Real-time Point Cloud Transmission for Immersive Teleoperation of Autonomous Mobile Robots

Nunzio Barone
Politecnico di Bari
Bari, Italy
nunzio.barone@poliba.it

Walter Brescia
Politecnico di Bari
Bari, Italy
walter.brescia@poliba.it

Gabriele Santangelo
Politecnico di Bari
Bari, Italy
g.santangelo@studenti.poliba.it

Antonio Pio Maggio
Politecnico di Bari
Bari, Italy
a.maggio4@studenti.poliba.it

Ivan Cisternino
Politecnico di Bari
Bari, Italy
i.cisternino@studenti.poliba.it

Luca De Cicco
Politecnico di Bari
Bari, Italy
luca.decicco@poliba.it

Saverio Mascolo
Politecnico di Bari
Bari, Italy
saverio.mascolo@poliba.it

ABSTRACT

Autonomous mobile robots (AMRs) are increasingly deployed for remote inspection in various scenarios. While AMRs can perform missions autonomously, human intervention becomes necessary in cases where safety risks or potential damage to the robot arise. Since these platforms are often deployed in remote or potentially dangerous areas, physical access is not always available. In such situations, teleoperation provides an effective solution to resolve issues. The navigation stack of AMRs include obstacle avoidance and mapping functionalities that rely on the environmental feedback, typically acquired with stereo cameras which represent depth and color information using Point Clouds (PCs). In addition to obstacles mapping, PCs provide operators and algorithms with real-time environmental data improving situational awareness and supporting AI-based downstream tasks. However, the large volumes of data inherent to PCs pose challenges for real-time transmission. For this reason, dedicated transmission pipelines and data compression algorithms compatible with the limited computational capabilities of the on-board computer are required. This paper demonstrates such a scenario, equipping a differential drive ground robot with a stereo camera that produces PCs that are streamed to a remote operator wearing a standalone VR Headset. To accommodate the limited computational capabilities of the on-board computer and reduce bandwidth requirements, a distance-based filtering and quantization encoding is applied. Users will step in to assist the robot, with remote real-time teleoperation, when the AMR issues a *"cry for help"* signal during an autonomous mission.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
MMSys '25, March 31-April 4, 2025, Stellenbosch, South Africa
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1467-2/2025/03
https://doi.org/10.1145/3712676.3719263

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Human-centered computing** → *Virtual reality*; • **Computer systems organization** → **Robotics**.

KEYWORDS

Teleoperation, Mobile Robots, VR, Point Clouds, WebRTC

ACM Reference Format:
Nunzio Barone, Walter Brescia, Gabriele Santangelo, Antonio Pio Maggio, Ivan Cisternino, Luca De Cicco, and Saverio Mascolo. 2025. Real-time Point Cloud Transmission for Immersive Teleoperation of Autonomous Mobile Robots. In *ACM Multimedia Systems Conference 2025 (MMSys '25), March 31-April 4, 2025, Stellenbosch, South Africa.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3712676.3719263

1 INTRODUCTION

Autonomous Mobile Robots (AMRs), such as aerial and ground robots, are increasingly deployed in remote or hostile environments to perform several tasks. These robots leverage sensing layers, typically including stereo cameras and LiDARs, to observe and extract environmental information, enabling navigation and situational awareness. Despite their capabilities, the autonomous navigation stacks on board these robots can be faced with difficulties or fail to execute specific operations, often due to inappropriate navigation parameters or the complexity and danger of certain movements. In such scenarios, human intervention is required to mitigate risks, safeguard expensive assets, and ensure mission success. Teleoperation is a practical solution, particularly for robots deployed in inaccessible or dangerous areas. Teleoperation can also be used for managing large fleets of autonomous robots leased to customers globally for industrial applications, enabling remote handling of failures and reducing response times and operational costs.

Teleoperation requires addressing several challenges. First, operators need optimal spatial awareness of the robot's environment to efficiently teleoperate the robot. Head Mounted Displays (HMDs), paired with volumetric video, provide an immersive solution, offering detailed visual representations of the surroundings and distance

information from obstacles. However, volumetric video – typically represented using meshes or Point Clouds (PCs) – poses several challenges due to its high bandwidth requirements and for this reason compression techniques need to be employed [16]. Additionally, the limited computational resources of mobile robots exacerbate the difficulty of providing short glass-to-glass latencies required for efficient teleoperation.

This work demonstrates an immersive platform to allow operators to teleport into remote locations and take control of connected mobile robots equipped with stereo cameras streaming PCs[1]. The operator will use a VR headset, f.i., the Meta Quest 3, to explore the 3D scene received in real-time from the mobile robot through a wireless link. By using dedicated controllers or hand gestures, the operator can both remotely control the mobile robot in different modalities, and navigate and manipulate the 3D environment exploiting different points of view to enhance safety of operations.

To implement the proposed system we make the following contributions. First, we design a capture to rendering pipeline that relies only on the robot's on-board computer and the operator's standalone VR headset. To reduce data latency and meet the real-time requirements we transmit point clouds, telemetry, and commands data using WebRTC. To address the challenge of time-varying bandwidth conditions, we develop a distance-based filtering and quantization encoding methodology that adapts the data stream bitrate dynamically, allowing teleoperation in constrained network environments. On the operator side, we implement a WebXR standalone application for the VR headset. This application includes a graphical user interface (GUI) that allows operators to monitor and interact with the selected robot seamlessly.

The rest of this paper is organized as follows. Section 2 provides background material. Section 3 describes our system in terms of hardware and software components, highlighting some technical difficulties inherited by our robotic and standalone setup. In Section 4 we describe the demo experience presented to the users. Finally, in Section 5, we summarize the key insights provided by the demo, discuss potential future adaptations for improvements, and outline future research directions.

2 BACKGROUND

The sensing stack of AMRs employs stereo cameras and LiDARs to generate point clouds, providing rich environmental data for navigation and situational awareness. This data is employed for autonomous navigation, as it allows robots to map obstacles, plan trajectories, and make decisions in dynamic environments. However, teleoperation scenarios require real-time transmission of this data to a human operator, which poses significant challenges due to the large size of raw point clouds.

In fact, a single point cloud frame at a 480×480 resolution can have a size of up to 29.5 Mbits (see Section 3.3.1), meaning that delivering raw point clouds at a 15 fps framerate would require a bandwidth of approximately 442 Mbps. Such requirements are unrealistic, particularly on mobile networks [2, 15]. To address this, adaptation and compression techniques must be employed to reduce the data size while preserving critical information.

Unlike typical volumetric streaming applications, such as teleconferencing applications [3], robot teleoperation is constrained by the limited computational power of on-board computers due to strict power consumption and mobility requirements. This restriction prevents the use of high-performance computing devices available on high-end workstations, significantly narrowing the range of feasible solutions for adaptation and encoding. Consequently, the algorithms employed must be as lightweight as possible to meet real-time operational constraints.

Additionally, because the robot operates in a real-world environment, the data transmission pipeline must be designed to provide a low-latency communication service. The specific requirements of the communication pipeline depend on the target control application.

In particular, two different scenarios of Human Remote Control (HRC) can be defined [1]:

(1) *Indirect HRC*: The operator provides high level setpoints related to the robot's pose (e.g., position, orientation). This control mode is useful for assisting the robot with global trajectory planning and can tolerate delays of up to 300 ms, as the robot still relies on its navigation stack to carry out missions autonomously [1, 13].
(2) *Direct HRC*: The operator directly controls the robot's actuators (e.g., velocities and accelerations). In this case, the operator acts as a local planner, requiring interaction frequencies that cannot tolerate delays beyond 150 ms [1, 13].

Existing 3D point cloud compression techniques, such as octree representations [6, 10], k-d tree structure [11], geometrical approaches [4], and machine learning approaches [9, 14], have been extensively studied. However, these methods often demand significant computational resources and power, making them unsuitable for the constrained environments of mobile robots.

3 SYSTEM DESCRIPTION

The proposed system is composed of a dedicated hardware and software stack. In this section, descriptions will be provided along with a performance analysis of technologies such as Unity and Draco in our resource-constrained computing units.

3.1 Hardware components

The hardware platform, depicted Figure 1, consists of the robot, the user equipment, a router, and a web server.

The robot is a *Turtlebot3 Waffle Pi*[2], an open-source and open-hardware robot platform designed for education, research, and prototyping. It is widely used in robotics due to its modular design, affordability, and ease of use. The robot's main equipment is composed of a stereo camera and a companion computer. The chosen camera is a *ZED2i* from Stereolabs, an RGB-D camera equipped with internal accelerometer, gyroscope, magnetometer, barometer, and temperature sensor. The companion computer is an *NVIDIA Jetson Xavier NX 16 GB* with a 512 GB SSD storage device which is

[1] In this work, we focus on PCs instead of meshes because such data are inherently used in robotics navigation stacks, making them suitable for integration into teleoperation systems.

[2] https://www.robotis.us/turtlebot-3-waffle-pi/

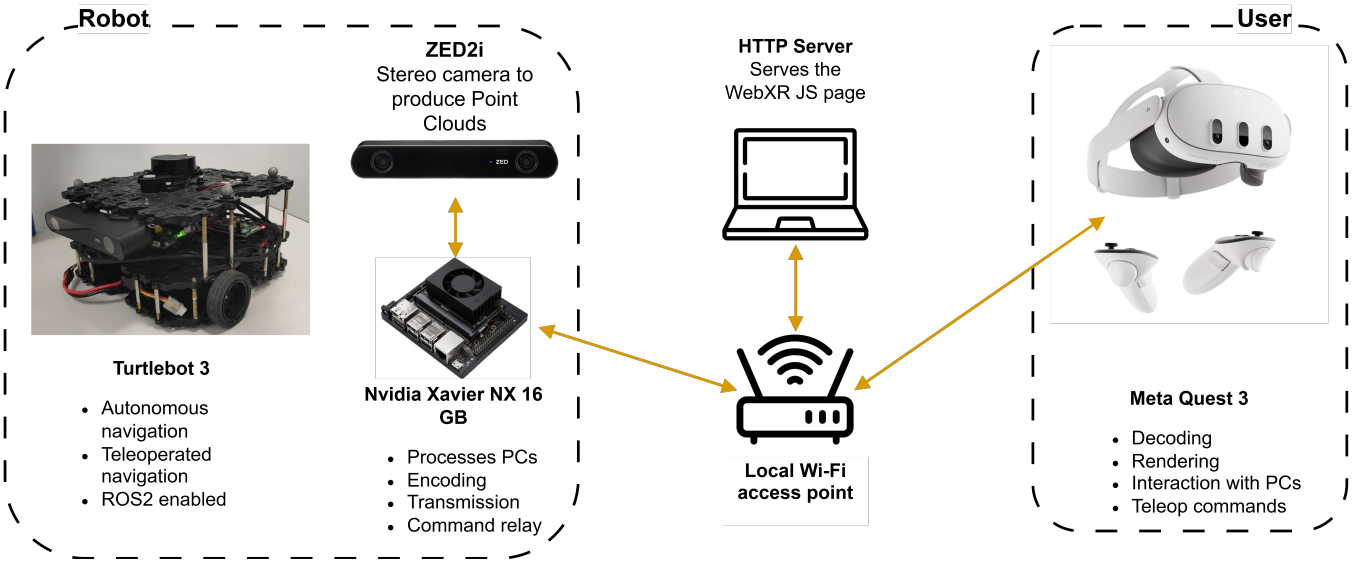


Figure 1: Hardware setup of the proposed system which includes: Robot with all its equipment, Network devices, User equipment consisting in the VR headset, and a server hosting the JS page with the code for the GUI

particularly suitable for mobile platforms due to its compact size and advanced computing performance.

The user equipment mainly consists of the HMD, specifically the *Meta Quest 3*, and the controllers, whose commands are sent to the robot based on the selected teleoperation modality. We deliberately chose this HMD due to its affordability and widespread availability. Moreover, its onboard computation capabilities enable the application to run in standalone mode, unlike other HMDs that required tethering to an external computer.

The web server used in the demo runs on a laptop and serves the WebXR JavaScript page. As a basic web server, it does not perform any computation on the streamed data. Its main function is to provide the JavaScript application to all headsets connected to the network.

3.2 Resource-constrained hardware performance

Having to meet low-power and small-size requirements, the computing capabilities of companion computers on board the robots are significantly limited compared to those of general-purpose laptops or workstations. This reduces performance, limiting the tools and algorithms that can be executed in real-time.

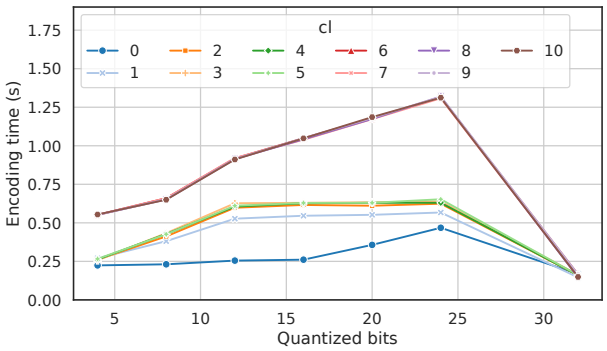
3.2.1 Draco. Among the most popular technologies for point cloud compression, Draco [7] stands out, with the promise of real-time performance for large PCs [5]. In particular, Draco codec has two configurable parameters for PC compression:

- **Quantization parameter (*qp*):** a value ranging from 0 to 30, this parameter sets the number of bits that Draco employs to quantize the positions of the points in the PC. Notice that, according to the Draco documentation when *qp* is set to 0 no quantization is applied, meaning positions are represented using 32 bits. To avoid confusion, instead of reporting the quantization parameter, we show the *quantized bits* (*qb*), which represent the actual number of bits used to encode positions in the compressed PC.
- **Compression level (*cl*):** a value ranging from 0 to 10 that selects different compression features. Typically, a larger value of *cl* should correspond to a lower size of the compressed PC at the expense of a higher encoding latency.

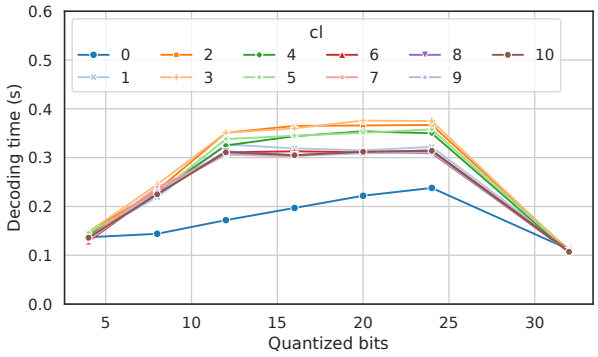
We tested all the possible combinations of these parameters on the reference hardware (*Nvidia Xavier NX*) using a reference point cloud of 810077 points, logging encoding and decoding duration and saved space percentage[3]. Results are presented in Figures 2 and 3. Figure 2(a) shows that the shortest encoding time is 150 ms, achieved when no quantization is applied, i.e., when points are represented using 32 bits ($qb = 32$) regardless of the compression level. Similarly, decoding times, shown in Figure 2, follow the same trend, with a minimum value of approximately ~100 ms, also observed for $qb = 32$. Consequently, the lowest total latency for encoding and decoding is around 250 ms. Adding the point cloud grabbing time, transmission time, and rendering time would result in a total latency exceeding the target maximum for both indirect and direct HRC. Therefore, despite achieving a data size reduction of approximately 68%, Draco is unsuitable for this scenario.

3.2.2 Unity and WebXR. The Meta Quest 3 supports several frameworks for developing immersive interfaces, including Unity [18] and WebXR [8]. Unity is a widely used high-performance game engine, that supports VR and XR applications. On the other hand, WebXR is browser-based framework that employs the `Three.js` library [12]. We experimentally compared Unity and WebXR rendering PCs with increasing sizes. As shown in Figure 4, WebXR with

[3]Notice that, since Draco did not satisfy the maximum latency requirements, we did not measure the distortions in PCs using metrics such as the chamfer distance.



(a) Encoding time



(b) Decoding time

Figure 2: Draco Encoding and Decoding time varying *qb* and *cl* parameters on a Nvidia Xavier NX

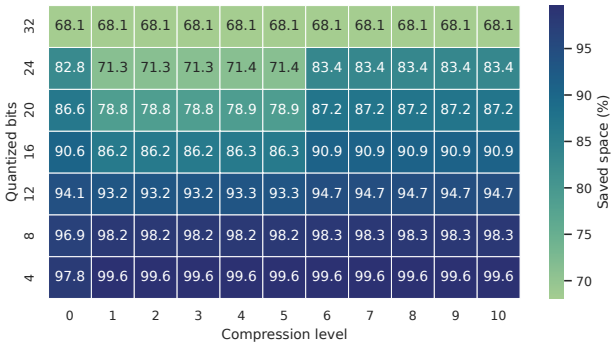


Figure 3: Saved space percentage using Draco codec varying *qb* and *cl* parameters

`Three.js` performs better, maintaining frame rates above 100 fps for point clouds containing up to 2 millions points.

Based on these results, we decided to forgo the use of Draco and Unity in favor of a custom distance-based point cloud filtering approach and the development of the operator GUI using WebXR.

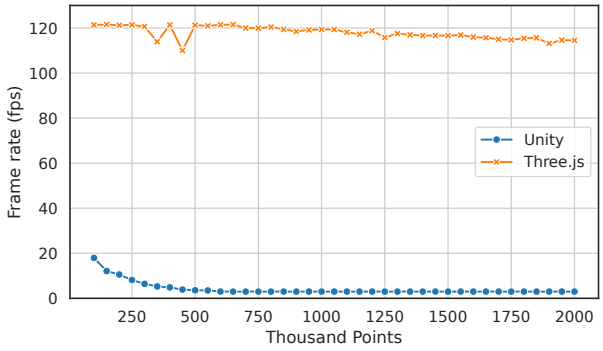


Figure 4: Comparison of rendering frame rate of a test PC with increasing size in Unity and WebXR

3.3 Software stack

The software stack, depicted in Figure 5, was designed and implemented to address the hardware's resource constraints while leveraging its strengths wherever possible.

3.3.1 Robot side. PCs are acquired using the ZED SDK at the desired resolution, which in our case is 480×480. The resulting object is a matrix with dimensions $480 \times 480 \times 4$, containing X, Y, Z coordinates and RGBA in `Float32` for each point, reaching a size of 128 bits per point [17]. Tensors are pre-allocated on GPU memory to further optimize performance. The point cloud matrix is converted to a GPU tensor with a negligible computational overhead exploiting the unified memory of the Nvidia Jetson Xavier NX. The matrix is flattened after outliers are filtered using a mask Quantization: X, Y, Z coordinates converted to `int16` and RBGA to two `uint16` (RGB + alpha). This allows to pass from a size of 128 bits to 80 bits per point. Thus, a compressed point cloud has a size of $480 \times 480 \times 5 \times 16 = 18.4$ Mbits. After this first quantization step, a filter based on distance is applied depending on the available bandwidth. Additional points are uniformly decimated if the distance filter is not enough. The compressed and filtered point cloud is then sent via a WebRTC data-channel to the other peer. Besides the PC transmission pipeline, the robot executes its autonomous routines using ROS2 packages. In particular, the *Nav2 package* is used to carry out the autonomous navigation. This set of tools takes advantage of sensors data to build a representation of the surroundings. This representation, typically a (2D or 3D) map, is used to identify the best trajectories that enable the robot to reach the desired position autonomously while avoiding obstacles. To avoid obstacles, a local cost map is produced. Depending on the selected thresholds the local path planner can make the robot behave either cautiously or recklessly. Then, a dedicated node has the task of bridging ROS2 network with the WebRTC peer, sending telemetry and receiving teleoperation commands.

3.3.2 User side. The received byte stream is decoded into 3D coordinates (positions) and R, G, B color values. The decoded data are used to update the 3D geometry for rendering. The `Three.js` library is employed with `BufferGeometry` for efficient geometry representation. The positions and colors obtained during decoding

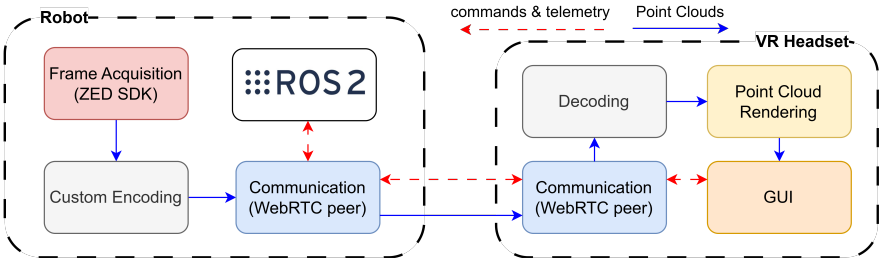


Figure 5: Software stack, representing point clouds (blue arrows) and Telemetry & Control (red dashed arrows) pipelines



Figure 6: Example of navigation scenario. On the left a frame of the volumetric video from robot's PoV. On the right a picture of the navigation environment

are set as geometry attributes. These attributes are then utilized in the WebGL rendering pipeline to render the geometry onto the canvas. A custom `ShaderMaterial` has been implemented in `Three.js`, written in OpenGL Shading Language (GLSL). In addition to PC visualization, the system features a GUI with floating buttons, telemetry banners, and notification flags. This interface enables switching between robots, points of view, and operational modes.

4 DEMO EXECUTION

The proposed demonstration replicates a typical teleoperation scenario, such as the one depicted in Figure 6, involving two robots: the Turtlebot and, for logistic reasons, a stereo camera connected to a laptop acting as a second robot. The user will be able to seamlessly switch between the two robots to experience different situations. These scenarios are designed to showcase the capabilities of the teleoperation system under both ideal and constrained network conditions.

4.1 Unperturbed operational flow

In this scenario, bandwidth is not constrained, allowing the maximum QoE. The following tasks are demonstrated:

(1) **Turtlebot autonomous navigation:** While the robot executes autonomous navigation tasks, the user can view the environment from the robot's PoV or freely explore the mapped area. By manipulating their PoV, users can rotate or zoom into specific parts of the scene using simple hand gestures. The autonomous tasks may be either user-defined (indirect HRC) or autonomously planned by the robot.
(2) **Switching to the other robot:** The system allows the user to monitor different robots. For demonstration purposes, the second "robot" is represented by a stereo camera connected to a laptop. In this case, the user will still be able to move around the mapped scenario, changing the PoV in the same way as the real robot.
(3) **Turtlebot Crying for Help:** During navigation, the Turtlebot may encounter a challenging situation, such as a narrow passage considered too risky to cross based on its navigation stack. When this occurs, the robot will stop and issue a *cry for help*, which is notified to the user via the GUI. This situation is artificially induced by modifying the local cost map, simulating an increase in perceived risk for crossing certain areas. This feature will demonstrate how the teleoperation system can intervene to address limitations in the robot's autonomous capabilities.
(4) **Turtlebot teleoperation:** Upon receiving a "Cry for help" notification, the user will take manual control of the robot, guiding it through the challenging passage. This interaction showcases the system's capability to switch from autonomous navigation to direct teleoperation seamlessly, allowing the robot to complete its mission.

4.2 Perturbed operational flow

In this scenario, different bandwidth constraints are introduced to simulate realistic fluctuations of the available network bandwidth. A dedicated dashboard enable users to adjust system parameters to optimize performance under these conditions. The goal is to demonstrate the system's adaptability and its impact on teleoperation performance and QoE.

Users will be able to adjust the following settings to explore the system's capabilities under constrained conditions:

- **Compression mode selection:** User can choose from the available compression methods and tune their parameters to balance quality and latency.
- **Minimum rendering distance guaranteed:** This parameter ensures that a certain visible range is maintained, regardless of bandwidth limitations, to prioritize critical spatial awareness.
- **Enable/Disable fallback to indirect HRC:** If latency requirements for direct teleoperation are not met, the system can automatically switch to indirect HRC, depending on the user's preference.
- **Enable/Disable maximum velocity adaptation:** If enabled, the robot's maximum velocity is dynamically adjusted

based on the rendering distance to maintain safe and effective operation.

The perturbations that can be applied mainly consist on:

- **Costmap threshold updates**: modifications to the robot's cost map can trigger *Cry for help* messages, requiring user intervention.
- **Bandwidth fluctuations**: The available bandwidth is dynamically varied, impacting the rendering distance and overall latency. The system adapts by modifying the robot's operational mode or maximum velocity as per the user's preferences.

5 CONCLUSIONS

In this paper, we propose a demo of immersive human teleoperation for a wheeled mobile robot, highlighting the limitations imposed by its resource-constrained hardware and exploring alternative solutions. In the demo the users will have the possibility to experience spatial awareness of the area where the robot is located and control it from different PoVs in real-time.

Future research directions include: i) the enhancing the bandwidth adaptation algorithm, by considering semantic information from the scene to allow non-uniform decimation, thus improving visual quality and reducing latency; ii) conducting a QoE assessment campaign to evaluate the impact of visual quality and latency on immersive mobile robot teleoperation.

ACKNOWLEDGMENTS

This work has been funded by the European Union through the DIANE project winner of the Open Call n.1 of the SPIRIT project (101070672).

REFERENCES

[1] 5GAA. 2020. Tele-Operated Driving (ToD): System Requirements, Analysis and Architecture. https://5gaa.org/tele-operated-driving-tod-system-requirements-analysis-and-architecture/ Accessed: 2025-01-18.

[2] Nunzio Barone, Walter Brescia, Saverio Mascolo, and Luca De Cicco. 2024. APEIRON: a Multimodal Drone Dataset Bridging Perception and Network Data in Outdoor Environments. In *Proceedings of the 15th ACM Multimedia Systems Conference* (Bari, Italy) *(MMSys '24)*. Association for Computing Machinery, New York, NY, USA, 401–407. https://doi.org/10.1145/3625468.3652186

[3] Matthias De Fré, Jeroen van der Hooft, Tim Wauters, and Filip De Turck. 2024. Scalable MDC-Based Volumetric Video Delivery for Real-Time One-to-Many WebRTC Conferencing. In *Proceedings of the 15th ACM Multimedia Systems Conference*. Association for Computing Machinery, New York, NY, USA, 121–131.

[4] O. Devillers and P.-M. Gandoin. 2000. Geometric compression for interactive transmission. In *Proceedings Visualization 2000. VIS 2000 (Cat. No.00CH37145)*. IEEE Computer Society, USA, 319–326. https://doi.org/10.1109/VISUAL.2000.885711

[5] M. De Fré, F. De Turck, and J. van der Hooft. 2023. *Low-Latency Volumetric-Video Delivery for Real-Time Conferencing*. Master's thesis. Ghent University. http://lib.ugent.be/catalog/rug01:003150149 Master Thesis Dissertation.

[6] Tim Golla and Reinhard Klein. 2015. Real-time point cloud compression. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE Computer Society, Hamburg, Germany, 5087–5092. https://doi.org/10.1109/IROS.2015.7354093

[7] Google. n.d.. Draco - A Compression Library for 3D Graphics and Point Clouds. https://google.github.io/draco/

[8] Immersive Web Community Group. n.d.. WebXR Device API. https://github.com/immersive-web/webxr.

[9] Bo Han, Yu Liu, and Feng Qian. 2020. ViVo: visibility-aware mobile volumetric video streaming. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking* (London, United Kingdom) *(MobiCom '20)*. Association for Computing Machinery, New York, NY, USA, Article 11, 13 pages. https://doi.org/10.1145/3372224.3380888

[10] Yan Huang, Jingliang Peng, C.-C. Jay Kuo, and M. Gopi. 2008. A Generic Scheme for Progressive Point Cloud Coding. *IEEE Transactions on Visualization and Computer Graphics* 14, 2 (2008), 440–453. https://doi.org/10.1109/TVCG.2007.70441

[11] Erik Hubo, Tom Mertens, Tom Haber, and Philippe Bekaert. 2006. The Quantized kd-Tree: Efficient Ray Tracing of Compressed Point Clouds. In *2006 IEEE Symposium on Interactive Ray Tracing*. IEEE Computer Society, Salt Lake City, UT, USA, 105–113. https://doi.org/10.1109/RT.2006.280221

[12] Ricardo Cabello (mrdoob) and Contributors. n.d.. Three.js - JavaScript 3D Library. https://github.com/mrdoob/three.js.

[13] Oren Musicant, Assaf Botzer, and Shraga Shoval. 2023. Effects of simulated time delay on teleoperators' performance in inter-urban conditions. *Transportation Research Part F: Traffic Psychology and Behaviour* 92 (2023), 220–237. https://doi.org/10.1016/j.trf.2022.11.007

[14] Dat Thanh Nguyen, Maurice Quach, Giuseppe Valenzise, and Pierre Duhamel. 2021. Learning-Based Lossless Compression of 3D Point Cloud Geometry. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE Computer Society, Toronto, ON, Canada, 4220–4224. https://doi.org/10.1109/ICASSP39728.2021.9414763

[15] Darijo Raca, Dylan Leahy, Cormac J. Sreenan, and Jason J. Quinlan. 2020. Beyond throughput, the next generation: a 5G dataset with channel and context metrics. In *Proceedings of the 11th ACM Multimedia Systems Conference* (Istanbul, Turkey) *(MMSys '20)*. Association for Computing Machinery, New York, NY, USA, 303–308. https://doi.org/10.1145/3339825.3394938

[16] Sebastian Schwarz, Marius Preda, Vittorio Baroncini, Madhukar Budagavi, Pablo Cesar, Philip A Chou, Robert A Cohen, Maja Krivokuća, Sébastien Lasserre, Zhu Li, et al. 2018. Emerging MPEG standards for point cloud compression. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 1 (2018), 133–148.

[17] Stereolabs. n.d.. Stereolab ZED SDK - Using the Depth Sensing API. https://www.stereolabs.com/docs/depth-sensing/using-depth#getting-point-cloud-data

[18] Unity Technologies. n.d.. Unity - Game Engine and Development Platform. https://unity.com/.