# Real-Time interactive simulation and visualization framework in AR for robotic systems based on ROS

Jonas Frey
Zürich Switzerland
jonfrey@ethz.ch

Raffael Theiler
Zürich Switzerland
ratheile@ethz.ch

Turcan Tuna
Zürich Switzerland
tutuna@ethz.ch

Eric Vollenweider
Zürich Switzerland
evollenwe@ethz.ch

## Abstract

*Augmented Reality (AR) bears great potential in the field of robotics. Despite the maturity of both AR and robotics, there is still a large gap in between them for researchers to bridge. Commonly used tools and frameworks currently do not cooperate very well. Given the recent increase in robotic projects involving Augmented Reality components, there is a dire need for an intuitive infrastructure to fuse typically used development tools and standards from both domains.*

*In this paper, we present a framework built for the robotic community that enables (pre-existing) Robot Operating System (ROS) based projects to more easily utilize Augmented Reality components simulated in the game engine Unity. Our framework allows real-time virtual robot simulation, data visualization, sensor simulation and a dynamic component management. The framework is extendable, customizable, and available as Open Source.*

## 1. Introduction

Recent breakthroughs in computational technology, computer vision and manufacturing allow the efficient use of head-mounted displays (HMDs) and Augmented Reality (AR) to assist operators in performing their tasks. Especially in the context of robotics, current interdisciplinary developments utilizing AR seem very promising. Mixed Reality (MR) based systems have three crucial features, namely **(1.)** combining the virtual and real world together, **(2.)** real-time interaction and **(3.)** providing a mapping between real and the virtual worlds to create realistic interactions [1]. Example use-cases can be found in the fields of medicine [2, 3], robotics, teleoperation, civil engineering, arts, gaming, robotics and various others domains where additional information or new means of interaction, such as gesture control, are beneficial [4]. AR technology has real potential to improve human life and will have a great impact on the economy [5].

The field of robotics is and will be influenced by MR, since their combination bears great potential during development, testing and operation phases. Given the maturity of both disciplines, it is surprising that there still is a big gap between them. Better tools bridging this gap between AR and robotics are needed, to allow robotic engineers and researchers to exploit the full potential of both technologies.

Previous work has shown that there is a high unmet demand for establishing connectivity between AR and ROS. A flexible all-compatible bridge between the most common tools of both communities lacks.

In this paper, a ARbotics framework is presented, that first and foremost facilitates the development of AR components in the robotic context by providing a configurable infrastructure and features for communication, robot simulation & rendering, data visualization and sensor simulation. The ability to customize and extend the provided tools was prioritized throughout the development. It allows bidirectional real-time connectivity between a ROS network and the HMD, while providing a feature-rich infrastructure for easy setup and efficient deployment. The target audience of our framework is researchers and engineers in the robotic domain who want to leverage the benefits of AR for their projects. The full source code including example projects using the Microsoft HoloLens 1&2 are available on GitHub. [1]

The contributions of each project member is given as follows: **Eric Vollenweider**: Development of Unity-Side backbone & messaging, extending ROS# (AOT buildable & state/robot synchronization); **Raffael Theiler**: HoloLens deployment & testing, development of the data visualization tools, usability testing, architecture expert.; **Turcan Tuna**: Development of the sensor simulation and parameter configuration; **Jonas Frey**: Off-Device state manager & request processing, ROS packages, motion planning, configurator and container management.

[1] https://github.com/EricVoll/ARbotics

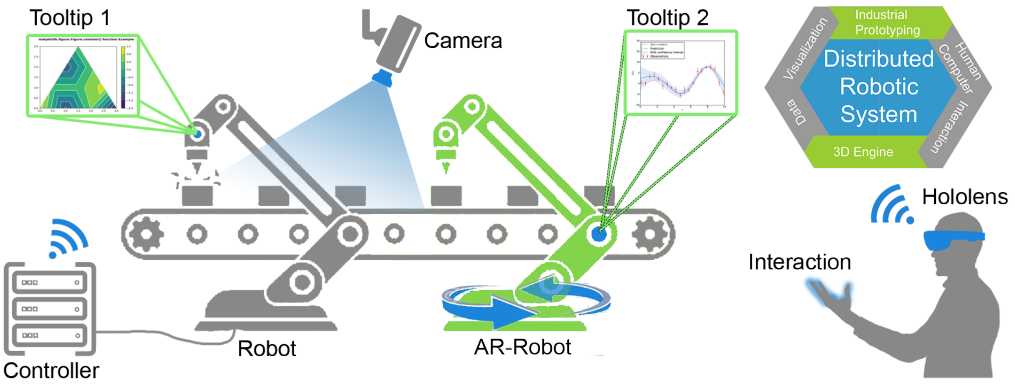


Figure 1. **Use Case Illustration**: The operator wearing a HoloLens (right) commissions a new robot at an existing factory (grey). Data about the system in production is displayed to the operator (Plot 1 left). The field of view of the ceiling mounted camera is overlayed to the real world (blue). The new robot is simulated and illustrated by a hologram (green). Simulated data is available to the operator (Plot 2 right). The operator can change the position and type of robot. Robot trajectories can be tested by the operator while taking workspace, safety zone and singularity constrains into account. The operator can interact with the complete system by intuitive gestures.

## 2. Related Work

In this section, previous works on combining robotics and MR-HDM technology are investigated. Firstly, in 2014 [6] WebSockets were used to establish data transfer between ROS and Unity3D environments for an immersive teleoperation application. The WebSockets are created utilizing the rosbridge [7] library. Furthermore, in [8] a high-fidelity Multi-UAV navigation and control simulator is developed in ROS and Unity. The data transfer of sensor and navigation data is established in the same manner as in [6]. In 2016, [9] introduced ROSUnitySim a real-time experimentation simulator for multi-unmanned aerial vehicles. To accomplish this a costume TCP/IP WebSocket implementation to transfer data between ROS and Unity was developed. They were able to stream LiDAR data at a rate of 40Hz between ROS and Unity.

An industrial robot use case of AR is presented in [10]. The authors showed that augmented reality can be used to track a welding robot with an in Unity simulated AR robot. Instead of using the well-established Gazebo simulator, they used a tailored $C\#$ inverse kinematics solver to simulate the AR robot in Unity. Therefore they did not make use of the full infrastructure provided by ROS regarding path planning and robot simulation. In [11] an extensive simulator for human-machine interaction was developed by combining Unity and ROS. Their simulator allows an operator to dive into VR and interact with robots that are simulated in ROS. To establish the connectivity between ROS and Unity they used similar to previous work Websocket (JSON), but additionally utilized BSON message transportation to meet real-time requirements for camera image transfer. In [12] the authors investigated diagnosing, teaching and patching interpretable knowledge of a robot to an operator. For this, they also relied on ROS Unity and implemented a TCP connection between ROS and Unity. The used communication library is publicly available but tailored to their specific task and therefore not easily extendable to other scenarios or robots. The authors of [13] tackled the challenge of creating an immersive operator interface to control Multi-robot Systems by utilizing ROS and Unity. The dire need for a framework that is not only able to transfer data between ROS and Unity but additionally empowers Unity to interpret standard robot description formats was satisfied $ROS\#$ [14]. A robot described by its Universal Robot Description Format URDF can be linked by $ROS$ to a Unity Game Object. This functionality and the good documentation which eases the learning process of a new developer, led to fast acceptance in the robotic community. Based on this software module zhang2019imp showed that Unity can be used to virtually test computer vision algorithms. [15] mentions that even $ROS\#$ is a good effort on easing the communication between ROS and Unity it is lacking the ability to realize real-time data transfer due to using WebSocket for receiving commands.

Additionally, in recent years new containerization tools are emerging such as Docker [16]. Docker is an open-source project that allows application automation by containerizing multiple applications into isolated containers. The containers are executed on an underlying Linux kernel and a high-performance networking stack allows data transfer within the docker network and to external sources. A Docker container is stated to be a lightweight, less time and resource consuming solution than the common virtual machine alternatives [17, 18]. Moreover, robotics lacks an effective workflow for rapid prototyping, test verification and continuous integration [19]. Challenges such as hard integration, the unintuitive setup process can hinder the research process and Docker found to be helpful to these

problems [19–21]. Docker was used in several previous works [22, 23] to build up a ROS Network. Cross-platform compatibility and the deployment to multiple hosts are only a few reasons why currently a major shift to containerized applications and tools is present.

## 3. Design Consideration

In the following section, we will layout requirements that our ARbotics framework should provide. We derive design principles from those requirements as a foundation for the implementation of ARbotics.

### 3.1. Objectives

The requirements enclose two perspectives: Firstly, the functional requirements that the framework should deliver to the user[2]. Secondly, the non-functional requirements of the framework.

**Functional Requirements:**

1. Bidirectional communication between sensors, robots and augmented reality components.
2. Support for real and simulated sensors and robotic systems.
3. Soft real-time capability on state of the art HMDs.
4. Synchronization of a robotic system's state (configuration, location and joint-states) and its counterpart in AR.
5. Visualization of data in AR linked to physical objects.

**Non-Functional Requirements:**

1. Simplicity, ease of use.
2. Clear dependencies and transparent system architecture.
3. Extendability and usage of open standards.
4. Easy and quick setup (base functionality accessible out of the box).

### 3.2. Resulting Principles

Given the multilayered and complex requirements of our ARbotics framework, it is logical to set up the design principles in an adequate manner. To establish a strict project segmentation and full-fill the requirements of extendability and usage of open standards we will base our framework on **loosely coupled building blocks (P1)**. This allows us to select top-performing frameworks for each task. Given the targeted functionality and desired simplicity it is crucial to make use of **existing open standards and libraries (P2)**. The used libraries have to be selected carefully to avoid unnecessary or redundant dependencies. By embedding those tools that are most familiar to people in the robotics and AR domain we lower the entry barrier for new users.

To ensure adequate performance for soft real-time constraints a **distributed architecture (P3)** is preferable to shift heavy computing to adequate hardware. A wisely chosen **communication infrastructure (P4)** between (real & simulated) robotic systems and augmented reality components is necessary to meet the targeted real-time requirements. The functionality of individual components[3] should be **configurable and accessible by well documented APIs (P5)**. To allow the interplay of different components and enduser[4] interactions simultaneously the full configuration of the system state needs to be adjustable during runtime. Adjustable refers to starting and stopping components as well as configuring new ones. With this dynamic configuration, dynamic scenes including varying components can be created that make our framework suitable for a wide range of robotic applications.

Modularity and flexibility is met by **containerizing (P6)** the framework. It not only intends to shorten the configuration effort for a newcomer but also allows developers of robotic systems to bundle their work, removing that burden from a user. While virtualization has certain performance implications, it removes many operating system related obstacles that might appear when setting up complex projects on highly customized personal computer setups as we see them among researchers and engineers.

## 4. Method

To fulfil the defined requirements and to follow the derived design principles from section 3, the architecture shown in Fig. 2 was designed.

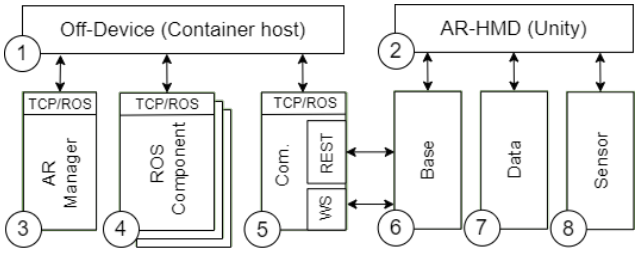


Figure 2. **Project architecture:** Fundamental building blocks [3-8] of ARbotics framework, sorted according to containerized Off-Devices (1) and unity applications running on the AR-HMD (2). For detailed architecture information we refer the reader to the text in the method section 4.

Since applications using this framework might perform computationally heavy tasks, principle P3 suggests building a distributed architecture, hence the framework is split into two main parts. First, there are the "Off-Device-Components" (Fig. 2 (2)) (Server or PC) running containerized applications (following P6). The framework's central manager, various ROS applications and user code are contained in this component. Secondly, there is the AR-HMD (Fig. 2 (3)) running Unity, which is mainly used for AR-I/O purposes (virtual robot rendering, user AR interaction,

[2]**User**: A researcher or engineer that utilizes our framework

[3]**Components**: Real or simulated robots, sensor and others

[4]**Enduser**: Person wearing the HMD

etc.). This setup also facilitates the fulfillment of principle (3), which requires soft real-time capabilities, since the AR-HMD device can run on high frame rates due to the off-loading computationally heavy tasks. The applications running on the container host system and the HMD communicate using the communication container (5), either using a REST API or a WebSocket, were the latter is established by a RosBridge.

The specific function of the mentioned components are the following:

1. Off-Device: Hardware that hosts containers for computationally costly tasks
2. AR-HMD: Standard Augmented Reality devices running Unity
3. AR Manager: A server accepting client requests, containing system configurations and managing the framework's containers. Launches & stops containers. Monitors & publishes the system's state
4. ROS Component: Container composed of ROS packages that launch ROS nodes (robots, sensors, ...)
5. Communication: Communication endpoint providing WebSocket and REST endpoints to communicate with the HMD, enabling communication between all components
6. AR.Base: Communication, URDF state synchronization, Hologram management
7. AR.Data: Interactive live data visualization
8. AR.Sensor: LiDAR (2D/3D) & Camera simulation, parameter adjustment, prototyping

With all individual components briefly explained, the following part will describe the information flow between the individual components. First, a user configures the capabilities of their system using the yaml format. Users can specify the content of containers with all ros packages to be launched, network settings, volume mounting, the urdf content describing the component, meta-information such as pretty names for front-end display, and native docker run-parameters.

```
- comp_type: ros
  docker:
    command: [...] roslaunch ur_gazebo ur3.launch
    environment: URI=http://ros1_base:11311
    image: ros1_ur_rossharp
    network: docker_rosnet
    volumes: /home/Mesh
  max_instances: 10
  pretty_name: UR3
  urdf:
    dyn: [current urdf : string]
    static: [Initial urdf : string]
```

It is important to state that the goal is not to replace a container management system such as Kubernetes. Instead, we aim to provide a simple and configurable handle for the user to access the native docker API and allow simple utilization of existing ROS packages.

Upon launching the application, the AR manager (3) reads the yaml-config, starts a WebServer (5), and offers the REST API to read and write data, following the client-server schema. A client-server implementation was chosen to allow multiple clients to request state changes simultaneously. Using a REST API intrinsically avoids concurrency issues given the stateless requests. Any client in the network can request information about the available components. A REST-PUT request, containing the name of the component to spawn, initiates the spawning process. The AR manager starts the requested container (4) and registers it in the framework's state when it is fully up and running.

As soon as the requested container is launched it begins to publish information on the ROS network. The AR.Base (6), which uses $ROS\#$ and our custom URDF parser, synchronizes the state of the AR-Scene with the incoming state of the AR-Manager. The AR.Base creates or removes virtual robots, which were added or removed from the current state and synchronizes the Unity-GameObjects with the current URDF of the edited component. This allows dynamically changing the robot's configuration, such as removing or adding tools, during run-time. All virtual components also subscribe to the joint states of their robots, such that the virtual AR robot behaves just like the real or simulated ones.

Our custom URDF parser offers an API to easily read custom XML tags. These tags are used to define the location of data sources, which are visualized using the AR.Data (7) component. Each tag defines an attachment point on the robot with the corresponding ROS topic, which contains the actual data. The AR.Data namespace offers an API to spawn and remove plots, which subscribe to specified topics and display the incoming data using AR plots. The data can currently be any kind of image, such as plots generated with the MatPlotLib or even images filmed with real cameras. These plots offer the end-user different functionalities such as collapsing, showing, zooming and starting a "follow me" mode.

Lastly, the AR.Sensor (8) namespace provides tools to simulate different ray-based sensors, such as cameras or LiDARs (2D & 3D). These sensors can be attached to any real-world object or to holograms such as the end-effector of a robot. The sensor moves with the exact component it was attached to, allowing the end-user to rapidly prototype different locations for the sensor. Additionally, the AR.Sensor component allows configuring the sensor's parameters. The simulated sensor data (images or point clouds) can be published to ROS-topics.

The user can extend the system by either creating custom docker containers based on our default template containing custom ROS code, or by quickly prototyping applications

using roslibpy [22]. Roslibpy is a lightweight Python library especially suitable for prototyping and allows the user to access the ROS network via the already established WebSockets. Similarly, users can add Unity scripts and adapt the application logic as needed.

### 4.1. Stateless Communication

The messages sent from the AR-Manager to the AR.Base component are stateless. This means, that rather than sending messages with the goal of performing differential actions, the AR Manager sends the complete state with every single update. The AR.Base component has to adapt the Unity scene to reflect the AR-Manager's state. The advantage of stateless communication is that the last message represents the complete information needed about the system and no historical data is needed. Assuming the Unity application was to crash, it would immediately restore the complete state after receiving the first update, allowing the end-user to continue working quickly.

## 5. Accomplishment

To demonstrate the capabilities of our framework we selected a complex environment with many obstacles, such as poles supporting a roof to mimic an industrial setting. We use the spatial mapping meshes (SMM) of MRTK to map the real world into AR space. We set the tessellation detail of the SMM to be coarse because of its performance implications.

Our sensors interact with both, virtual objects and the SMM. Two example sensors of our sensor-toolbox are provided in figures 3 and 4. Figure 3 illustrates a simulated camera with its frustum. Figure 4 shows a 2D LiDAR sensor at the same location with its monitored 2D plane and maximum reach. Both sensors' fields are ray-based and therefore collide with the environment. The sensors' specifications are configurable within the AR application for placement and calibration during prototyping.

ARbotics can be used for different prototyping tasks, which is demonstrated with the following example. Having a specific camera sensor in mind, the operator starts a simulation and sets the camera parameters. Figure 5 shows the preparation stage in which the operator monitors the joint position while adjusting the virtual sensor's position on the end effector. Given our high-performance image plotting, encoding, decoding and rendering pipeline multiple plots refreshing at **30 fps** can be displayed simultaneously. Plot holograms display data (e.g. the system's state variables) to the operator and visually link the data to their physical sources, if there is one (e.g. a joint's encoder value). These plots may be hidden until needed and can also follow the operator on demand, which is useful for larger systems. From figure 5 to 6, the operator moves the robot to a new 6DoF pose, indicated by the position of the target hologram. To perform this movement, the target pose is sent to the off-device robot simulation, where a motion plan is generated and executed by the simulated controller. The resulting joint angles are published to the AR-robot. The communication delay induced by the WebSocket data-transfer is in the magnitude of the usual WLAN 802.11b/TCP Round Trip Time of **10ms**. The delay is therefore not noticeable as an operator.

Due to our dynamic urdf syncing, the robot's configuration (urdf) can be manipulated during runtime, and the AR simulation will adapt. This means, that complete robots, robot parts or only smaller details can be changed dynamically.

In table 5 we provide a comparison between ARbotics and most relevant previous work. We evaluated to the best of our knowledge supported robotic systems (Robots), extendability (Ext.), dynamic scene adjustment (Dyn.), documentation (Docu.), user setup time (Time) and featured applications (Applications).

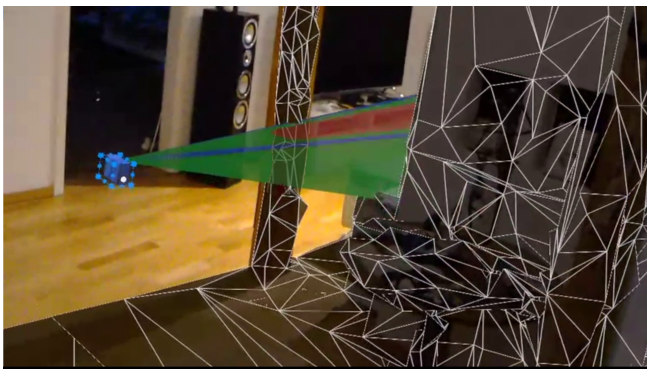

Figure 3. The generic camera sensor hologram with its visualization of basic parameters and indicated FoV. The frustum is a ray based simulation that highlights collisions with the real world and other holograms in red.

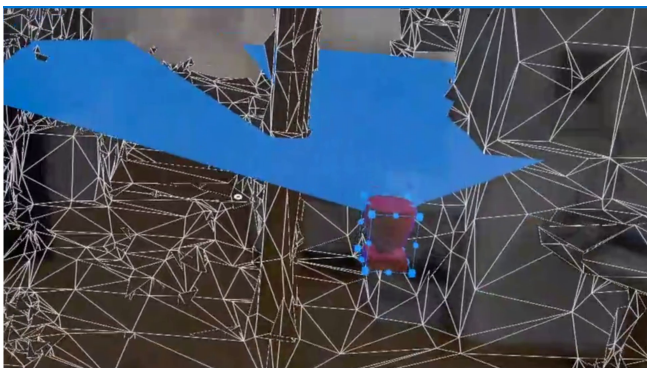

Figure 4. The generic 2D LiDAR sensor and its sensitive 2D plane. Rays collide with real world and virtual obstacles. Basic parameters featuring sensitivity plane radius, depth and resolution can be configured.

Table 1. Comparison of Existing Frameworks: supported robotic systems (Robots), Extendability (Ext.), Dynamic scene adjustment (Dyn.), Documentation (Docu.), user setup time (Time), featured applications (Applications).

| Framework | Robots | Communication | Ext. | Dyn. | Docu. | Code | Time | Applications |
|---|---|---|---|---|---|---|---|---|
| [6] | Mobile Robot P3-AT | WS (rosbridge) | ● | ● | ● | - | - | Teleoperation in VR |
| [8] | ROS | WS( rosbridge) | ● | ●● | ● | OpenSource | - | UAV simulatior |
| [11] | ROS | WS (rosbridge/BSON) | ● | ●●● | ●● | OpenSource | - | VR HMI simulator |
| [12] | Rethink Baxter | TCP (costum) | ● | ● | ●● | OpenSource | - | Data visualization |
| ROS# [24] | ROS | WS (rosbridge) | ●● | ●● | ●● | OpenSource | + 2h | VR/AR Robotics, Limited runtime capabable |
| Ours | ROS | WS (rosbridge) | ●●● | ●●● | ●●● | OpenSource | 30 min | VR/AR Robotics, Runtime capabable |

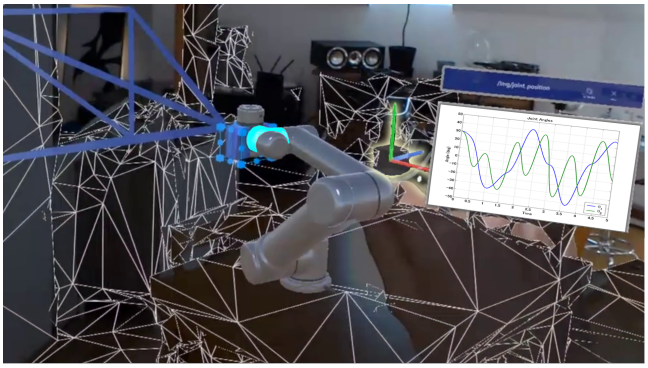

Figure 5. An ensemble of components used in a prototyping scenario. (left) A simple camera sensor attached to the end effector. (center) A UR5 robot next to the end effector target hologram (highlighted). (right) A plot that streams live data visualized with MatplotLib from an off device developer laptop via ROS. The encoder values originate from the highlighted joint (turquoise).

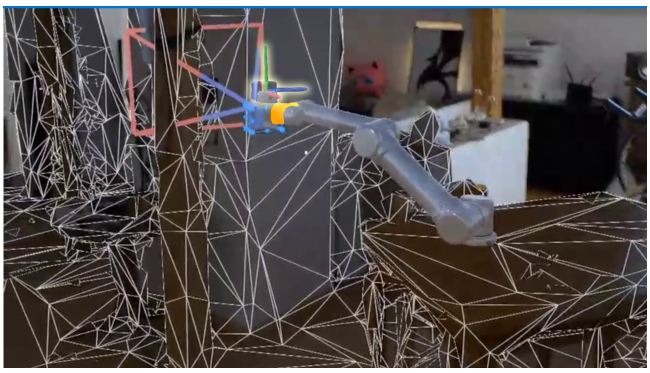

Figure 6. The extended arm after the robot planned and executed its motion. The interactive frustum shows that the selected camera's wide FoV leads to a recording of unnecessary data from the now close poles to the left and right of the gap. The orange color indicates that the plot is now hidden, but the user may interact with the joint to show it again.

## 6. Conclusion and Discussion

Until now, a fast adoption of AR technology to commercial robotic applications did not take place. One major factor is the limited hardware of currently available HMDs. Their lack of performance and comfort limits the possible real world use-cases substantially. Additionally, the high entry barrier to develop AR applications for robotic systems limits the number of interested and capable developers.

The upcoming second generation of HMDs, such as the HoloLens2, will increase the performance and usability compared to current HMDs significantly. The HoloLens2 allows to capture higher resolution meshes of the environment and provides a more intuitive gesture control, which opens a wide variety of possible use-cases. With our framework we lower the entry barrier for developers to utilize the full potential of AR in their projects. We successfully demonstrate that a useful sensor and robotic commissioning task can be achieved with very little effort. We have shown that robot path planning, data visualization and sensor & robot simulation can be achieved simultaneously, while providing an intuitive AR interface for the end user.

Our modular architecture allows to easily port the framework from to ROS1 to ROS2, which might be necessary in the coming years. One drawback that currently limits our framework is the necessity to import robot assets into Unity once before starting the application. This requires a recompilation of the AR application. This burden is acceptable but not optimal. Possible solutions are known, but not implemented due to time constraints.

Many of our design decisions were correct or at least pointed towards the right direction. From being easy to use by basing the framework on numerous open source components over to being future proof by building a modular system. Especially containerization has proven to be useful during the development but also during the deployment and operation of our test-runs.

Without doubt there is future work to be done on this framework. Namely moving from 2D to 3D plots to fully take advantage of AR, solving asset importation issues, improving the computational efficiency of 3D LiDARs and offering more intuitive & a wider variety of user interaction in the virtual space.

Anyhow, this field promises to be exciting for the years to come. We consider our initial goals as achieved and believe that our framework helps to overcome the problems given in the introduction. Special thanks to ETH Zürich for providing the MS HoloLens 1.

## References


[1] S. Rokhsaritalemi, A. Sadeghi-Niaraki, and S.-M. Choi, "A review on mixed reality: Current trends, challenges and prospects," *Applied Sciences*, vol. 10, no. 2, p. 636, 2020.

[2] T. Huber, E. Hadzijusufovic, C. Hansen, M. Paschold, H. Lang, and W. Kneist, "Head-mounted mixed-reality technology during robotic-assisted transanal total mesorectal excision," *Diseases of the Colon & Rectum*, vol. 62, no. 2, pp. 258–261, 2019.

[3] L. Chen, T. W. Day, W. Tang, and N. W. John, "Recent developments and future challenges in medical mixed reality," in *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 123–135, 2017.

[4] J. C. Cheng, K. Chen, and W. Chen, "State-of-the-art review on mixed reality applications in the aeco industry," *Journal of Construction Engineering and Management*, vol. 146, no. 2, p. 03119009, 2020.

[5] Z. Makhataeva and H. A. Varol, "Augmented reality for robotics: A review," *Robotics*, vol. 9, no. 2, p. 21, 2020.

[6] R. Codd-Downey, P. M. Forooshani, A. Speers, H. Wang, and M. Jenkin, "From ros to unity: Leveraging robot and virtual environment middleware for immersive teleoperation," in *2014 IEEE International Conference on Information and Automation (ICIA)*, pp. 932–936, IEEE, 2014.

[7] C. Crick, G. Jay, S. Osentoski, B. Pitzer, and O. C. Jenkins, "Rosbridge: Ros for non-ros users," in *Robotics Research*, pp. 493–504, Springer, 2017.

[8] W. Meng, Y. Hu, J. Lin, F. Lin, and R. Teo, "Ros+ unity: An efficient high-fidelity 3d multi-uav navigation and control simulator in gps-denied environments," in *IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society*, pp. 002562–002567, IEEE, 2015.

[9] Y. Hu and W. Meng, "Rosunitysim: Development and experimentation of a real-time simulator for multi-unmanned aerial vehicle local planning," *Simulation*, vol. 92, no. 10, pp. 931–944, 2016.

[10] E. Sita, C. M. Horváth, T. Thomessen, P. Korondi, and A. G. Pipe, "Ros-unity3d based system for monitoring of an industrial robotic process," in *2017 IEEE/SICE International Symposium on System Integration (SII)*, pp. 1047–1052, IEEE, 2017.

[11] Y. Mizuchi and T. Inamura, "Cloud-based multimodal human-robot interaction simulator utilizing ros and unity frameworks," in *2017 IEEE/SICE International Symposium on System Integration (SII)*, pp. 948–955, IEEE, 2017.

[12] H. Liu, Y. Zhang, W. Si, X. Xie, Y. Zhu, and S.-C. Zhu, "Interactive robot knowledge patching using augmented reality," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1947–1954, IEEE, 2018.

[13] J. J. Roldán, E. Peña-Tapia, D. Garzón-Ramos, J. de León, M. Garzón, J. del Cerro, and A. Barrientos, "Multi-robot systems, virtual reality and ros: developing a new generation of operator interfaces," in *Robot Operating System (ROS)*, pp. 29–64, Springer, 2019.

[14] M. Bischoff, "Ros sharp," *Ros Sharp Framework*, 2018.

[15] T. Inamura and Y. Mizuchi, "Sigverse: A cloud-based vr platform for research on social and embodied human-robot interaction," *arXiv preprint arXiv:2005.00825*, 2020.

[16] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[17] A. K. Yadav, M. Garg, *et al.*, "Docker containers versus virtual machine-based virtualization," in *Emerging Technologies in Data Mining and Information Security*, pp. 141–150, Springer, 2019.

[18] C. Pahl and B. Lee, "Containers and clusters for edge cloud architectures–a technology review," in *2015 3rd international conference on future internet of things and cloud*, pp. 379–386, IEEE, 2015.

[19] R. White and H. Christensen, "Ros and docker," in *Robot Operating System (ROS)*, pp. 285–307, Springer, 2017.

[20] R. White, "Ros + docker: Enabling repeatable, reproducible, and deployable robotic software via linux containers," 09 2015.

[21] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.

[22] R. Rust, G. Casas, S. Parascho, D. Jenny, K. Dörfler, M. Helmreich, A. Gandia, Z. Ma, I. Ariza, and M. Pacher, "COMPAS FAB: Robotic fabrication package for the compas framework." https://github.com/compas-dev/compas_fab/, 2018. Gramazio Kohler Research, ETH Zürich.

[23] R. White and H. Christensen, *ROS and Docker*, pp. 285–307. 05 2017.

[24] M. Bischoff, "ROS," June 2019.