Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 51-29 (2018) 26–31

ARSEA: A Virtual Reality Subsea Exploration Assistant ⋆

Francisco Bonin-Font * **Miquel Massot Campos** *
Antoni Burguera Burguera *

* *Systems, Robotics and Vision Group, University of the Balearic Islands, 07122 Palma de Mallorca (e-mail: francisco.bonin@uib.es, miquel.massot@uib.cat, antoni.burguera@uib.es).*

Abstract:
This paper presents a new application designed and developed to teleoperate underwater vehicles from a comfortable and immersive virtual reality environment. This environment is developed in Unity with the Oculus Rift as a visualizer, and communicates with the ROS-based underwater robot architecture via a Unity-ROS-bridge library. The proposed controlling system offers two main utilities: a) it permits to command the vehicle with two Oculus Touch Controllers, and b) contrarily to traditional *Remotely Operated Vehicle* (ROV) control panels, this environment shows an accumulative 3D reconstruction of the whole zone explored by the robot. The proposed solution also offers, inside the virtual environment, complete navigation data, such as, vehicle speed, global pose and distance to the bottom and the original images captured by the vehicle cameras, facilitating its continuous and on-line analysis and the process of navigation decision taking. Teleoperation experiments have been conducted with a simulated robot in an simulated aquatic environment, while environment perception experiments have been done reproducing ROS log files which contain stereo video sequences taken by the vehicle in real marine scenarios. The stereo images have been used to reconstruct the natural scenarios in the virtual reality environment.

Keywords: Underwater vehicles, teleoperation, virtual reality.

1. INTRODUCTION

Thanks to recent technological advances in *Remotely Operated Vehicles* (ROV) and specially *Autonomous Underwater Vehicles* (AUV), remote and complex underwater scenarios are more accessible for scientific and industrial activities, such as surveying, scientific sampling (archeology, biology, geology, etc.), rescue, recovery or industrial infrastructure inspection and maintenance. Commercial ROVs usually imply a support vessel and a complex and expensive infrastructure formed by a vehicle and an associated *Tether Management System* (TMS). All these components are operated from a control panel, which offers continuous data of the surroundings given by the robot sensors, basically, images captured by the cameras, water temperature, depth, altitude, orientation and in some cases, the vehicle position in *North-East-Down* (NED) coordinates. The TMS configuration and the robot motion depend entirely on the decisions taken by the operator through the panel, as he visualizes the images and the rest of the data. Although there are some low-cost and size commercial ROVs, without TMS (Fishers, 2017), lightweight AUVs try to overcome their intrinsic limitations. AUVs are a promising tool for underwater missions in highly repetitive, long or hazardous missions. Because they are programmable, untethered and self-powered, AUVs offer a significant independence from surface support ships and weather conditions. This reduces notably the operational costs and human intervention in critical situations (deep ocean, under ice, mine retrieval or operating in classified areas), automatizing as many procedures as possible. However, the lack of permanent connection with a central station severely hampers the control of the mission evolution and its rectification online, if needed. Monitoring, continuously, the activity of the vehicle and all the environmental data available, becomes, sometimes, crucial to guarantee the mission success and the recovery of the vehicle.

The traditional way to operate in unknown environments starts with the construction of acoustic bathymetric maps. Once the bathymetry is available, ROVs or AUVs can be sent to obtain more details using short distance sensors with higher resolution (Viswanathan et al., 2017). Bathymetries are 2.5D elevation maps rather than 3D structures but they are a useful first approximation of the environment. In the last decade, the use of cameras as another part of the sensor suite of underwater vehicles has experienced a high increase and they are extensively used for short-range surveying. In particular, stereo vision perception can strongly benefit some robot abilities, such as 3D sensing and modeling, with limitations in the point cloud online process and visualization (Jasiobedzki et al., 2008; Weidner et al., 2017). Planning and executing underwater operations can be improved, corrected or even reorganized on line if a 3D model of the robot surroundings is available. Commercial ROVs are usually endowed with one or several cameras providing on line images to a human operator who is in charge of piloting, remotely, the whole infrastructure (vehicle, TMS and camera tilt) using complex control panels. Normally, AUVs can also be teleoperated as ROVs. However, limitations in the camera field of view and the difficulty to remotely maneuver a 6DOF vehicle with complex dynamics make the piloting a difficult and error prone task requiring trained operators. The main disadvantages

⋆ This work is partially supported by Ministry of Economy and Competitiveness under contracts TIN2014-58662-R (AEI,FEDER,UE), DPI2014-57746-C3-2-R (AEI,FEDER,UE), DPI2017-86372-C3-3-R (AEI,FEDER,UE).

10.1016/j.ifacol.2018.09.464

of the current ROV or AUV teleoperation systems are related to the impossibility to accumulate useful data during the operation process that can be consulted on line and used for decision making: a) images of already visited areas can not be re-visualized on line; the operator can only see at each moment what is in front of the cameras, b) the experience is not immersive, that is, there is no possibility to have a 3D model of the already visited area, which could be used to identify points of interest where to drive the vehicle again.

In the context of the ARSEA national research project, our proposal is aimed at solving these problems by providing a novel, advanced, practical and comfortable underwater vehicle control interface based on virtual reality, including head tracking. The main goal is to place the user on board the real vehicle from a remote virtual environment, focusing on two main objectives: a) The control system has to be immersive; it has to accumulate a 3D model of the environment being explored by the robot, and updated on line continuously, as the robot moves and captures successive images; this 3D reconstruction will be of great utility for the pilot to re-plan oncoming actions, or to simply observe the explored area at any moment, as he was immersed in it, and b) The control interface must permit the robot teleoperation, and the visualization of all the navigation data useful for the pilot, such as, robot pose, images, velocity, altitude, depth, etc.

Early approaches on virtual reality applied to underwater environments suggested a series of guidelines to implement interactive and collaborative underwater robot teleoperation utilities (Monferrer and Bonyuet, 2002). However, very few pieces of work mention the possibility to command underwater robots in real scenarios via virtual reality platforms. Garcia *et al* (García et al., 2015) presented a novel control environment based on virtual reality applied to the simulator UWSim (Prats et al., 2012). In this work, the feeling of operators when using different configurations of cockpit, leap motion or traditional joysticks was assessed. However, this work did not enter in the on line perception of real scenarios inside the virtual reality environment, leaving a promising research line opened to new contributions. Virtual reality has been also used to implement simulators (Carvalho et al., 2015) which reproduce real underwater environments with a high degree of realism. Simulators are usually used for integral testings before the real systems are deployed into the water (Zhang et al., 2015), but they are not designed to substitute the traditional ROV consoles.

As a result of the work presented in this paper, underwater inspection tasks can significantly change. On the one hand, the remote operation is facilitated, making it less error prone, reducing the operator training time and focusing their efforts on decision taking and fine maneuvering rather than console understanding. On the other hand, having a 3D model of the explored area helps to increase the ROV operating range, in space and time, thanks to the knowledge of the environment, when compared to tasks currently performed either by ROVs or human divers.

2. SOFTWARE STRUCTURE

The software infrastructure presented in this paper has two different parts: Unity (Unity Technologies, 2018) and ROS (Quigley et al., 2009), both connected by a Websocket bridge.

2.1 The ROS Environment

The ROS environment must be installed on a Linux operating system and supports the software architecture of our testing vehicle, a SPARUS II AUV unit (Carreras et al., 2018) called Turbot. The aforementioned vehicle architecture is based on the original Cola2 (Palomeras et al., 2012) which includes a teleoperation node. Turbot is equipped with an IMU, two stereo rigs (one looking downwards and another one with an adjustable tilt), a pressure sensor, a DVL and a *Ultra Short Baseline* (USBL) acoustic modem. The Turbot architecture provides to the ROS core the vehicle navigation status at a rate of 25 Hz, which includes the vehicle global pose (position-north, east and depth- and orientation), altitude, velocity and orientation rate. This navigation data is obtained integrating all information given by all the vehicle sensors in a double *Extended Kalman Filter* (EKF) context (Guerrero et al., 2017). The Turbot architecture can accept either a set of consecutive way points from a preprogrammed mission and generate the corresponding motion orders to respond to each goal point, or a continuous body-referenced velocity request (3 components for the linear velocity and another 3 for the angular velocity) sent by, for instance, a joystick or a laptop keyboard.

Together with the control architecture, the vehicle computer also runs:

1- An image processing node which: a) grabs images from the cameras, b) rectifies them, c) downsamples the original resolution by 4, and d) builds a 3D point cloud from each stereo pair, referenced to the camera system of coordinates.

2- A Point Cloud processing node which: a) downsamples the original point clouds using the PCL Voxel Grid utility (PCL Library, 2018) to reduce runtime memory requirements; this class creates a 3D voxel grid over the input point cloud; thereafter, all points that are in each voxel are approximated with their centroid, and b) it transforms every point cloud from the camera to the global frame, at the vehicle position in which the corresponding stereo pair was taken.

2.2 The Unity Environment

The immersive piloting environment has been build using Unity, a multi-purpose cross-platform game engine that supports 3D graphics and integrates the Oculus Rift headset and the Oculus Touch Controllers (Oculus, 2018a). The Oculus Rift is used as a visualizer and the Touch Controllers are used as remote control devices. Unity has version for Windows and Linux, but, to the extent of our knowledge, the Oculus driver for Linux has not been released yet. So, the Unity-Oculus binomial had to be installed on a desktop with Windows 10 operating system, an *intel-i7* processor with 32 Gb of RAM, one *solid state drive* (SSD) of 240Gb, and a special graphical card Nvidia GTX1080, compatible with the Oculus Rift demands.

The integration between the Linux-ROS robot environment and the Windows-Unity environment was performed using the ROSBridge Suite (WillowGarage, 2018), which provides a JSON (JSON, 2018) API to ROS functionality for non-ROS programs, such as, for instance, an Unity application. ROSBridge Suite has two main members, a) the ROSBridge Library which is the core package responsible for translating a JSON string into a ROS command and vice versa, and b) the ROSBridge Server which manages the transport layer that provides

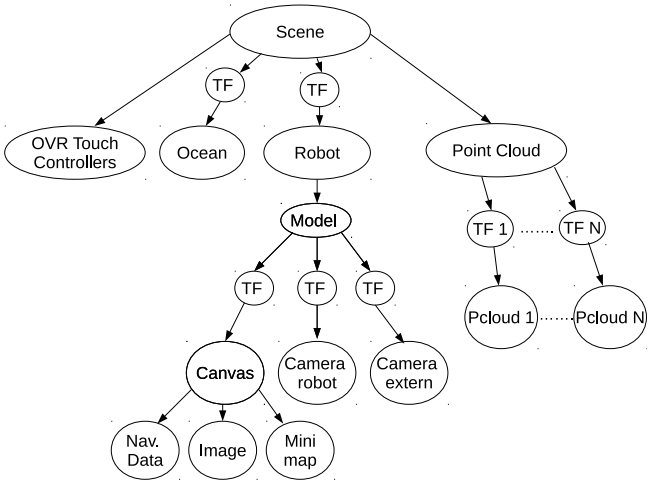


Fig. 1. Hierarchy of elements in the Unity scene.

a WebSocket connection, so that different ROSBridge partners can communicate with the ROS environment for data transfer. The development has been done analogously to the project *unityros* (Jenkin, 2017). Additionally, controls and *game objects* (elements on the Unity scene) for teleoperation and data visualization have been added.

The Unity project consists of two main modules: a set of *C#* scripts linked to a set of *game objects*, which are the elements displayed in the virtual environment.

The Unity environment is composed by 4 main elements: the OVR Touch Controller, the Ocean, the Robot and the Point Clouds.

1- The OVR Touch Controller components manage the tracking and manipulation of the Oculus Touch Controllers used to command the vehicle in the virtual environment.

2- The Robot model (SPARUS II) includes the two camera views (one external to the vehicle and another one attached to the vehicle) and a *Canvas*. The *Canvas* contains a panel with the navigation data (north, east, depth and altitude), the image and the mini-map (a visualization of the 3D model projected onto the 2D horizontal plane). The image shown in the panel corresponds to one of the two images of each stereo pair grabbed by the Turbot stereo-down camera. The point of view of the virtual cameras depends on the orientation of the Oculus Rift. These virtual cameras are completely unrelated to the Turbot cameras. They are just two different view points from which the virtual scene is visualized, and are moved according to the Oculus tracking.

Figure 1 illustrates the hierarchy of elements that form the scene. The TFs represent the coordinate transforms applied to each element to be placed in its corresponding location in the virtual scene. Figure 2 shows a screen shot of the Unity scene on the development environment.

The *C#* source structure starts from an Assets directory. An asset represents an item usable in the Unity project and it may come from a 3D model, an audio file, an image, or any other type of file that Unity supports. The structure contains 4 main blocks:

a) The OVRInput API (Oculus, 2018b) used to integrate, detect and manage the Oculus Touch Controllers. The secondary thumbstick is used to navigate in the (x, y) plane; the thumb-

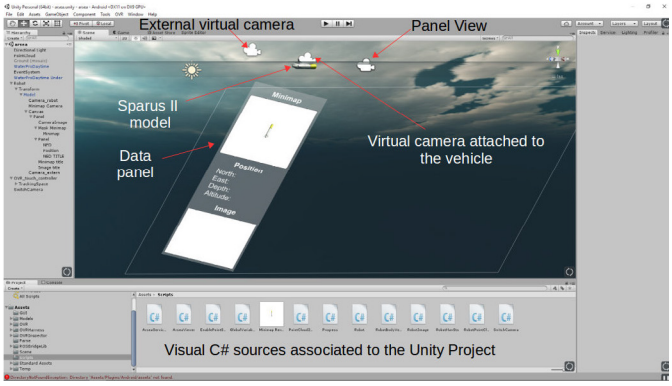


Fig. 2. The Unity development environment.

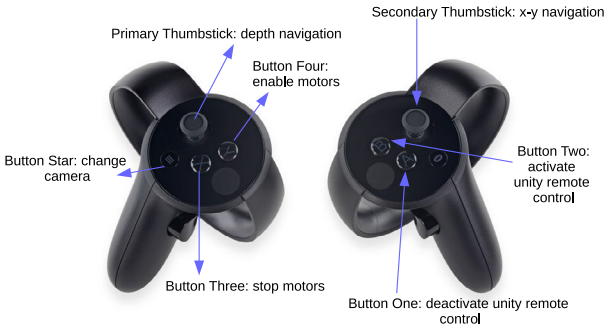


Fig. 3. The Oculus Touch Controllers.

stick y axis is mapped into the vehicle x (North) axis and the thumbstick x axis is mapped into the vehicle y (East) axis. The $-y$ axis of the primary thumbstick is moved into the z (Down) vehicle axis. Finally, the vehicle angular velocity in *yaw* is calculated according to $yaw = atan(v_y/v_x)$, where v_y and v_x are the linear velocities in the y and x axis, respectively. Figure 3 illustrates the different functionalities programmed at each Touch Controller.

b) The ROSBridge Library sources, the JSON libraries and all the JSON data structures that replicate the ROS original messages used in the Turbot architecture.

c) Scene which contains all the internal files needed for the Unity scene.

d) Scripts. This directory contains: 1- the main program that manages the ROSBridge Websocket connection, the declaration of subscribers, publishers and services, the point cloud accumulation and drawing and the transformation of the Touch Controllers into JSON structures of velocity requests; 2- the callback functions activated when ROS messages of navigation status, point clouds or stereo images are available in the Unity environment. This availability happens thanks to the ROSBridge Server. The pose data contained in the navigation status ROS message is used to move the SPARUS vehicle model in the Unity game scene, accordingly to the sequence of vehicle real positions, but taking into account that the Unity system of coordinates is turned 90° in roll and 180° in yaw with respect to the ROS system of coordinates. The point clouds arrive to Unity downsampled and positioned with respect to the origin of the global system of coordinates. In order to alleviate the

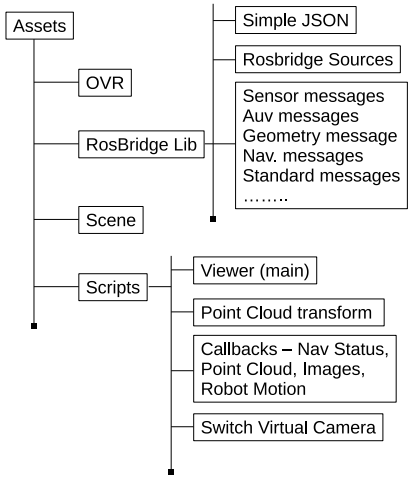


Fig. 4. Structure of the *C#* project.

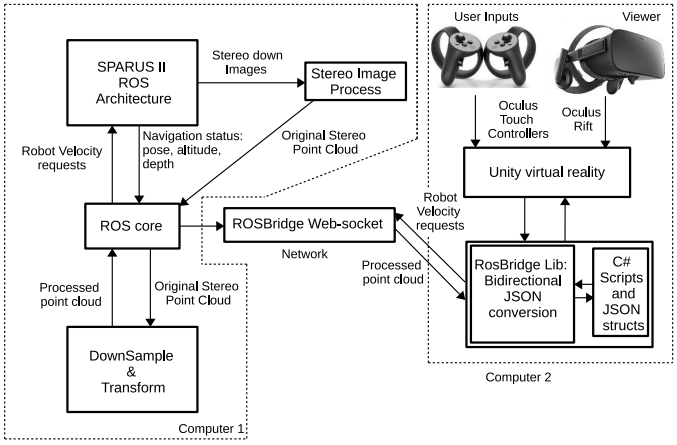


Fig. 5. Software Structure Schema.

amount of 3D points that need to be accumulated in the virtual environment, the system displays only the 3D reconstructions obtained from the images downsampled by 4. Since the Unity engine does not render meshes with more than 65000 points or vertex, all point clouds coming from the ROS environment have to be split into blocks of 65000 points, and each block must be placed in the scene separately. Oculus only shows, at each moment, those elements that are in the frustum of the current active camera, avoiding to charge in memory elements that are not being visualized.

Figure 4 illustrates the structure of the *C#* project associated to the Unity project. Figure 5 illustrates the whole software infrastructure and data exchange described previously.

3. EXPERIMENTS

Two different types of preliminary experiments were performed to validate the utility of the proposal: a) Several *bag files* (Quigley et al., 2009) recorded with the Turbot AUV in different real marine scenarios were run, and reproduced in the virtual environment. ROS *bag files* are files used for data logging. They can store the serialized ROS messages as they are received. These files can be played back in ROS to retrieve the stored data as they were obtained on line, and b) Teleoperation tests using a simulated SPARUS in a simulated environment.

3.1 Scene Perception in the Virtual Reality Environment

A set of experiments were conducted with a laptop running a ROS *bag file*, directly connected to the Unity desktop via a dedicated Ethernet cable. All data and image sets were recorded with Turbot in different marine regions of Mallorca, performing different trajectories. Some of these scenarios were densely colonized by seagrass, and others were formed by stones, sand and spread bunches of seagrass. All the *bag files* stored the vehicle navigation status, the stereo-down camera images and the altitude, among other topics of interest. The navigation data together with the scaled frames (original resolution downsampled by 4) and the downsampled point clouds were sent to the Unity environment to complete the virtual scenario. Since the navigation data and the images did not come from a vehicle but from a recorded *bag file*, the Oculus Touch Controllers were not used to command anything, but only to change the virtual camera view. Sending the downscaled images and point clouds to Unity permitted a considerable reduction of the data exchange through the Websocket connection, and the number of points to be stored in the graphical card memory, in case the exploration time (thus the 3D model) increases. The accumulated point clouds formed a virtual world from a real marine scenario and provided a captivating sensation of being in the place where the robot moved. The Oculus headset offers a view of the environment, either from the point of view of the camera attached to the vehicle or from the external camera, giving the feeling that the operator is immersed in it. The virtual 3D scene is shown depending on the orientation of the Oculus, meaning that, if the Oculus looks forward, the camera shows the virtual 3D scene that is in front of the vehicle and if the Oculus looks backwards, the camera shows what is behind the vehicle. The aim of these preliminary experiments was not to reproduce in Unity an extremely realistic 3D virtual map of the surroundings, with high resolution, texture blending and no motion drift, but to test the suitability of the system to perceive it and to control an underwater vehicle from a virtual and immersive environment close, as much as possible, to the reality.

An illustrative video of the Unity virtual environment reproducing the data of some of the aforementioned *bag files* can be watched at (Bonin-Font, 2017b). The Unity virtual environment is shown in the central part of the screen while the video showing the operator gesticulation with the Oculus Rift is superimposed in the top of the screen.

Figure 6 shows 4 views captured during the reproduction of the *bag files*. The virtual scenario is on the main screen while the operator with the Oculus Rift can be seen superimposed at the top of each Figure. Figure 6-(a) shows the operator looking forward, to the data panel, and what he sees from the external virtual camera. The data panel shows the image of the real marine scene captured by the Turbot stereo down camera, and the vehicle position data. Figure 6-(b) shows the operator looking forward, to the same direction as the vehicle motion, and the scene seen from the external virtual camera, showing part of the environment build in 3D, with the boat anchor chain just in front. Figures 6-(c) and 6-(d) show the operator looking backwards, and what he sees from the external virtual camera and from the virtual camera attached to the vehicle, respectively. Notice how, in Figure 6-(d) the operator sees what the vehicle left behind, while, in 6-(c), the view is partially occluded by the rear of the vehicle.

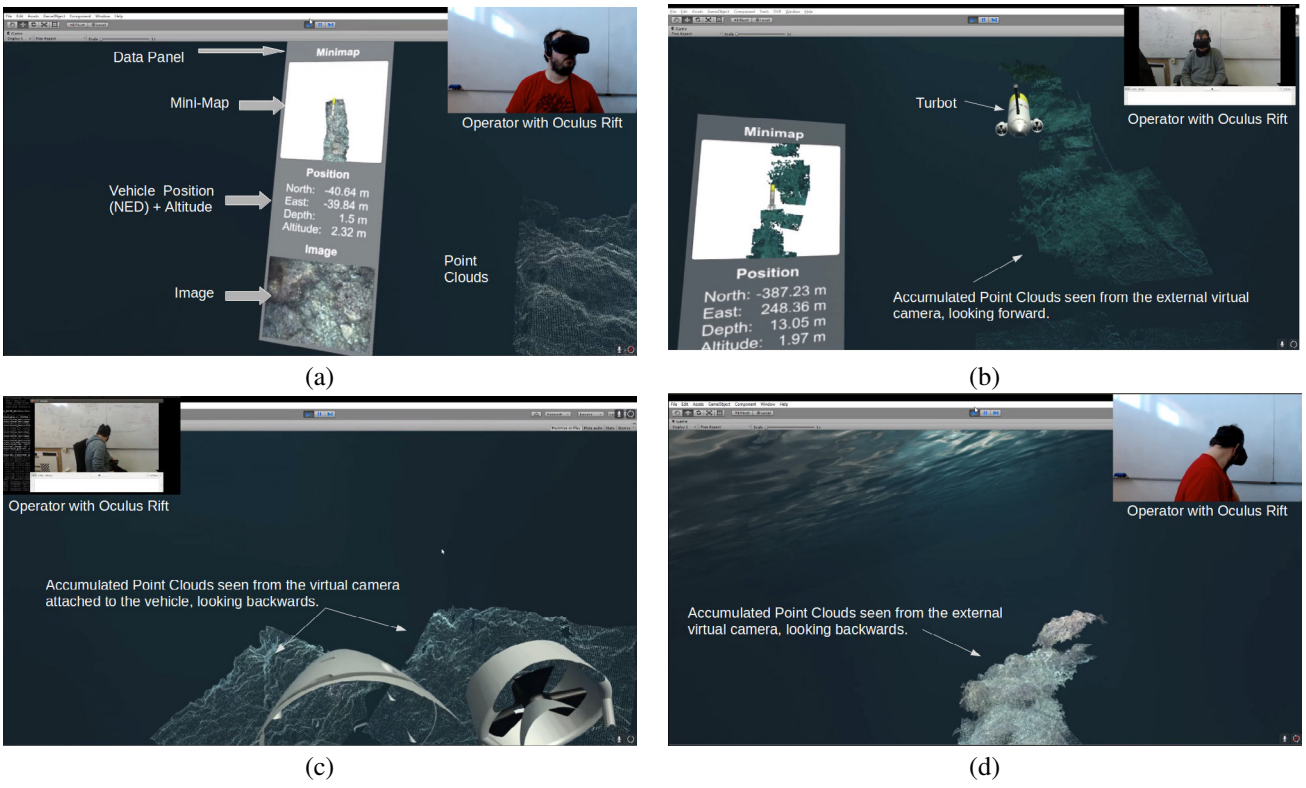


Fig. 6. Different views during the *bag files* reproduction. (a) The operator looks downwards. (b) The operator looks forward. (c) The operator looks backwards. (d) The operator looks backwards, slightly downwards.

3.2 Teleoperation

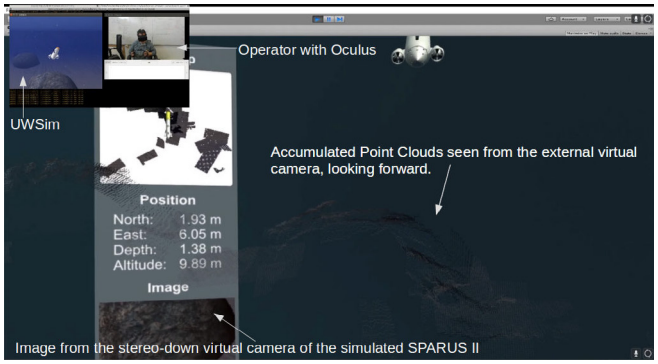
The first experiments of controlling Turbot from a virtual scenario were done with the laptop running the simulator UWSim, and also directly connected to the Unity desktop via an Ethernet cable. The Turbot dynamic model and software architecture are perfectly replicated in the simulator, in such a way that, the simulated vehicle in the simulated environment gives the same dynamic response in terms of navigation (body linear and angular velocities) and control as the real vehicle in a marine scenario, including wave forces, torques and currents. The simulated vehicle and environment acted as the real vehicle was connected to the Unity desktop via either an umbilical cable or, for instance, a long-range wifi antenna attached to a buoy connected to the vehicle. The vehicle was teleoperated from the Unity game scene using the Oculus Touch Controllers, obtaining a complete immersive piloting experience. The simulated SPARUS II on UWSim moved according to the commands generated by the Oculus Touch Controllers, exactly in the same way as the real robot would move in a real marine environment. As in the previous experiments, the virtual scenario showed also the accumulated 3D model, but this time, the 3D virtual scenario was build with the point clouds of the UWSim simulated scene, as they were progressively created during the robot motion. The images of the bottom were obtained by the simulated cameras created on the simulated Turbot model. Both, images and point clouds were also downsampled. From the perspective of control, the navigation of the vehicle is as fine as it is when the vehicle is commanded directly from the ROS architecture teleoperation module and a joystick.

Figure 7 shows 3 views captured during the teleoperation experiments. The virtual scenario is on the main screen while the simulated vehicle and the operator with the Oculus Rift can be seen superimposed at the top of each Figure. Figure 7-(a) corresponds to views from the external virtual camera while Figures 7-(b) and 7-(c) correspond to views from the virtual camera attached to the vehicle, one looking forward, and the other looking down.

An illustrative video of the simulated vehicle teleoperation from the Unity virtual environment can be watched at (Bonin-Font, 2017a). The Unity virtual environment run in the desktop is shown in the central part of the screen, from the point of view of the external camera, while the UWSim simulator and the operator gesticulation with the Oculus Rift and the Touch Controllers are superimposed in the top of the screen.

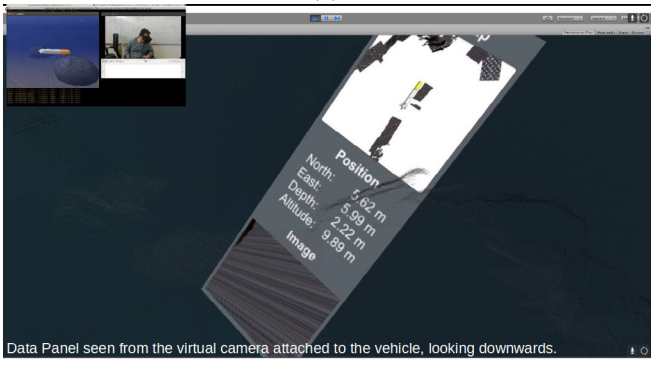
4. CONCLUSION

This paper has presented a new immersive platform to teleoperate underwater robots, based on Oculus Rift and Oculus Touch Controllers. This platform endows the pilot with additional features that can not be offered by traditional ROV consoles, such as the possibility to command the vehicle inside a virtual replica of the vehicle surroundings, as the pilot was immersed in it. Several experiments including teleoperation and visualization of data pre-recorded by an AUV in several marine scenarios, show the feasibility of our proposal in terms of perception, navigation and control. This utility is not only designed for ROVs control but also to supervise the evolution of AUVs and their surroundings, if a permanent network connection is feasible. Ongoing work includes the transformation of the data panel into a real *Head-Up Display* (HUD) in such a way that its position on the virtual scene keeps constant regardless of the virtual camera direction. Forthcoming work includes: a) Changing the Point Clouds resolution and their accumulation



(a)

(b)



(c)

Fig. 7. Teleoperation. (a) View from the external virtual camera, (b)-(c) View from the virtual camera attached to the vehicle.

strategy in Unity to obtain more realistic 3D models, b) 3D map optimization, in terms of trajectory drift, 3D-points matching in areas with overlap and texture blending also in zones with overlap; at this moment, all these issues are out of the scope of this paper. Other instruments could be added in the virtual scene to increase the application functionality, such as controlling a TMS or a robotic arm attached to the vehicle. Since the source code is still under development, it is available on a cloud repository under demand to the authors.

REFERENCES

Bonin-Font, F. (2017a). SPARUS II Teleoperation from a VR Environment. URL https://youtu.be/TJr4d7-anqg.

Bonin-Font, F. (2017b). Underwater Scene Perception from a VR Environment. URL https://youtu.be/I_6q0BJkRi0.

Carreras, M., Hernández, J., Vidal, E., Palomeras, Ribas, D., and Ridao, P. (2018). Sparus II AUV - A Hovering Vehicle for Seabed Inspection. *IEEE Journal of Oceanic Engineering*, 43(2), 344–355.

Carvalho, F., Dunker, P., Motta, T., Albuquerque, E., Raposo, A., Santos, I., and Galassi, M. (2015). SimVR-Robotics: A Virtual Reality Tool for Evaluating and Planning the Use of Robots in Offshore Scenarios. In *Proceedings of XVII Symposium on Virtual and Augmented Reality SVR*.

Fishers, J. (2017). Remote Operated Vehicles. URL http://www.jwfishers.com/products/rov.html.

García, J., Patrao, B., Almeida, L., Pérez, J., Menezes, P., Dias, J., and Sanz, P. (2015). A Natural Interface for Remote Operation of Underwater Robots. *IEEE Computer Graphics and Applications*, 37(1), 34–43.

Guerrero, E., Bonin-Font, F., Negre, P.L., Massot, M., and Oliver, G. (2017). USBL Integration and Assessment in a Multisensor Navigation Approach for AUVs. In *20th World Congress of the International Federation of Automatic Control (IFAC World Congress)*. Toulouse, France.

Jasiobedzki, P., Se, S., Bondy, M., and Jakola, R. (2008). Underwater 3D Mapping and Pose Estimation for ROV Operations. In *IEEE Oceans*, 604–613. Quebec (CA).

Jenkin, M. (2017). Unity-ROS Bridge. URL https://github.com/michaeljenkin/unityros.

JSON (2018). JavaScript Object Notation. URL http://www.json.org/.

Monferrer, A. and Bonyuet, D. (2002). Cooperative Robot Teleoperation Through Virtual Reality Interfaces. In *Proceedings of the Sixth International Conference on Information Visualisation*.

Oculus (2018a). Oculus Rift. URL https://www.oculus.com/.

Oculus (2018b). OVR Input API. URL https://developer.oculus.com/documentation/unity/latest/concepts/unity-ovrinput/.

Palomeras, N., El-Fakdi, A., Carreras, M., and Ridao, P. (2012). Cola2: A Control Architecture for AUVs. *Journal of Oceanic Engineering*, 37(4).

PCL Library (2018). Downsampling a PointCloud using a VoxelGrid filter. URL http://pointclouds.org/documentation/tutorials/voxel_grid.php.

Prats, M., Perez, J., Fernández, J., and Sanz, P. (2012). An Open Source Tool for Simulation and Supervision of Underwater Intervention Missions. In *Proceedings of International Conference on Intelligent Robots and Systems (IROS)*, 25772582. URL http://www.irs.uji.es/uwsim/.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. (2009). ROS An Open Source Robot Operating System. In *ICRA Workshop on Open Source Software*.

Unity Technologies (2018). URL https://unity3d.com/es.

Viswanathan, V., Lobo, Z., Lupanow, J., von Fock, S.S., Wood, Z., Gambin, T., and Clark, C. (2017). AUV Motion-planning for Photogrammetric Reconstruction of Marine Archaeological sites. In *IEEE International Conference on Robotics and Automation (ICRA)*, 5096–5103.

Weidner, N., Rahman, S., Li, A.Q., and Rekleitis, I. (2017). Underwater Cave Mapping Using Stereo Vision. In *Proceedings of International Conference on Robotics and Automation (ICRA)*, 5709–5715.

WillowGarage (2018). ROS Bridge Suite. URL http://wiki.ros.org/rosbridge_suite.

Zhang, J., Li, W., Yu, J., Li, Y., Li, S., and Chen, G. (2015). Virtual Platform of a Remotely Operated Vehicle. In *Proceedings of the IEEE Oceans*, 1–5. Washington.