SSC23-P5-25

An Update on the Virtual Mission Control Room

Felix Sittner, Oliver Hartmann & Sergio Montenegro
JMU Würzburg, Department of Computer Science, Chair VIII: Aerospace Information Technology
Emil-Fischer-Straße 70, D-97074 Würzburg; +49-931-3188786
felix.sittner@uni-wuerzburg.de

Jan-Philipp Friese, Larissa Brübach & Marc Erich Latoschik
Human-Computer Interaction (HCI) Group, University of Würzburg
Am Hubland, D-97074 Würzburg
jan-philipp.friese@stud-mail.uni-wuerzburg.de

Carolin Wienrich
Psychology of Intelligent Interactive Systems, University of Würzburg
Oswald-Külpe-Weg 82, D-97074 Würzburg
carolin.wienrich@uni-wuerzburg.de

ABSTRACT

In 2021 we presented the Virtual Mission Control Room (VMCR) on the verge from fun educational project to testing ground for remote cooperative mission control. Since then, we successfully participated in ESA's 2022 campaign "New ideas to make XR a reality", which granted us additional funding to improve the VMCR software and conduct usability testing in cooperation with the chair of human-computer-interaction. In this paper and the corresponding poster session we give an update on the current state of the project, the new features and project structure. We explain the changes suggested by early test users and ESA to make operators feel more at home in the virtual environment. Subsequently, our project partners present their first suggestions for improvements to the VMCR as well as their plans for user testing. We conclude with lessons learned and and a look ahead into our plans for the future of the project.

The Virtual Mission Control Room started as a Virtual Reality (VR) scene implemented in Unity, enabling the user to look and move around within a futuristic mission control room featuring not-yet-interactive consoles, huge wall displays and a central floating globe orbited by satellites. The wall displays were browser windows, on which telemetry data could be displayed using web-based visualization tools like Grafana.[1] This early setup had few ways to exchange data and only required access to a web server and a simple script for passing TLEs for the globe in addition to the VR scene itself. While the first VMCR scene was designed for multiple operators, the multi-user functions were not added before the redesign.

The second iteration of the scene - which we described in our 2021 paper[2] - was initially a single-user design, which put the operator at the very center. The main screens and consoles were arranged in a semi-circle in front of the operator and the globe was moved from the center of the scene to a smaller room at the back. Multiple assets, such as the browser-based telemetry displays were reused from the prior implementation. While we were implementing the second scene, preparations for the InnoCube mission started, and we began working on a new ground station software. In the course of this, we redefined the objective of the VMCR project to create an environment that was - at least for the timespan of a satellite pass - suitable for real mission control. We implemented multiple web-based telemetry visualizations, and started development of a telecommand web frontend to integrate into the consoles in the virtual scene.

Since then, we have developed the VMCR into a multi-user environment and added the features and cooperation tools we will talk about in the next chapters. We are now in the process of improving usability, reducing VR sickness and fixing issues that our early test users brought to our attention. At the same time, we are preparing the first round of user tests with our colleagues from the XR Hub.[3]

VR SCENE DEVELOPMENT

The VR scene is what the operators experience when using the VMCR client software. They utilize their hand-held controllers to navigate and interact with special game objects which are part of the scene. The user's virtual hands emit a visible beam called *ray cast* that changes color when intersecting with an object the user can interact with.

As we explained scene creation in-depth in our previous paper,[2] we only provide a very brief recap covering the terms and basic concepts to facilitate a better understanding of the following sections.

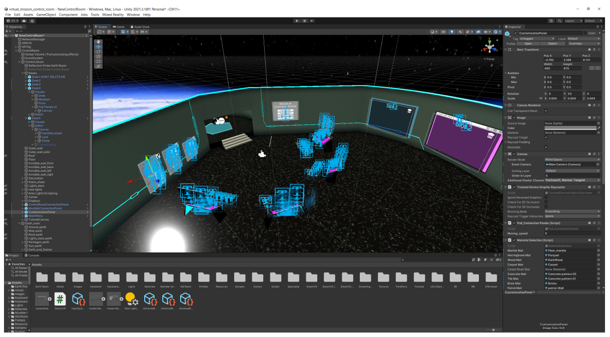


Figure 1: The second VMCR 3D scene in the Unity editor

A scene in Unity is built out of objects called *game objects*.[4] Game objects are hierarchical containers, containing *components*[5] such as a 3D model and a *transform* holding information about the objects position and orientation. Many parts of the scene are simple non-interactive game objects, such as walls, floor and other immovable objects. Some game objects, *interactables*, contain components which can be triggered by the user to perform actions. A *prefab* combines all components of a game object (child objects, scripts, settings) into a reusable asset. The *panels* (3D menus), VR keyboards, and consoles are examples of prefabs.

Several features of the scene, i.e. those connected to outside services, such as the voice chat, in-game web browser, and gesture tracking are built with the help of *plugins*.[6] Plugins contain game objects, scripts and pre-compiled executables providing access to system calls or third-party libraries. The XR Interaction Toolkit[7] provides a set of APIs, prefabs and scripts that can be used to extend a 3D scene into a VR scene.

The users do not need to install any additional software, since all required parts provided by the used plugins are included in the VMCR executable.

MULTI-USER FEATURES

We give a short summary of the cooperative tools implemented since our last publication:

Multiplayer

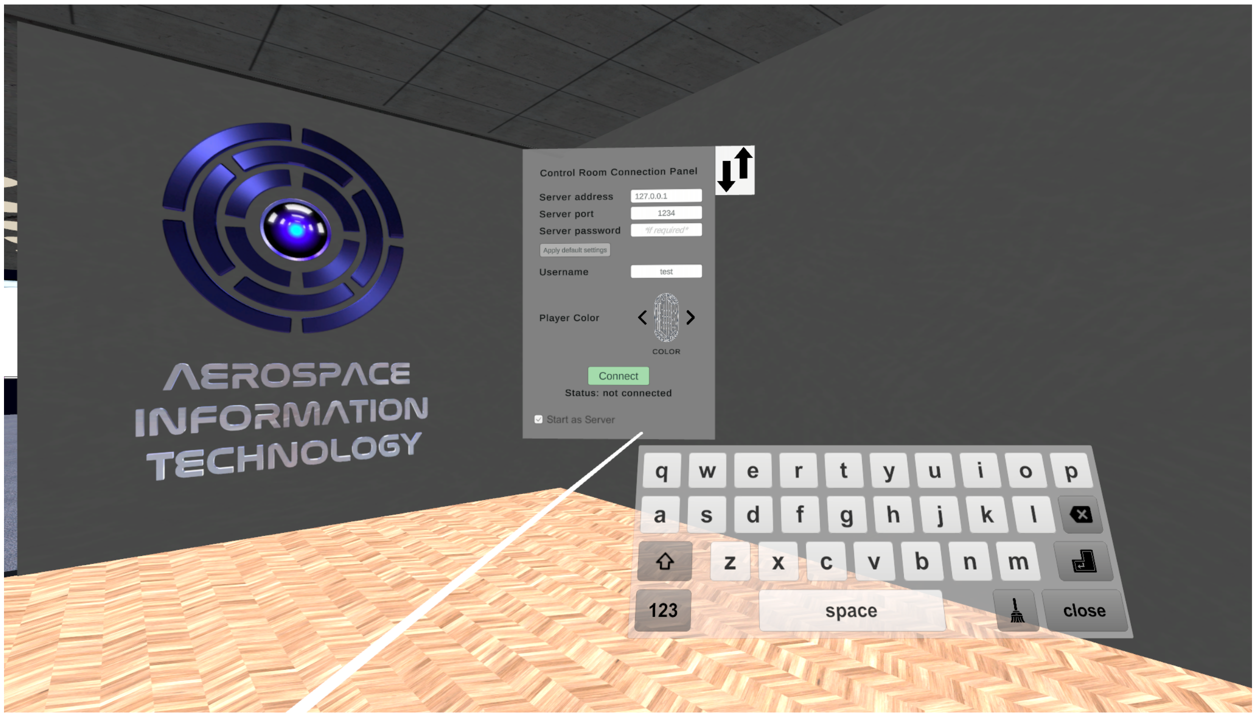
The first step towards creating a cooperative scene was of course to transform it into a multi-user environment (*multiplayer*). This feature was implemented as part of a student's master thesis, using Unity Netcode for GameObjects:[8]

They first added Netcode's *NetworkManager* component to the scene and created the *ConnectionPanel* user interface and implemented the main *ControlRoomConnectionManager* script. This script provides the connection settings based on the user input (server address, port, username, password), the amount and kind of network objects and configured bandwidth limitations. It also reacts on user input and triggers connection setup and termination. Subsequently, they added basic player avatars and attached Netcode's *Client-NetworkTransform* scripts to synchronize their positions over the network so that operators can see representations of each other in the shared virtual scene.

Usage Each VMCR executable can function as client or local server for testing purposes. In normal operation the server is only run on the main back end server and operators connect as clients, using the default connection settings configured by their VMCR administrators.

Each operator is initially placed in their own private room and can then connect to the server to join the multi-user VMCR scene using the connection panel depicted in Figure 2. They can enter the connection data using a VR keyboard, which is set up to float in front of the user avatar upon selecting any text input field. As it was rather tedious to input the required data each time a connection was made, they added an option to automatically populate the fields with predefined values. The panel was created as a reusable asset and is usually attached to a wall in the virtual scene. All panels can be pulled closer by clicking on a button or activating a special *interactable*.

Operators can activate a virtual laser pointer: while the *thumbstick* button is held, the emitted *ray cast* of that controller - normally only seen by the user - is also made visible to other operators



Control Room Connection Panel

Server address 127.0.0.1
Server port 1234
Server password *if required*
Username test
Player Color
Status: not connected
☑ Start as Server

Figure 2: Connection panel and keyboard floating in front of the test user ray cast

Security Due to other plugins and dependencies, our student was working in a version of the Unity editor not compatible with Unity Transport[9] version 2.0. Hence, they could not easily implement acceptable encryption using the Netcode version available to them. Hence we postponed this part and the first multiplayer implementation transferred data unencrypted relying on the security (and usage) of an external VPN connection. Since we upgraded the project in the meantime, this issue is currently being addressed.

In addition to adding multiplayer functionality, our student also implemented the shared browser views and voice chat integration which are described in the following sections.

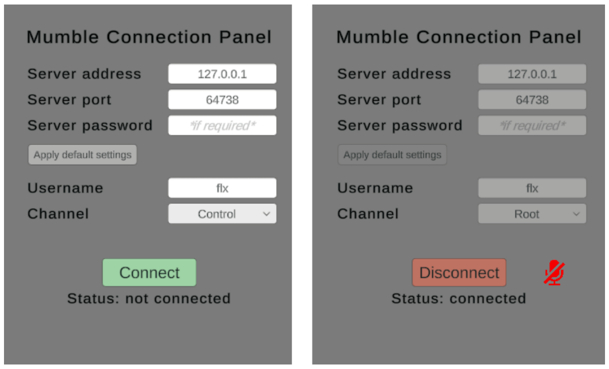
Voice Chat

The first collaboration tool we added to the multi-user scene was voice chat. We assessed several voice chat plugins, with the prerogative to use a free and open source software solution if possible. In addition, we were looking for a solution that is free to use and does not require subscription or monthly payments.

However, our most important requirement was based on data security: we deemed it essential that the employed protocol offers up-to-date encryption and the option to use a self-hosted server, with no unencrypted data relayed to any other, especially no non-european, backend infrastructure. These requirements ruled out using most popular voice chat plugins featured in the Unity Asset Store, such as *Photon*, *Agora* and *Vivox*.

Due to our criteria, we decided on using the Mumble[10] protocol and Murmur server together with the Unity plugin code the open source Mumble-Unity[11] project provides. The plugin provided an example script our student built upon and a code base they expanded and adapted to fix some minor issues. They created the user interface panel depicted in Figure 3 and corresponding scripts tapping into the plugin functions, in order to allow users to control the mumble client software made accessible by the plugin.

Users can enter the connection data using a VR keyboard. Like for the control panel, it is set up to float in front of the user avatar upon selecting any input field. They can also use a button to automatically fill all input fields with the preset default values. Users can connect, disconnect and switch channels using this panel. In addition they can mute and unmute their microphone when connected, as can be seen in Figure 3. The panel was created as a reusable asset and is also usually attached to a wall in the virtual scene.


Mumble Connection Panel
Server address 127.0.0.1
Server port 64738
Server password *if required*
Apply default settings
Username flx
Channel Control
Connect
Status: not connected

Mumble Connection Panel
Server address 127.0.0.1
Server port 64738
Server password *if required*
Apply default settings
Username flx
Channel Root
Disconnect
Status: connected

Figure 3: Voice chat user interface panel before and during connection (muted)

It is important to note that participants can join the voice chat either from the multi-user scene, their personal VR scene or their real-world working environment; experts consulted via voice chat are not obliged to join the VMCR.

The integration of the Murmur server and other server-side services into our backend architecture is explained in the backend section of this paper.

Shared Browser Views

All operators should see the same information on the main wall displays at the front of the virtual room. However, the displays render web page data.

The problem: If an externally hosted web page is accessed, e.g. to look for information on a topic, the users might be served data depending on their web browser settings or the location they access the site from. Even if operators scroll through Grafana telemetry dashboards provided by our own servers, they see different data depending on their interaction with the page. But in order to cooperate and discuss, they need to see the same data as they would if the were in the same physical control room. Operators should be able to point out details and anomalies, interesting things or share a web-based whiteboard solution.

To facilitate this, the synchronized displays are implemented as *network objects.* Network objects are game objects that are synchronized over the network and have owners. When the user who has ownership of a display calls a website, this triggers a remote procedure call on the server. The website content is always fetched by the back end server and distributed to all connected clients. Users can forfeit ownership of a display, which allows other users to claim it and control the browser front end.

Implementation: Presently, the web page is fetched by the back end browser, rendered as a texture and transmitted to all clients. Hence, user interactions must be recorded, sent to the back end and replayed there.

To mitigate the bandwidth problem at hand, all textures are compressed before transmission and the page content is synchronized at most once per 0.6 seconds. If however the maximum acceptable data rate is exceeded the transmission will be stopped and a warning message displayed. Therefore, the present solution is obviously not suitable for video streaming or very feature-laden web apps.

The operators can then open the respective web page or web app in a non-synchronized browser instance on their consoles or another wall display. If a user decides to stop the synchronization, a warning message is shown next to the display to indicate the display's contents might differ from the other instances'.

Text Chat

No multi-user environment is complete without a text chat, so we implemented this basic feature as well. After connecting to the cooperative environment, users can send and receive messages in a simple text chat window. When a user tries to send a message without being connected to the main control room, a warning is displayed as an overlay over the chat. The chat can be attached to a wall or displayed on a console screen. We are going to add more features, such as enabling the user to make the chat *interactable* follow them trough the scene.

SCENE CUSTOMIZATION

One main advantage of a virtual control room is that it can be adapted quickly and with almost no additional cost, to match an individual operator's needs and preferences. The scene customization was implemented as part of a student bachelor thesis and is currently extended by our staff.

How it Works

A customization overlay is part of every complex *customizable* game object, i.e. the workstations. This overlay is normally invisible and and non-interactable. If the operator activated customization via the *Customization Control Panel*, a script component of the customizable game object activates the overlays when the user moves near it. An example can be seen in Figure 4: a POV picture taken directly after the user rotated the desk.

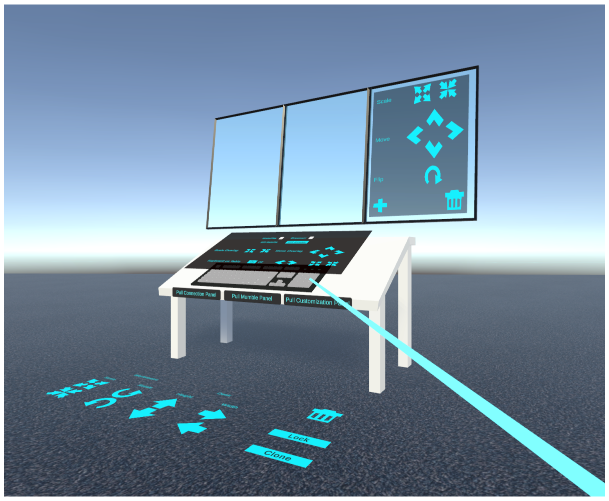


Figure 4: Desk prefab with customization overlays triggered by user proximity

When the user activates an element of the customization overlay, a script is called that changes properties the game object: for example, clicking on any *resize* menu will adapt values of the *transform* component the selected game object. In addition, the customizable elements can be picked up, moved and set down. This also changes the values of their *transform* component.

In Unity a *prefab* of every object that can be instantiated in-game to be "added" to the scene must already be part of the scene at compile time. Our students had to work around this, rather amusingly, by hiding various prefabs under the floor out of sight and reach of the users.

Customization Examples

After activating scene customization, users can add, delete and clone different workstations. They can configure the position, width and height of a workstation, and also configure the child objects: There are overlays enabling the user to add, delete and flip and re-size displays as well as arrange them within a plane. Several child-components of the workstations can be set active or inactive, examples are keyboard, browser windows, chat window and virtual camera feeds. This way the monitors and workstation table top can be assigned different functions. When a user is finished configuring a workstation, they can lock it to prevent accidentally moving it. Wall displays can be added, deleted and arranged within a slot system.

For various simple static objects, such as ceiling, walls and floor only textures can be changed and some attached decorative elements switched on or off. These settings can be adapted via the Customization Panel.

The current proof-of-concept implementation of customization menu can easily expanded by adding new workstation prefabs as well as span buttons and prefabs for other customizable objects.

Configuration Saving and Loading

Loading and saving of the configured scenes is rather complicated in Unity, as - unlike some other game engines - Unity provides no simple way to save the whole changed scene from within a running game. We implemented functions to store the configurable attribute values for every configurable component. When a user saves the scene, a script is called that inspects all game objects in the scene and serializes the state of each object with the "configurable" attribute to a special save file. When loading a configuration, this file is read, the existing configurable objects are destroyed, new objects are created and adapted based the loaded data.

The Question at Hand

The users' ability to configure their scenes makes it necessary to decide what happens when the operator joins the multi-user environment. Obviously, each user should see the same scene (notwithstanding other operator's personal workstation monitor content) when working together within a shared virtual environment.

We discussed several approaches to prevent uncanny experiences: One option would be that each operator can configure their private (local) scene and the shared scene is either preconfigured or can only be adapted by an administrator. Operators would then start in their personal room and enter an (from their point of view) immutable shared scene after connecting to the VMCR *multiplayer*. This could be made less uncanny by the avatar "stepping" trough a door into the shared scene. While this would be the simplest option to implement, we agreed that it would take too much flexibility out of the VMCR.

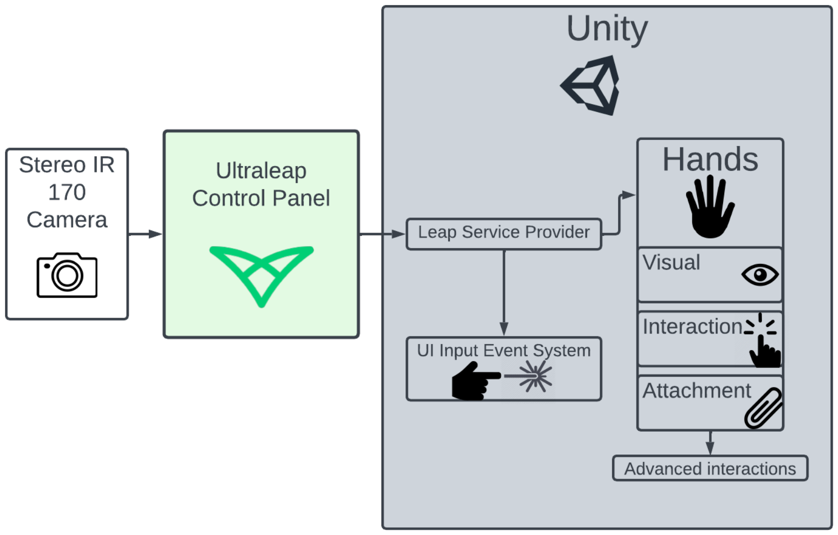


Figure 5: Structure of the Ultraleap integration in the VMCR

We prefer to create a solution where every operator is able configure a part of the central scene assigned to them. This can be done in two different ways: Either the users configure their workspace before connecting and changes are synchronized when they connect to the *multiplayer*. This would then, upon joining, trigger a sudden update of the respective user's area for all other operators, but they would not be distracted by a user "redecorating". Or the users could be to allowed to change their console setup while connected, so changes to the shared scene are introduced one at a time. In this case, we would need add an option to mask the area to prevent distraction.

If time permits, we will enable the operators to configure a local workspace and their part of the shared VMCR. As discussions are still ongoing we have not finally decided on the details of shared workspace configuration.

GESTURE BASED INTERACTION

As working in a virtual environment such as the Virtual Mission Control Room means staying in virtual reality for extended periods of time, it is important to make interactions with this virtual workspace as comfortable and intuitive as possible. One major obstacle for this are the VR controllers used in combination with most common HMDs. They are not precise and rather slow for common work related tasks such as typing on keyboards, and holding them for extended periods can put strain on users hands and wrists. To alleviate these issues we implemented hand tracking controls for the Virtual Control Room Scene by using an Ultraleap Stereo IR170 camera[12] and the Ultraleap Unity plugin[13] software.

Ultraleap

Ultraleap devices are capable of tracking hand movements by first illuminating an area in front of them in infrared light, and then searching for objects they recognize as hands in the highlighted area with their two cameras. The gathered tracking information from the Ultraleap control panel software can then be used via an API or plugins for games engines such as Unity.

Implementation

With this implementation, we focus on two key features along with making the scene interactable via hand tracking in general. The first is a new and improved virtual keyboard, while the second is controller-less locomotion.

Design Considerations & Guidelines

Along with their plugins, Ultraleap offers design guidelines for hand tracking applications.[14] These deal with both the technical limitations of the hand tracking devices and how to work with them, as well as user comfort and ease of use. We specifically focus on the guidelines regarding the classification of hand tracked interactions in two main groups, as well as the occlusion of hands.

Types of Interactions According to Ultraleap, all hand tracked actions can be categorized as either *physical* or *learned* interactions. Physical interactions mimic actions that would be possible with the users real hands outside of a virtual space, like touching and grabbing objects, and are thus intuitive and easy to understand. Learned interactions on the other hand may offer more possibilities, but also require training. Because of this, physical interactions should always be chosen over learned ones, if they are an option. In this project we additionally lock most learned interactions to only be available while the user is raising their left hand so that the palm is facing the camera, in order to avoid accidentally triggering them.[14]

Occlusion The hand tracking software can compensate for a partially occluded hand, such as covered fingers by predicting their position based on the visible part of the hand. Since this can however result in incorrect hand poses and errors, situations that may result in occlusion should be avoided when designing interactive elements.

Basic Hand Tracking

Basic hand tracking features and scene interactions can easily be achieved by using components included in the Ultraleap Unity plugin,[13] the most important of which is the *LeapServiceProvider*. When attached to the player rig it acts as the connection between the Ultraleap Control Panel and Unity, so that objects in the engine have access to the tracking data. Using this data, we then add three separate components to build up the features of the hands. *Visual Hands* visualize the tracked hands in the scene, *Interaction Hands* allow them to interact with it, and *Attachment Hands* allow us to connect additional game objects to them. In addition, the *UI Input Event System component* allows for interaction with distant objects by pointing at them with a raised hand, and pinching thumb and index finger. As it can be difficult to spot this ranged cursor over longer distances, we added an optional visible ray.

Dynamic Keyboard

We first replaced the previously used *CanvasKeyboard*[15] with the *XR Keyboard*[16] provided by Ultraleap. This VR keyboard asset was specifically designed for use with hand tracking and features raised buttons that react to nearby hands and button presses.

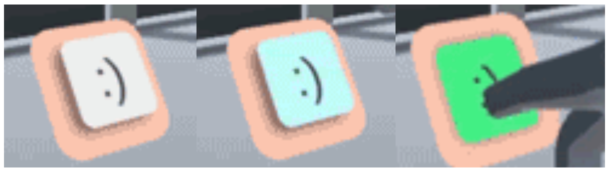


Figure 6: Ultraleap keyboard buttons reacting to tracked hands in proximity, and direct touch

To then change the behavior to give the user full control over the keyboard with its activity status and position, we combined multiple components of the Unity plugin. The script pair *Anchorable Behavior* and *Anchor* allow for attaching an additional object to a tracked hand, while also being able to grab the object and place it in any spot within the scene. Combined with the *Workstation Behavior*, which will activate a new object in the scene whenever such an *anchorable* object is dropped in the scene, we can then activate and move the keyboard at will.

Subsequently, we added two more quality of life features: first, a button to recenter the keyboard in front of the user to the palm of the left hand. And second, we added the option to move the keyboard by dragging an anchored object into the scene; which works even when there is already an active keyboard.

Lastly, we tried to reduce occlusion issues, to which virtual keyboards are in general more prone to, since the fingers are often covered by the rest of the hand from the perspective of the camera while typing. We therefore set the height of the automatic keyboard reposition low enough to counteract this as much as possible, and add an invisible volume around the keyboard to disable the ranged interactions for the tracked hands while they are being used for typing.

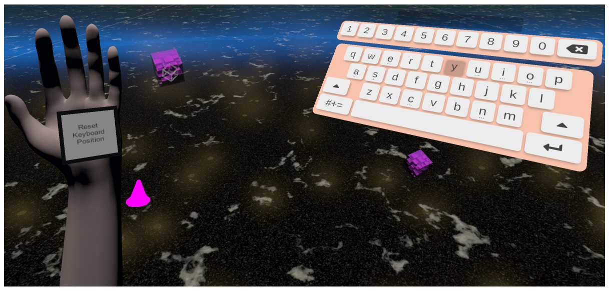


Figure 7: Dynamic keyboard with cube *anchorable* object next to hand representation with recenter button

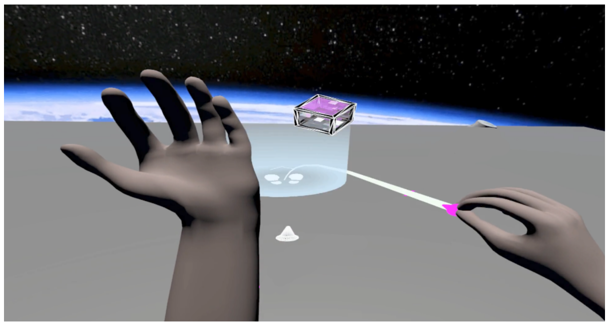


Figure 8: Grabbed *Jump Gem* projecting ray to teleportation target area

Locomotion

The implemented locomotion is based on Ultraleap's *Jump Gem* approach of having a hovering object next to the tracked hands, that can be grabbed to project a ray marking the desired location when it intersects with the floor. After aiming with the ray, letting go of the object will trigger a teleportation to the highlighted location. While this approach is already functional, it lacks rotational controls when compared to the locomotion options offered by VR controllers. These can be implemented in a variety of ways, such as simple buttons that trigger a snap rotation attached, simple gesture recognition for pointing in a certain direction, or by adding a virtual input device in front of the user, that can be operated to more directly replace the VR controller.

NEW PROJECT STRUCTURE

Since 2021, we have added several features that rely on software run outside of unity, such as voice chat and satellite emulation. Those services had to be set up and configured manually, which was acceptable for a test setup. But with prospective users from other facilities we needed to structure and document the project in a more professional way.

Client-Server-Setup

The basic setup of the VMCR, as depicted in Figure 9, is split into the standalone VMCR Application run at each operator's computer and the applications run on the central back end server. The back end server hosts the master VR scene to which operators can connect as well as the services supporting features of the scene.

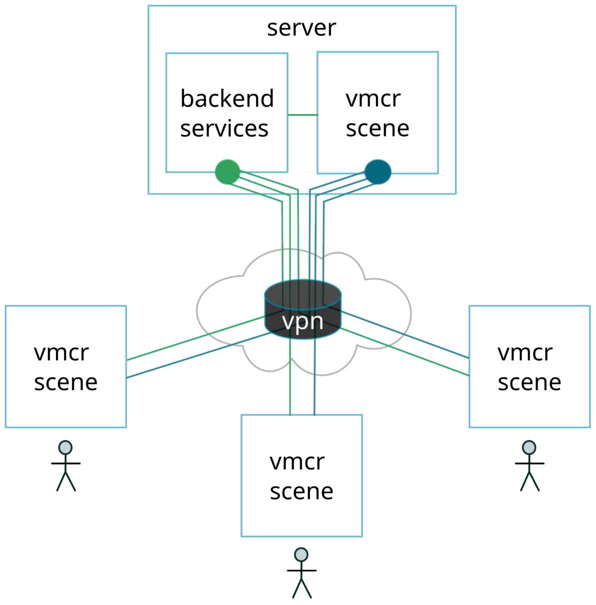


Figure 9: User (client) scenes connected to the VMCR back end over a VPN

Division into Sub-Projects

One major drawback of having everything within one big project was, that everybody had to download the whole virtual scene, even if they were only improving some details of the back end setup. We divided the VMCR into several git projects organized in a group, to enable developers and testers to only check out the parts of the project they intend to work with.

In the following section, we provide a short overview of the components the project is made of and which functions they add to the VMCR. The VR scene is what the operators directly experience when putting on their head mounted displays (HMDs) and start their VMCR client software. The back end services are everything apart from the VMCR master Unity scene, that runs on the VMCR server.

Back End Design

We redesigned the back end as *microservice* architecture so prospective users can switch out parts to adapt the project as needed: We choose to provide each back end service, except for the main-unity-scene, within its own Docker[17] container. These containers are linked together within in a virtual network to interact and facilitate the VMCR.

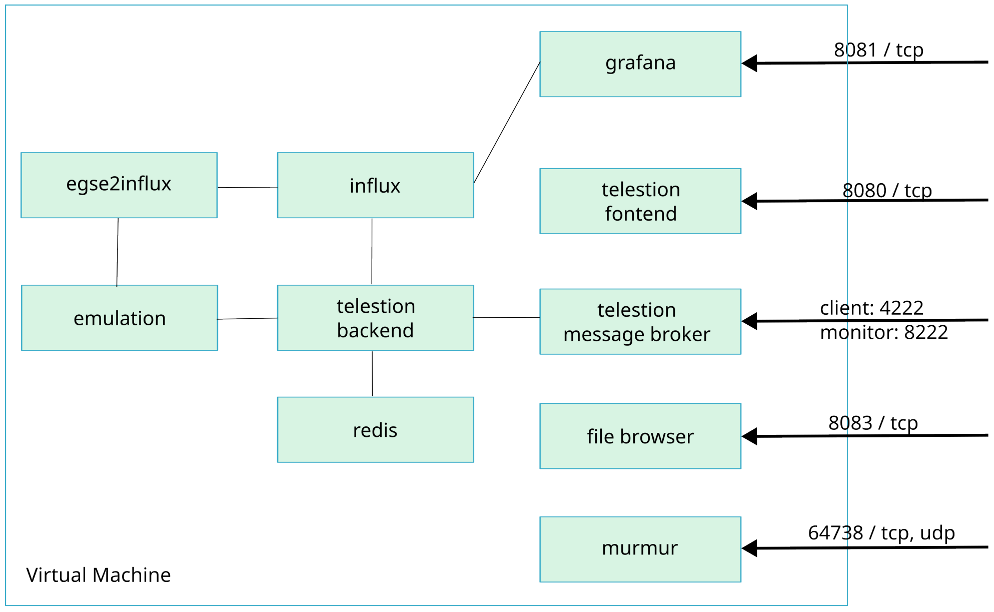


Figure 10: VMCR back end service containers within a virtual machine

Developers and administrators are able to add their own back end services by adapting the docker compose file to include their own containerized applications. In that way, a university or agency can plug in their own satellite simulation data or web-based ground station graphical user interface (GUI).

In addition, we prepared a pre-configured virtual machine (VM) for our test users who need to deploy the back end on a windows server, without the option to run docker-compose within the Windows Subsystem for Linux (WSL).

Figure 10 depicts one of multiple possible VMCR back end setups: this VM is configured for training purposes, with a locally run instance of our CORFU[18]-based on-board software and control front end provided by the Telestion[19] ground station software developed by WueSpace.[20]

The (satellite) emulation container is where our RODOS-based on-board software is deployed and run. This container is connected to two other back end containers: The egse2influx container holds a very minimal CORFU[18]-based ground station software, that receives telemetry and stores the data in the InfluxDB[21] database located in another container. The Telestion back end container hosts an instance of the Telestion ground station back end with a special extension that allows it to decode CORFU messages. It is connected to the front end / message broker service and also to two database containers, the InfluxDB instance and a Redis[22] database for long-term storage.

The containers visible on the right side of the figure host the services the VR scene plugins connect to: All of these services, except for the murmur server depicted at the bottom, are accessed with the help of the web browser plugin. The Grafana[1] container serves the telemetry visualizations to be shown on the huge wall displays in the front of the VMCR scene. The Telestion front end provides user authentication and sets up the message broker session between the web-app run in the operator's browser and the Telestion back end. The file browser[23] back end facilitates an easy way to exchange files, e.g. to share documentation, from within the scene. Lastly, the Murmur server provides the back end for the Mumble-based voice chat.

For real missions the emulation container is replaced by a container that forwards data through a secured channel from and to the (UHF) ground station radio control server.

Documentation

All documentation was moved into the *Documentation* subproject and made accessible within a wiki. This wiki is structured in three sections providing documentation tailored towards administrators, developers and users.

The **admin manual** focuses on set-up and maintenance of the server side VMCR applications, and intersects partly with the back end developers' manual pages.

The **developer manual** provides information and how-tos to help developers adapt and extend the VMCR. Like the VMCR itself, it is structured in two main parts: the VR-focused sections explain how to adapt, extend and compile the VMCR scenes using Unity3D and working with plugins, assets, prefabs and a C#-IDE. The remaining sections teach developers how to fork and adapt the back end and provide links to the documentation of several sub projects, such as the CORFU-based satellite simulation and the Telestion ground station software.

The **user manual** covers everything from setting up the VR headset over menu-interaction guides to how to adjust one's personal VMCR scene in-game.

EARLY USER FEEDBACK

We presented the second iteration of the VMCR to several test users as well as our ESA contacts. These are the problems we found and addressed during development, prior to the official, structured user testing phase.

Control Room Scene

Among the first responses to the VMCR was, that while the futuristic scene was an interesting concept, our prospective users envisioned a bigger, more conventional scene with a stronger resemblance to their real mission control rooms.

As a first measure, we created a more conventional scene, of which the floor plan can be seen in Figure 11. In this scene, the operator avatar spawns in a vestibule, depicted in the right side area of the floor plan. They can then either connect to the chats and multiplayer using the control panels and "walk" into the more spacious room or enter a tutorial.

All components and functions developed for the futuristic scene are also available within the conventional VMCR, as the same scripts, assets and prefabs created by our students were used to build it.

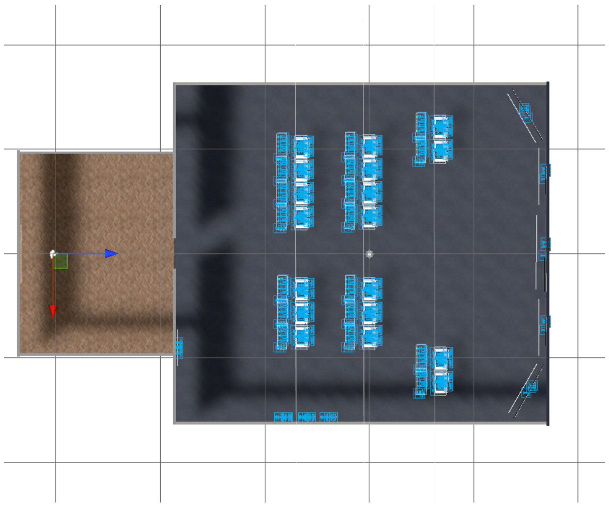


Figure 11: Conventional scene floor plan

This scene will be adapted further according to user input in the course of this project and other designs are already in the making. The scenes approved by ESA will then be included in the final delivery as part of the unity-scenes subproject.

Tutorial

Several users were overwhelmed when directly entering the VMCR without any introduction to the various ways to interact in there. Even more so, after we added some tutorial elements to the main scene. To address this issue, we created a separate simple tutorial scene, which new operators can experience before they enter the VMCR.

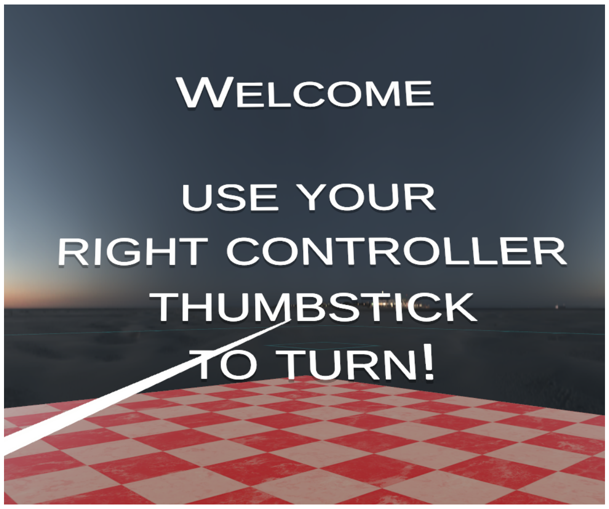


Figure 12: A new user's POV from the tutorial scene prototype

At the very beginning users are shown and practice basic movement, later on they learn to enter data into fields and adapt their environment. Users can at any time quit the tutorial and transition to the main VMCR scene using the *Scene Loading Menu* panel. Our tutorial scene is still very basic and will be improved as part of a student thesis within the year.

Shared Whiteboard

In addition, the ESA operators were concerned about the shared whiteboard being implemented as an in-browser-solution, with which they had mixed experiences in the past. Hence, we are reimplementing the shared whiteboard as a Netcode[8]-based Unity-native solution.

VPN Service

We initially planned to ship an open-source VPN solution as part of the VMCR back end. But as most facilities are bound to use a predetermined VPN technology with specific settings and/or provider-specific solution, we do not include that anymore.

Panel Design

Another (minor) issue mentioned was that most panels and menus while functioning, are basically out-of-the-unity-tutorial-designs and not yet styled in an appealing manner. In addition, ESA remarked that for the voice-chat control, they would prefer an *asset* modeled to resemble the table top voice loop control panel used at the ESA mission control.

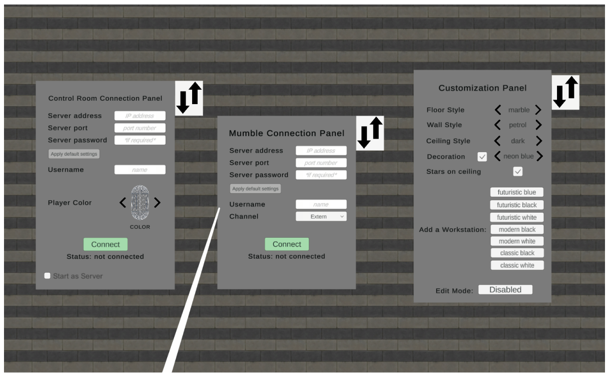


Figure 13: Connection, mumble and customization panel prototypes

We are currently working on improving the design and usability of the VMCR with the help of our colleagues from the XR Hub[3] located at our University during the current phase of the project. They present their plans and findings in the following chapter.

IMPROVING USABILITY

VR Application

The following section provides an overview about minor usability issues we found in the VR application and additional design suggestions how the ground station software could be improved in VR.

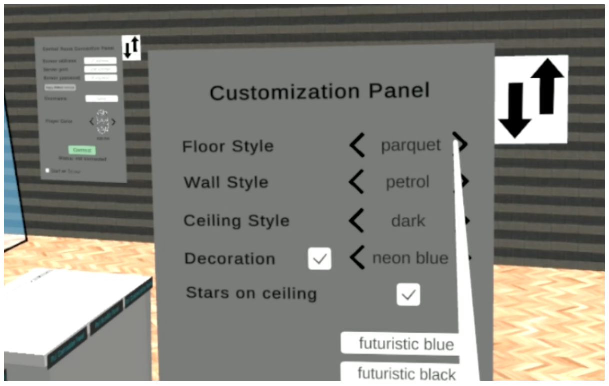


Figure 14: No highlighting of selection

When the user iterates through different options of menu items, the interaction symbol (e.g. an arrow) was not highlighted in some cases (Figure 14). The user sees the ray of the controller but cannot be sure whether the interaction surface is really targeted. To solve this issue, we will highlight the item in a color which has a high contrast compared to the colors of the rest of the menu. The low contrast of some selected menu items also presented a significant usability issue since the user is not able to distinguish between the selection and the rest of the menu. The same solution as explained before should be applied here as well.

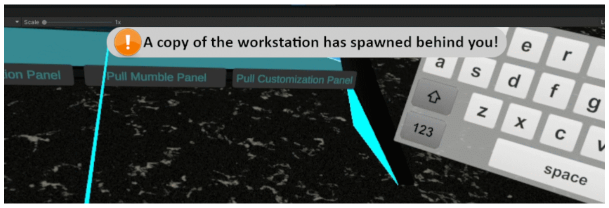
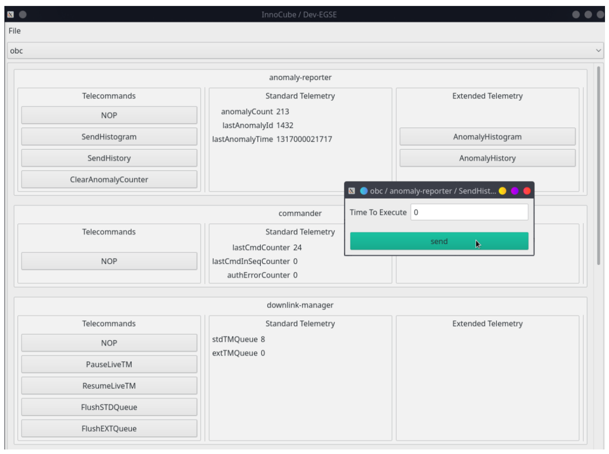

A copy of the workstation has spawned behind you!

Figure 15: Proposed HUD notification

During the creation of new control workstations, the new workstation is placed at the main spawn point of the virtual environment. If the user is not located there or the gaze direction is different to the spawn location, he will be not informed that a new workstation has been generated. This can be avoided by adding a HUD message which gives the user a hint (Figure 15). Alternatively, a flashing direction arrow could be inserted on the edge of the screen indicating where the workstation has spawned. This arrow should disappear when the user turns to face the workstation. Lastly, a few small bugs will be fixed that could lead to motion sickness.

Ground Station Software User Interface

Figure 16 depicts the standard ground station software developed for the InnoCube mission. A web application of similar design was later developed to be accessed from the console screens in the VMCR.



```
anomalyCount 213
lastAnomalyId 1432
lastAnomalyTime 1317000021717

Time To Execute 0

lastCmdCounter 24
lastCmdInSeqCounter 0
authErrorCounter 0

stdTMQueue 8
extTMQueue 0
```

Figure 16: CORFU-generated ground station GUI with a pop-up telecommand window

The on-board software applications are listed horizontally, telecommands can be issued by clicking the named buttons on the right side, the standard telemetry is shown as text in the middle and the extended telemetry for each app can be opened by clicking the buttons on the right.

Due to the long item- and data lists of the ground station software, we first suggest splitting up the standard telemetry data from the main window because they are mandatory for the usage of the application and have to be visible at any time for the user. Subsequently we can group the different functions of the software and create new sub-menus. Each sub-menu contains one data point where users can send the associated commands and receive extended telemetry data from the satellite. The standard telemetry of the data point is separated in an extra window as previously described.

Regarding the telecommand input, it would be beneficial to only display a number pad instead of a whole keyboard when a user selects an input field only accepting number values. This has the advantage that the individual fields of the input field can be designed larger and thus increases readability and making them easier to hit with the ray cast.

We also discussed whether a more graphical design such as a vector representation of the data would be beneficial. This suggestion was discarded because users are already familiar with the representation of satellite displays and the design idea would take up too much space in the standard telemetry display.

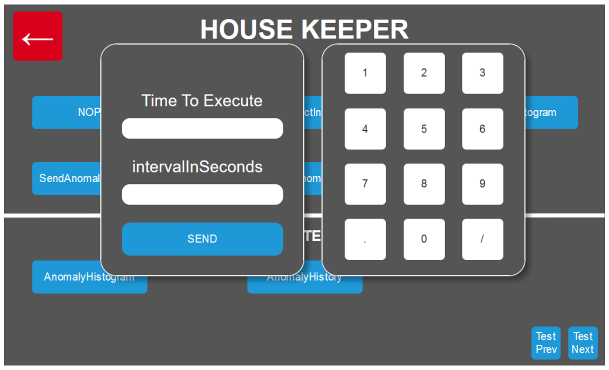


Figure 17: VR telecommand input overlay

Interaction Technologies

To control the software mentioned above, we discussed different technologies which could be potentially relevant for interacting with our virtual environment. One crucial factor is user movement in VR. The goal is to get a good mixture of efficient and precise freedom movement in the mission control room. By implementing multiple travel techniques like walking-based- and selection-based travel, the user can precisely adjust his position with the joysticks of the motion controller (walking-based). In addition, the possibility to teleport by pointing to a specific location provides far distance navigation. The combination of both could result in an efficient solution for navigating through the control room.

Another key consideration is the ergonomic control of user inputs. Aside from communication over voice, the primary interactions with the application involve sending commands and receiving telemetry data. Users employ a keyboard to insert data for the commands. One type of control devices is the motion controller of the VR setup. By utilizing ray selection, users can select the different letters by pointing with the physical motion controller to the associated key on the keyboard and pressing a button on the motion controller to select that key.

A second type of control device is the UltraLeap Hand Tracking System where the user's physical hands are tracked by infrared light to type on a keyboard. The advantage of the UltraLeap system is an intuitive way of interaction, but the user has no haptic but only visual feedback. A long-time usage could be also exhausting because the keyboard isn't located on a surface where the user can put down his hands on a table. Instead, inputs need to be made in mid-air, which may become fatiguing over prolonged usage.

A third interaction technique could be a physical keyboard equipped with markers so that the VR application can track the keyboard. Although the physical keyboard would reduce the effort for typing compared to typing in the air with the UltraLeap technology, a permanent surface is still required. The main advantages here are the haptic feedback and ergonomic typing in case of a support surface. Further user tests will provide more insights into the individual technologies.

User Testing

To evaluate our changes to the user interface and the discussion about the interaction techniques, it is mandatory to test our findings with real users. Therefore, it is planned to conduct a user study where students of the chair of aerospace information technology are traversing a satellite flyover in the virtual mission control room. The study focuses the main interaction with the ground control software and will evaluate the usability of the application's functions as well as other notable measures such as motion sickness or stress induction. We will probably use questionnaires like the Simulator Sickness Questionnaire[24] for motion sickness, the NASA-TLX Questionnaire[25] for workload analysis and PASA Questionnaire[26] for stress induction. This satellite flyover will take around 20 - 30 minutes for each participant. In addition to the VR application test, we will evaluate the revised VR-interface of the ground software with a pluralistic walkthrough and an expert review.

The pluralistic walkthrough is a corporate meeting of target users, developers and usability experts where the user accomplishes a task and occurring usability issues are discussed afterward with all participants. Through this collaborative analysis, the developers gain direct access to new design ideas which are developed in conjunction with the usability experts and the users.[27]

To gather more ideas how we could improve our application design, a contextual inquiry (contextual design process) would provide more information about the working structure, communication and detailed working processes during a satellite flyover.

A contextual inquiry consists of two parts. The first one is a traditional interview where the researchers gain an overview about the relation of the user's life to the target activity and information about tools that are used during the activity. The second part is the main inquiry and it is more an observation and discussion than a regular interview. The emphasis lies on the user's actions during the tasks and the artifacts employed, with the researcher interpreting these observations and engaging the user in discussions to obtain their interpretations. For instance, the user should be asked after a phone call about the content of the call because this could be essential for subsequent analysis, understanding user needs and generating design concepts.[28]

We are preparing to use both of these user centered methods in our upcoming meeting with employees from the European Space Agency. With an in-depth analysis of an expert workflow, we are able to increase the efficiency of the VR application and distinguish between user needs and user wishes. This distinction is essential in determining the appropriate requirements for application design as the VR environment introduces new benefits while simultaneously posing challenges for the user.

LESSONS LEARNED & OUTLOOK

The open source community provides millions of projects, many of the tools we use and all of the services our back end is built of. We set out to create a solution using closed source and proprietary software as sparingly as possible But this is rather difficult in the VR-department.

Limits of our free and Open Source Approach

First of all, the Unity editor and engine are proprietary. Unity-Technologies published reference C# source code[29] for both GitHub under a reference-only-license, which permits developers to look into the source code to better understand how the engine works and "reproduce and use the Software for Reference Purposes only.".[30] While Unity is free to use for noncommercial users, small studios and education, bigger companies and institutions such as ESA might have to purchase a professional license.

Second, the (optional) gesture-based control is facilitated by drivers and plugins provided by Ultraleap Technologies.[13] While the SDK is not free-to-use, purchasing the hardware needed to use this feature - the camera development kit - also comes with the SDK and plugin licence for commercial use.

And lastly, the most used VR headsets require proprietary Unity-plugins and driver software.

These three obvious issues notwithstanding, we tried to abstain from using any software that would force developers building upon our solution to purchase any additional licenses, in order to to enable as many people as possible to work with our code. There are plenty of free to use open source plugins, and even catalogs[31] of them, as well as thousands of free Assets in the Unity store. Many of these assets are provided under the very permissive MIT License[32] or the more restrictive Unity Asset Store License.[33]

The problem: At the moment there is no really "production-ready" free browser-plug-in for unity.

To "use" a web-browser within Unity, the content must be fetched by a "normal" browser instance (usually made available as part of an unity plugin), and a snapshot of the website must then be rendered as a 2D texture and handed to Unity to be rendered onto an in-game object. In addition, commands, such as entering an URL to navigate to and start loading a website must be made accessible in the front end and be sent back to the browser instance. So to really navigate and use a website, a ground station software web interface especially, one must be able to click on buttons and enter data into fields. Interactions with website elements are made possible by determining the exact position of the user input and forwarding the position, interaction and text input to the browser. Apart from this not insignificant task, web browsers are updated often and new Unity versions are not always backwards compatible, keeping a browser plugin up to date requires constant attention and work.

Initially, we included the open source SimpleWebBrowser[34] Unity plugin provided by Vitaly Chashin under the GPL−3.0 license. This plugin was developed five years ago and is compatible with Unity up to version 2020.3.20f1 LTS. Unfortunately we need to work in a more recent version of Unity to develop for current VR hardware.

Lesson Learned If depending strongly on an open source component provided by a single developer, a project should be prepared to commit a few person months to help keep that component alive or even improve it. We did not plan or allot time for this at the beginning of the VMCR and presently do not have the resources to contribute to a free browser plug-in implementation in the course of this project.

As a pragmatic solution, we will provide our source code as well as the compiled executables, that were created using a licensed proprietary browser plugin. Even so, the compiled VMCR executable and our source code are still freely shareable by ESA. However, to modify the scenes in Unity, developers will need to buy the license for the commercial browser plugin asset.

When a suitable free browser plugin becomes available, developers are free to adapt the scene, scripts and prefabs to work with that instead of the commercial solution included at present.

Outlook

Which features we adapt and/or implement next depends mostly on the results of the first round of user tests scheduled for summer 2023. And we hope to deliver a usable and well-documented VMCR prototype to ESA at the end of the funding period in May 2024. But this date does in no case mark the end of the project. Our chair will continue work on the VMCR not only by facilitating student thesis and internships, but also actively developing it. When further advanced, the project and code will be made available openly.

Acknowledgments

The development of the Virtual Mission Control Room is co-funded by the European Space Agency (ESA) as part of their "New ideas to make XR a reality"[35] campaign.

We would also like to thank all members of the open source community for providing many of the tools and assets we are using during development.

References

[1] https://grafana.com/.

[2] Felix Sittner, Cedric Liman, Gino Schulze, Jan Schmieder, Jan Tischhöfer, Marlene Busch, and Sergio Montenegro. Creating a Setup to Assess the Use of Virtual Reality for Mission Control. *Small Satellite Conference*, 2021.

[3] https://xr-hub.hci.uni-wuerzburg.de/.

[4] https://docs.unity3d.com/Manual/GameObjects.html.

[5] https://docs.unity3d.com/Manual/Components.html.

[6] https://docs.unity3d.com/Manual/Plugins.html.

[7] https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.3/manual/index.html.

[8] https://unity.com/products/netcode.

[9] https://docs-multiplayer.unity3d.com/transport/current/about/index.html.

[10] https://wiki.mumble.info.

[11] https://github.com/BananaHemic/Mumble-Unity.

[12] https://www.ultraleap.com/product/stereo-ir-170/.

[13] https://github.com/ultraleap/UnityPlugin/releases/.

[14] https://docs.ultraleap.com/xr-guidelines/.

[15] http://talesfromtherift.com/vr-canvas-keyboard/.

[16] https://github.com/ultraleap/XR-Keyboard/.

[17] https://www.docker.com/.

[18] Frank Flederer and Sergio Montenegro. Model-Based Framework for On-Board-Software. *Small Satellite Conference*, 2021.

[19] https://telestion.wuespace.de/.

[20] https://www.wuespace.de/.

[21] https://www.influxdata.com/products/influxdb/.

[22] https://redis.io/.

[23] https://github.com/filebrowser/filebrowser.

[24] Robert S. Kennedy, Norman E. Lane, Kevin S. Berbaum, and Michael G. Lilienthal. Simulator Sickness Questionnaire: An Enhanced Method for Quantifying Simulator Sickness. *The International Journal of Aviation Psychology*, 3:203–220, 1993.

[25] Sandra G. Hart and Lowell E. Staveland. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. 1988.

[26] Jens Gaab. PASA – Primary Appraisal Secondary Appraisal. *Verhaltenstherapie*, 19:114–115, 06 2009.

[27] Jakob Nielsen. Usability Inspection Methods. *Conference companion on Human factors in computing systems*, pages 413–414, 1994.

[28] Karen Holtzblatt and Hugh Beyer. *Contextual Design (Second Edition).* Morgan Kaufmann, 2017.

[29] https://github.com/Unity-Technologies/UnityCsReference.

[30] https://unity.com/legal/licenses/unity-reference-only-license.

[31] https://github.com/StefanoCecere/awesome-opensource-unity.

[32] https://en.wikipedia.org/wiki/MIT_License.

[33] https://unity.com/legal/as-terms.

[34] https://github.com/tunerok/unity_browser.

[35] The ESA Discovery Campaign on extended realities (XR). https://www.esa.int/Enabling_Support/Preparing_for_the_Future/Discovery_and_Preparation/The_Discovery_Campaign_on_extended_realities_XR.