award number 80NSSC19M0125 as part of the Center for High-Efficiency Electrical Technologies for Aircraft (CHEETA), by the Center of Excellence for NEOM Research at the King Abdullah University of Science and Technology under grant OSR-2019-CoE-NEOM-4178.12, and in part by The Boeing Company through its Charitable Partnership with Rensselaer Polytechnic Institute.

The first author is supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 1744655 and the Chateaubriand Fellowship of the Office for Science \& Technology of the Embassy of France in the United States.

9. References

Bellman, T. (2009). *Interactive Simulations and advanced Visualization with Modelica*. 10.3384/ecp09430056

CATIA Systems. (2018). *Unity FMI Add On*. Github. https://github.com/CATIA-Systems/Unity-FMI-Addon

Epic Games. (2021). *Unreal Engine*. Unreal Engine. https://www.unrealengine.com/en-US/

FMI Standard. (n.d.). *FMI Standard*. FMI Standard. https://fmi-standard.org/

Greenwood, S. (2020, November). *Unreal Engine - FMI Plugin*. Github. https://github.com/ORNL-Modelica/UnrealEngine-FMIPlugin

Hellerer, M., Bellman, T., & Schlegel, F. (2014). The DLR Visualization Library - Recent development and applications. *International Modelica Conference*, *10*, 899-911.

Martin-Villalba, C., Urquia, A., & Dormido, S. (2010). Development of Virtual Training Simulators with Modelica. *Proceedings of the 2010 Summer Computer Simulation Conference*, 413-418.

Modelica Association. (n.d.). *Modelica*. Modelica Modeling Language. https://modelica.org/

Otter, M., Elmqvist, H., & Mattsson, S. E. (2003). The New Modelica MultiBody Library.

Podlaski, M., Vanfretti, L., Nademi, H., & Chang, H. (2020). UAV Dynamics and Electric Power System Modeling and Visualization using Modelica and FMI. *Vertical Flight Society Annual Forum 76*.

Thiele, B., Beutlich, T., Waurich, V., Sjölund, M., & Bellmann, T. (2017). Towards a Standard-Conform, Platform-Generic and Feature-Rich Modelica Device Drivers Library. *International Modelica Conference*, 713-723. https://2017.international.conference.modelica.org/proceedings/html/submissions/ecp17132713_ThieleBeutlichWaurichSjolundBellmann.pdf