Autonomous Robotic Inspection and Maintenance on Ship Hulls and Storage Tanks

Deliverable report – D7.2

Context	
Deliverable title	Virtual Reality for Real-Time Mission Monitoring
Lead beneficiary	RWTH
Author(s)	Simon OEHRL Sebastian PAPE Torsten W. KUHLEN
Work Package	WP7
Deliverable due date	31st March 2022
Document status	
Version No.	1
Type	DEMONSTRATION (REPORT)
Dissemination level	PUBLIC
Last modified	04 April 2022
Status	RELEASED
Date approved	04 April 2022
Approved by Coordinator	Prof. Cédric Pradalier (CNRS) Signature: [illegible]
Declaration	Any work or result described therein is genuinely a result of the BURWRIGHT2 project. Any other source will be properly referenced where and when relevant.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 871260

TABLE OF CONTENTS

LIST OF FIGURES

HISTORY OF CHANGES

Date	Written by	Description of change	Approver	Version No.
21.03.22	Simon OEHRL	Start of the document		1
28.03.22	Sebastian PAPE	Update		1.1
30.03.22	Laura MONNIER	Proofreading and validation	Cedric Pradalier	1.2

REFERENCED DOCUMENTS

- BURWRIGHT2 Description of the Action (DoA) Number 871260

This document will be stored on the file sharing site hosted by CNRS.

ABBREVIATIONS

VR	Virtual Reality
ROS	Robot Operating System
VPN	Virtual Private Network
POI	Points Of Interest

Executive summary

The goal of the BURWRIGHT2 project is to perform hull inspections using a fleet of different robots where mission planning and supervision should be supported by virtual reality (VR) user interfaces. This document aims to give an overview on the use of virtual reality for the user interface allowing the supervision of the robot team. It discusses how the VR application connects to the robots and how data is exchanged between them. It will present the different demonstrators that were built within the project to monitor all robot types used in BURWRIGHT2. Finally, it will cover the joint effort between UT (University of Trier) and RWTH to design and implement a sophisticated user interface for mission planning and supervision and discuss the evaluation considerations for the use of such interfaces in the field.

1. Connecting Unreal to ROS

As stated in 3.1.10 of the DoA, the user interfaces for mission planning and supervision are created using a game engine to accelerate the development. We chose to use the *Unreal Engine*[1] to utilise the existing expertise of the group. *Unreal Engine* is widely used for games and other 3D applications. It is highly modifiable through a plugin system and supports virtual reality applications natively. It can be programmed via C++ which makes it possible to include a wide variety of third-party libraries. In addition, it provides a visual scripting language called *Blueprints* which are excellent for rapid prototyping.

In the project, communication between robots and other systems is handled by ROS (Robot Operating System)[2], which is not supported by *Unreal Engine*. While it would in theory possible to integrate a ROS node directly into the application, either via a plugin or by modifying the engine itself, it does not seem feasible due to the different build systems and difference in the application architecture. A more maintainable solution to communicate with ROS is via the ROS bridge[3]. The ROS bridge provides a mechanism to interact with ROS functionality via a JSON/BSON API that can be reached via TCP and WebSockets. These standardised data formats and methods of transportation make it easy to communicate with a ROS core from any type of application. A third-party plugin exists that enables accessing and providing ROS topics and services[4]. However, the plugin has two limitations that made it unfeasible for the use in the project. First, it is limited to the BSON data format which is incompatible with other tools used in the project that require the JSON format and hard to debug in case of failures. Second, its functionality is not accessible via Blueprints, the visual scripting language used in *Unreal Engine*, which limits its usefulness for rapid prototyping.

To overcome these limitations, we developed a custom plugin for Unreal Engine called ROSBridge2Unreal[5]. It aims to provide an easy-to-use API that enables communication with a ROS core and builds the foundation for retrieving the data displayed in virtual reality applications. Figure 1 shows how the plugin fits in the context of a virtual reality application. The plugin is open source and was released under the 3-clause BSD license.

[1] https://www.unrealengine.com
[2] https://ros.org
[3] http://wiki.ros.org/rosbridge_suite
[4] https://github.com/code-iai/ROSIntegration
[5] https://github.com/VRGroupRWTH/ROSBridge2Unreal

It provides functionality for subscribing and publishing topics as well as calling and offering services from either C++ or *Blueprints* (see Figure 2). It supports the most commonly used message types as depicted in Table 1 and provides simple mechanism to implement additional message types.

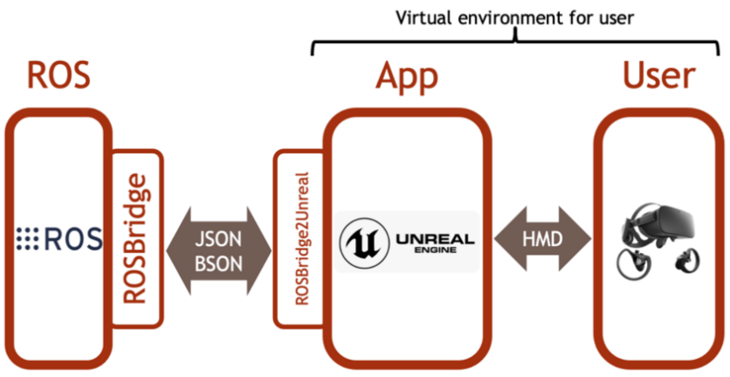

Figure 1 : Data exchange between ROS and Unreal Engine. The user interacts with the application developed in Unreal Engine via a head mounted display (HMD). The application uses the developed plugin to communicate with the ROS bridge that is connected to a ROS core instance.

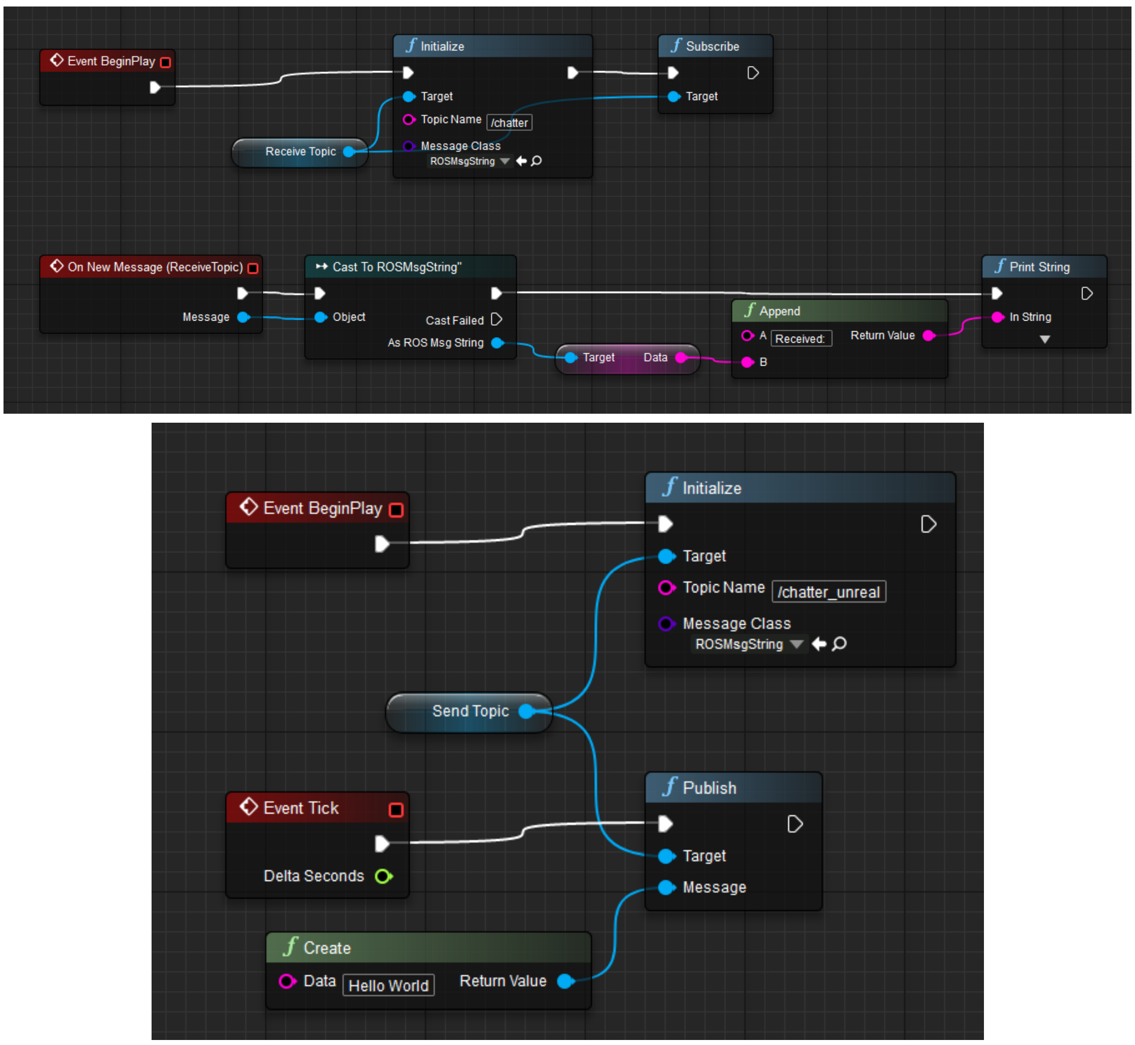

Figure 2 : Subscribing to and publishing a ROS topic. The Blueprint code necessary to publish a new topic (top) or subscribe to an existing one (bottom).

Message Type	Tested Use Case
std_msgs/Header	✗
std_msgs/String	✓
std_msgs/Float32	✗
std_msgs/Float32MultiArray	✗
std_msgs/MultiArrayDimension	✗
std_msgs/MultiArrayLayout	✗
std_msgs/UInt8MultiArray	✗
geometry_msgs/Point*	✗
geometry_msgs/Pose*	✓
geometry_msgs/PoseStamped	✗
geometry_msgs/PoseWithCovariance*	✗
geometry_msgs/PoseWithCovarianceStamped	✓
geometry_msgs/Quaternion*	✗
geometry_msgs/Transform*	✗
geometry_msgs/TransformStamped	✗
geometry_msgs/Twist*	✗
geometry_msgs/TwistWithCovariance*	✗
geometry_msgs/Vector3*	✗
rosgraph_msgs/Clock	✗
nav_msgs/Path	✓
sensor_msgs/CompressedImage	✓

Table 1: Implemented message types. The right column indicates whether there is a tested use-case for this message type within the project yet. A star next to the message indicates that the Blueprint version of the message will have reduced precision for floating point numbers, as they only offer 32 bit floats compared to the 64 bit floats specified by ROS. Using the messages from C++ will not suffer from this reduced precision.

2. VPN Acquisition

An unforeseen challenge in the project was the lack of physical meetings with project partners due to the travel restrictions caused by the COVID-19 pandemic. Thus, all integration work must be performed remotely. Establishing a remote connection between partners turned out to be difficult due to the restricted network access caused by the firewalls of the individual partners. Opening the networks is a time-consuming process and only solves the problem for specific partners. It is also not possible for every partner as their network administration may not allow opening the network. Thus, a general solution to this problem was needed. A virtual private network (VPN) that is accessible by all partners will solve the problem without additional involvement of the party's individual IT departments. We ordered a VPN with high bandwidth for the use in the whole project for its remaining duration. It was set-up successfully and invitation links as well as instructions on how to use it were sent to all partners. It is fully working and has been already proven to allow collaborative work in multiple occasions.

3. Robot Monitoring

During the project several demonstrations were developed that show the ability to connect to different types of robots to extract and visualize their data. The following sections will describe three different demonstrations for each of different robot types that are used inside the project. All applications are created with the *Unreal Engine* and communication is done via the *ROSBridge2Unreal* plugin described in Section 1.

3.1. Drones

The first demonstration was closely developed together with UNI-KLU and is able to monitor and steer a single drone in the real world from inside a virtual reality application. Upon launching the application, the user is placed on top of a ship inside a virtual world which is shown to them via a head mounted display (see Figure 3 top right).

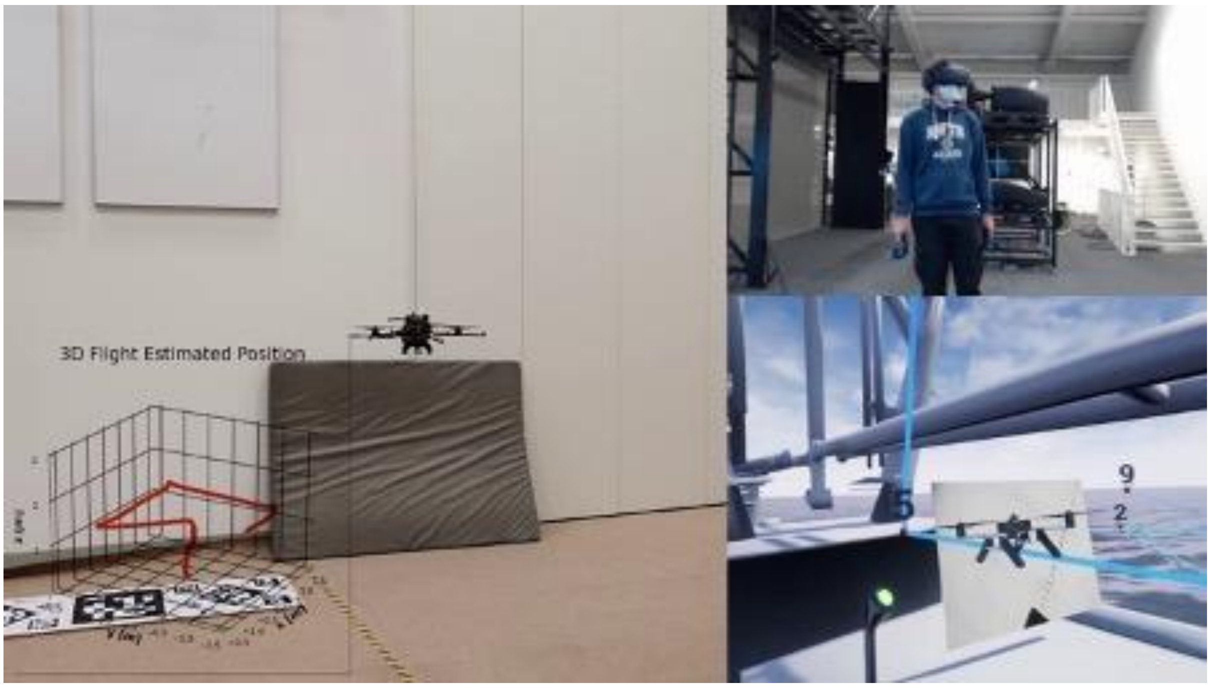

Figure 3: Live demonstration of drone monitoring and steering using a VR application.

The user is able to freely navigate inside the virtual world by moving physically. The user can create and delete waypoints in the space to define a path for the drone. The waypoints are shown to the user with an attached number next to them to indicate their order. Additionally, a line connecting the waypoints is rendered with an animation indicating the direction of the path to give the user a more intuitive understanding on how the drone will move through the world (see Figure 3, bottom right).

There is a button on top of the ship that can be pressed to send the path to the drone which will then start to follow the path in the real world. The estimated position of the drone as well as the onboard camera feed is visualized inside the application and can be monitored in real-time. The demonstration was shown at the second integration week in January 2021 and was performed remotely. I.e., the drone was located in a drone hall in Klagenfurt, Austria and monitoring and steering was performed from Aachen, Germany.

3.2. Crawler

A second demonstration was developed in close cooperation with CNRS to monitor the crawler movements on a steel plate. The application shows a 3D model of the steel plate and the crawler (see Figure 4). The position of the crawler in the application is synchronised with the estimated position of the crawler in the real world. The path the crawler has already taken is indicated by a dotted green line. In this case, it performed a sweeping pattern on the steel plate. The user can freely move around in the virtual world and inspect the mission from an arbitrary viewpoint. This can be either done as a normal desktop application by using keyboard and mouse input or via an HMD with more natural controls and increased spatial awareness.

The demonstration shows the output of the depth sensor built into the crawler as a green point cloud in front of it. Additionally, two camera views are embedded in the bottom left and right corners of the application. The camera image on the left shows the experiment setup from an external view and was only there for demonstration purposes. The camera image on the right shows the view from the crawler projected into the plane it is moving on. This can be used to create a colour texture for the ship by projecting the received image onto the 3D model of the ship.

An algorithm for projecting images from crawlers or other robots onto arbitrary 3D models was implemented within the Unreal Engine. It assumes that a 3D model exists with non-overlapping UV coordinates and consists of two steps. First, the UV coordinates of the 3D model are rendered from the point of view of the robot into a texture with the same dimensions as the image (see Figure 5 left). The virtual camera has to be carefully adjusted to match the point of view of the real camera exactly. In a second step, the UV coordinates in the resulting image can be used as a lookup table for every pixel of the original image (see Figure 5 right). Carefully weighting the pixels before writing them to the texture accomplishes the aggregation of all images during over time yielding a more complete texture for the underlying model. Such methods can not only be used for colours but also for measurements such as the thickness of the ship hull.

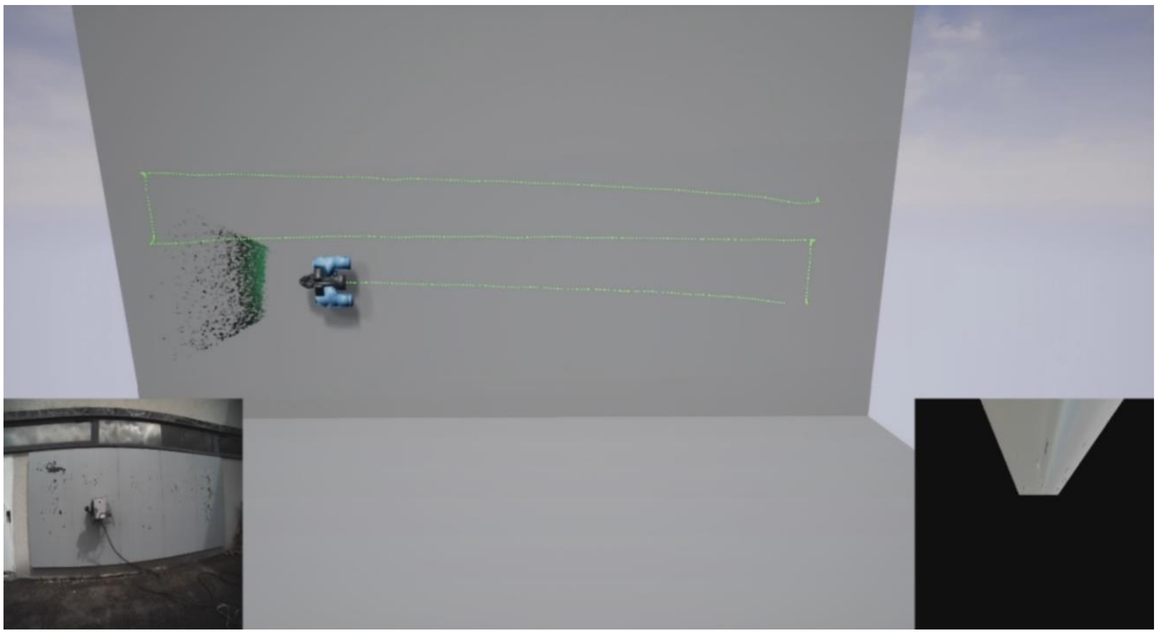
Figure 4 : Demonstration of crawler monitoring

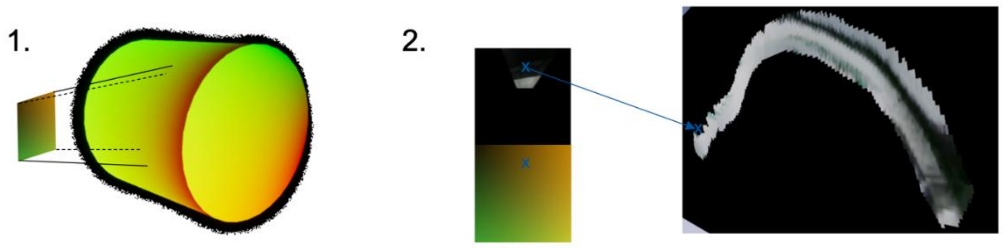

Figure 5 : Projection and aggregation of images onto arbitrary 3D models

3.3. Underwater UAVs

Similar to the demonstration of the crawler monitoring, we developed a proof-of-concept application that allows the monitoring of the Pioneer, underwater UAV manufactured by Blueye (BEYE). Thus, the demonstration was closely developed in cooperation with BEYE. The demonstration shows a virtual world containing a simple ship model and animated water that serve as a frame of reference (see Figure 6).

Monitoring the location of the UAV underneath the water surface is done by rendering a 3D model of the Pioneer at its estimated position. Previous positions are indicated via a purple marker. In addition, a live feed of the Pioneer's built-in camera is attached to its model. For this demonstration, the robot was manually steered. The data was recorded and later played back for the monitoring application.

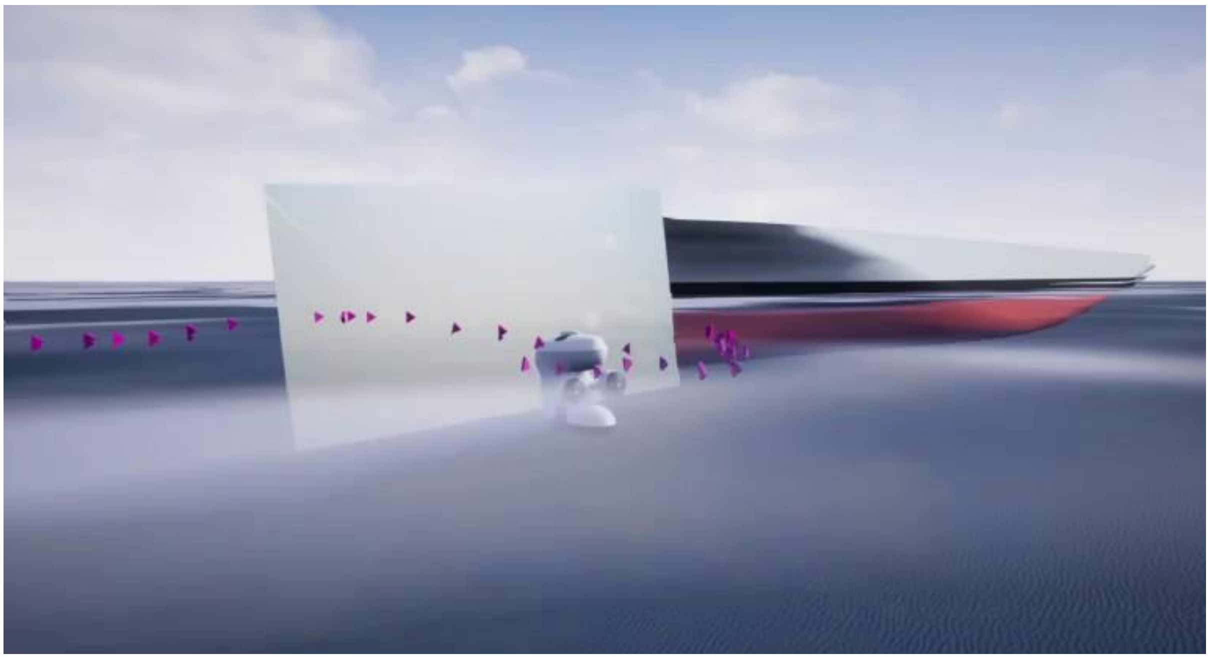
Figure 6: Demonstration of monitoring a Blueye Pioneer underwater UAV

4. User Interface Implementation

The long-term goal inside the project is to unify the different prototypes presented in Section 3 into a single application used for monitoring and mission planning. An important aspect of the resulting application is a convenient user interface that provides all necessary information in a clear way. To achieve this goal, we closely collaborate with UT to design the interface in an iterative manner in regular meetings. The basis for the design is the workflow analysis performed in T1.5. The user interface is designed as a desktop application utilising a multi-view approach. An example of a single screen is shown in Figure 7.

The shown screen called *Dashboard* should give an overview of the current inspection process. The central points of this screen are the two 3D views showing a 3D model of the ship which can be individually controlled. The top shows the list of points of interest (POIs) in 3D space. POIs can be manually or automatically placed and should indicate areas that need special attention as there are detected defects or there have been in the past. The panel to its right contains more details about the selected POI. The bottom view shows all the camera images of all currently deployed robots projected onto the ship. Similar to the POIs, a panel to its right contains more information about a selected robot. The two panels on the left shows a preview of all camera images and a live view of all sensor data that is currently collected. Arrows connect the data in the panels to the robots / POIs in the 3D view to enable the user to quickly locate the area a specific dataset belongs to.

The user should then be able to switch to a virtual reality environment on demand for certain tasks that benefit from a VR interface, e.g., in places where increased spatial awareness is crucial or a more natural interaction with the virtual world is desirable. An example of such a task could be defining an area in the virtual world that should be inspected by a given robot.

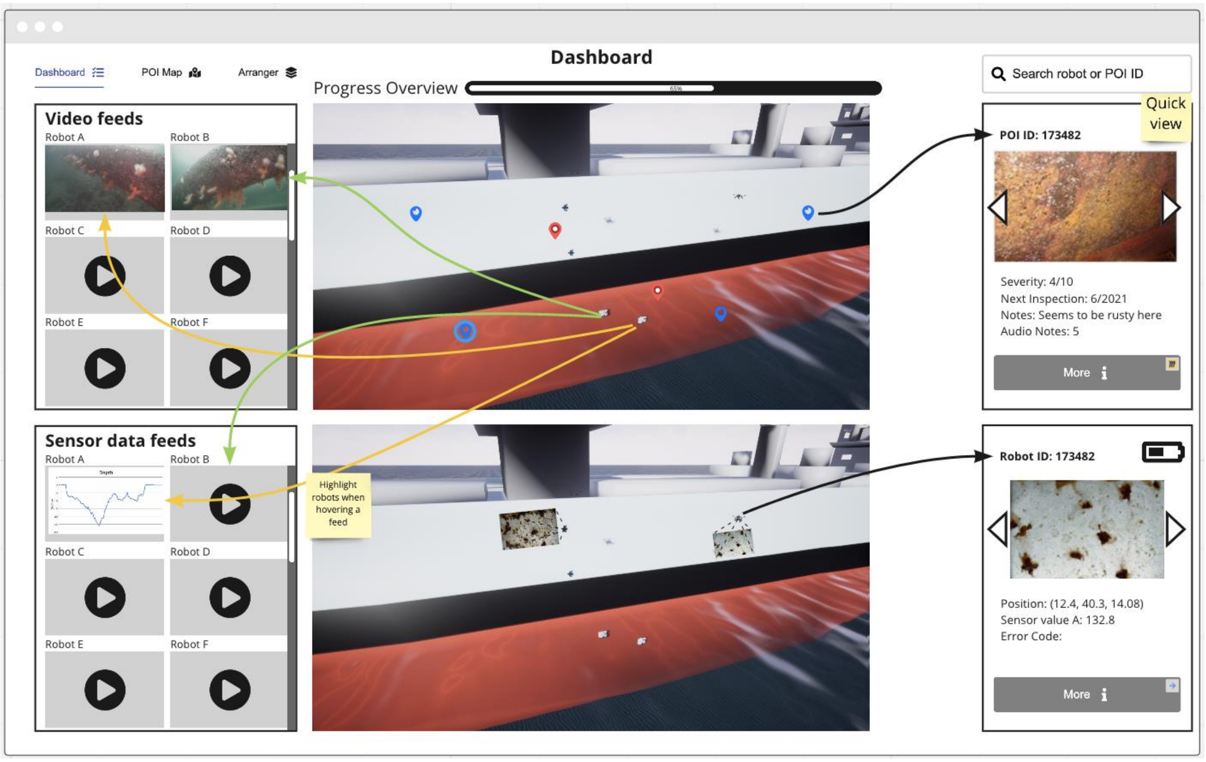

Figure 7: User interface design concept

Alongside the design of the interface, we started implementing the concept using the *Unreal Engine*. The current implementation of the previously described screen in shown in Figure 8. The current implementation features the two independent 3D views of the ship with alongside the robots on the hull marked by the blue icons. The live video feeds are displayed on the side and linked to the 3D view via a dynamically rendered arrow. The sensor data feed is currently showing placeholder data as such data is currently not collected from any of the robots.

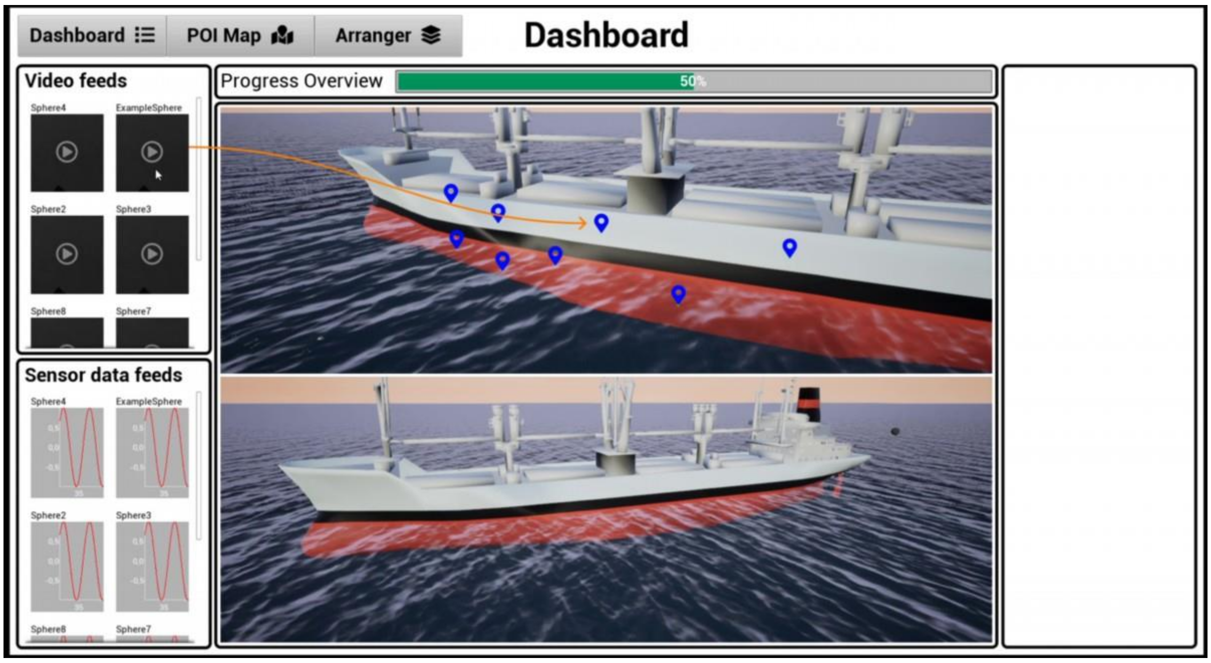

Figure 8: Implementation status of the user interface design

5. Evaluation of the suitability for field deployment

An important aspect when designing and implementing user interfaces is the evaluation by actual users. This includes which data is relevant to them during which parts of the process and how the user interacts with the interface. In the context of virtual reality, it is especially important as many users are not familiar with it.

We presented all demonstrators described in Section 3 as well as the user interface prototype shown in Section 4 to the project partners, including potential end-users, during the virtual integration weeks. The discussions afterwards gave valuable insights on their expectations and the importance of the individual components.

A field visit to the Arsenal Do Alfeite (AASA) in Lisbon, Portugal was planned in cooperation with UT to evaluate the user interface implementation as well as VR interaction techniques using hands on approaches. However, the participation of RWTH had to be cancelled due to concerns regarding travelling in the times of the COVID-19 pandemic, which led to the cancellation of the hands-on evaluation.

Further evaluation will however be done in the following integration weeks. A demonstration similar to the one described in Section 3.1 is planned to be used in the integration week in May 2022 in Greece for an on-site mission planning and monitoring demonstration. This will gain insights on how suitable such interfaces are in the field. The in-person integration week in Norway planned in June 2022 can be used to get hands-on feedback on the interface implementation by project partners as well.

6. Conclusion

This document showed the progress on robot supervision using VR interfaces in the project BURWRIGHT2. We elaborated on how a connection between ROS and an application developed in Unreal Engine can be established and presented the plugin we developed for this purpose. The proper functioning of the plugin was demonstrated by developing monitoring applications for all robot types used within the project (drones, crawlers and underwater AUVs). The demonstrations showed that we were able to connect to all platforms and are able to supervise them during mission execution.

In addition, we presented the user interface design developed together with UT and showed the current progress in implementing the design. Finally, we covered how the interfaces concepts were discussed with potential end-users and how the demonstrations and the interface prototype will be evaluated in future integration weeks that can take place physically.