



***The EU Framework Programme for Research and Innovation H2020
Research and Innovation Action***

CENTAURO

***Deliverable D4.3
Constructing simulated world from robot percepts***

Dissemination Level: Public

Project acronym: CENTAURO

Project full title: Robust Mobility and Dexterous Manipulation in Disaster Response by Fullbody Telepresence in a Centaur-like Robot

Grant agreement no.: 644839

Lead beneficiary: RWTH – Rheinisch-Westfaelische Technische Hochschule Aachen
MMI - Institute for Man-Machine Interaction

Authors: Torben Cichon

Work package: WP4 – Modeling and Simulation

Date of preparation: 2017-10-26

Type: Report

Version number: 1.2

Document History

Version	Date	Author	Description
1.0	2017-09-03	Torben Cichon	Initial version
1.1	2017-09-19	Torben Cichon	Minor updates
1.2	2017-10-26	Torben Cichon	Post integration

Executive Summary

In D4.3 we present the developments towards the predictive robotic system. Modeling a digital twin of the real system for direct control or visualization purposes is accompanied by static or dynamic environments to interact with. The digital twin represents the same integrated functionalities (especially with respect to the XBotCore developments in WP2) and the same interfaces, especially in terms of ROS, as the real robot. Additional sensor data processing, rendering and visualization is also part of this work package. These can then be used to dynamically generate environments based on the robot's percepts.

Contents

1	Introduction	5
2	Overview	6
3	Interfaces	8
4	Simulatable Robot Model	10
5	Visualization and Rendering	17
6	Simulatable Environment Model	23
7	Overall Setup	30
8	Conclusions	32
9	Appendix	32

1 Introduction

The 3D simulation system is the basis for the predictive robot model and the interaction of the operator with the look-ahead simulation (see Grant Agreement [1]):

On the operator's end, the physical simulation of the robot in its environment (developed in T4.2) must be updated from the percepts and actions of the robot. With the help of semantic information about the robot's surroundings such as, e.g., type of terrain, graspable objects, or accessible paths, the CENTAURO robot creates a complete representation of every environmental detail available to it. The result is stored in the CWM (see T4.1) and must be processed fully automatically for later use in the predictive robot model.

But all this can only happen dynamically: Objects in the real world move, terrain changes, walls may collapse. Even when the robot does not move and its vicinity stays constant for a while, assessments about, e.g., objects in the environment, stability of paths or possible decisions may change. Hence, the model of the environment inside the simulation system has to support creation, modification and deletion of entities not only in the CWM, but also in the rigid body dynamic simulation and in the rendering subsystem.

Eventually, real-time sensor simulation is needed to predict the robot behavior during changes in the environment model. In the discussed example of a collapsing wall, e.g., a laser scanner simulation would predict what the robot perceives and show it to the operator, even during loss of the communication link. For this, existing methods of sensor simulation from the software package VEROSIM must be adapted to the highly dynamic simulation of the environment.

Since all information about the environment will be stored inside the CWM (see T4.1), changes in the environment and in the outputs from algorithms will be reflected here. Visualization algorithms can thus easily track and highlight changes and notify the operator (for example about the collapsing wall). This is closely related to T3.5.

The core components of work package WP4, also with respect to WP8 (Requirement Specification and Evaluation), are:

1. Integration and Interfaces,
2. Simulatable Robot Model, and
3. Simulatable Environment Model.

Thus, D4.3 is not only limited to prediction but moreover the use and connectivity of all modules and the simulator. The interface of the simulation model to real hardware components and external input or output devices is of paramount importance.

2 Overview

The deliverable D4.3 (Constructing simulated world from robot percepts) encompasses:

1. Interfaces,
2. Simulatable Robot Model,
3. Visualization and Rendering,
4. Simulatable Environment Model.

These central compartments can be found in the following sections. As an overview, we want to point towards the desired final functionalities of the overall project and the CENTAURO robotic system, focusing on their consequences for the WP4 simulator.

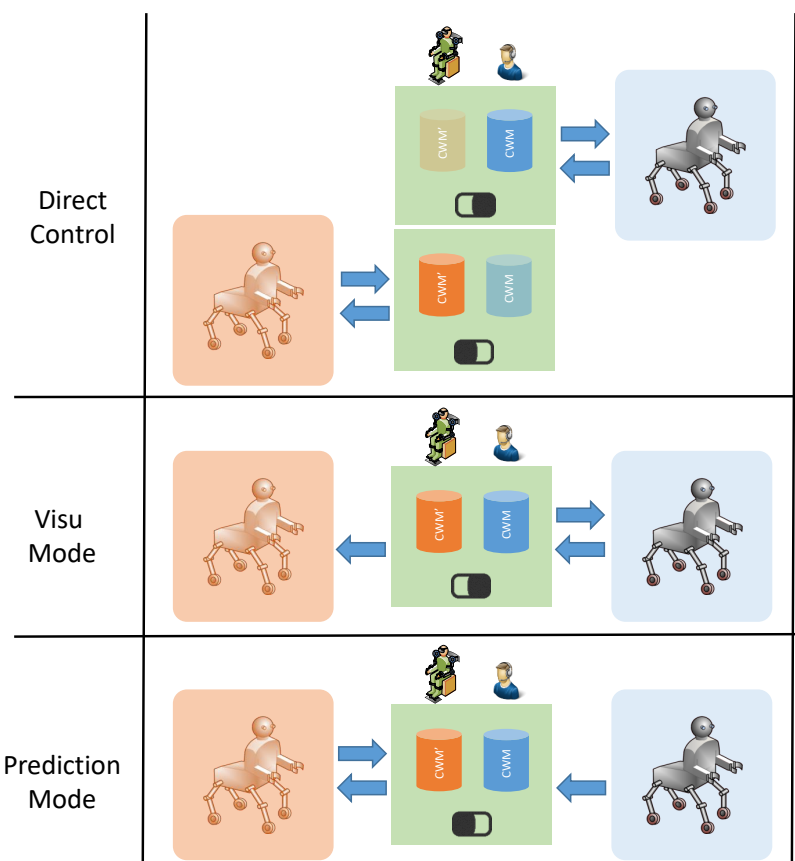


Figure 1: Centauro Modes.

The different operational modes are, as we also see in Fig. 1:

1. direct control of the real system
2. direct control of the digital twin in a static predefined environment
3. direct control of the digital twin in a dynamic perceived environment + visualization of this in the virtual world
4. switch between direct control of real or digital system in a dynamic perceived environment

Thus, the digital twin has to represent all necessary aspects of the real system, especially providing the same interfaces to all project partners. Thus, the digital twin has to be developed in close cooperation to the robot' hardware partner IIT and also all other "interface" partners. This encompasses robot commands and feedback, robot state, sensor data, etc. distributed into the different work packages. In general we discovered the following interfaces with the robotic system (by work packages):

- Navigation (NAV)
- Manipulation (MAN)
- Terrain Classification and Object/Workspace Perception (TC & OWP)
- Exoskeletal control (EXO)

Consequently, we present the main interfaces in Sec. 3 first before we describe the robot model in sec 4. The visualization of live robot' data, the stereoscopic rendering in general, and the setup of natural human operator interfaces are shown in Sec. 5. Finally, the different missions/scenarios — static or dynamically generated during the mission — are then presented in Sec. 6. Everything then comes together in an overall holistic system of operator(s), robot, digital twin, and data processing systems in Sec. 7, before we conclude our work and research in Sec. 8.

3 Interfaces

Based on the work of previous work packages (D4.1 and D4.2) we enlarged the scope towards the changed requirements of the robotic system. Due to the central so-called **XBotCore** framework which is used on the real robot connected to the real-time capable core, we want to mimic this as good as possible. Thus, the integration of XBotCore into the VEROSIM simulator, as well as extensions to the general ROS interface are the main foci of implementation. As presented in Sec. 2 the main interfaces are the Robot, MAN, NAV, EXO, TC&OWP, which partly depend on XBotCore, or on ROS. The general interface standards are presented in the following, whereas specific interface details are described in the modeling section (Sec.4), where they are applied to the different robotic systems.

3.1 ROS Interface

The overall structure of the ROS interface of the simulator is still the same. We used the modular implementation scheme motivated in D4.1 and D4.2, and extended the functionalities towards the needs of all project partners. This led to the following functionalities:

- `std_msgs/*`
Float64, Bool, Int, ...
- `sensor_msgs/*`
JointState, Image, CompressedImage, PointCloud2, ...
- `tf2_msgs/*`
tfMessage
- `rosgraph_msgs/*`
Clock
- `nav_msgs/*`
Path
- `centauro_msgs/*`
HeightMap, ModelPose, DrivingMovement, LegMovement, TexturedPolyhedronMesh
- `xcm/*`
ADVRJointState, ADVRJointCommand

Depending on the use case, publisher and/or subscriber of these message types have been implemented, with respect to the CENTAURO mode (see also Fig. 1). For the 'Visu Mode' the simulator has to subscribe to everything the robot publishes, whereas the 'Direct Control' (of the virtual robot) one has to produce and publish all sensor data in the simulator and publish these to the processing nodes in the system. Thus, each mode has other requirements for the type and number of ROS nodes used.

3.2 XBotCore Interface

The XBotCore architecture has already been presented in previous deliverables, mainly D2.1 and D2.2. In summary, XBotCore is used for the real time capable control of the final CENTAURO robot and should be also used in the according simulator. The XBotCore platform can

be directly incorporated into C++ based code or can be extended via "Plugins" to generate for example ROS message in- and outputs, which is the main middleware used in this project. Thus, in general the XBotCore interface with its plugins can be used to connect the different modules (NAV, MAN, EXO, TC&OWP) to the real robot or the simulator using the exact same interface. For the simulator this means an integration of XBotCore and an adoption of all other interfaces needed (mainly in the ROS context) which are not directly covered by XBotCore.

4 Simulatable Robot Model

To mimic all necessary aspects of the real robotic system, we want to instantiate so-called "digital twins" of the used robot. This means to not only import the robot model, but also representing all necessary internal and external interfaces. Based on the robot these are of course different. Due to the used robotic systems in the CENTAURO project, we instantiated one *Momaro* robot and one *Centauro* robot, which their unique interfaces. In the following we present those robot models and the according interfaces we implemented in the surrounding simulator *VEROSIM*.

4.1 Different Robot Models

For each robot we can define two different robot models:

1. **digital twin model**
2. **visualization model**

The "digital twin" is a copy of the real system. This means we use the same input devices to control the robot which is based on the same URDF model and incorporates the same real-time capable core software. Additionally, this model has the same setup of sensors which publish the same sensory output on the same ROS topics as the real robot. Of course, the sensor data gathered in simulation can deviate from the sensor data of the real robot.

The "visualization model" is a pure subscriber model. The robot's joint state is visualized, the real sensor data is visualized and everything else the real robot publishes is collected in the simulator. Thus, we have a visualization of the environment, the robot's state, and the sensory output (in- and excluding all processing steps). The gathered sensor environment data can then be used later for the dynamic environment generation.

As one can see in Fig. 2 the general setup of our robotic systems (no matter if we use Momaro or CENTAURO system) can be divided in the robot's base (base + upper body + legs + arms), the two (different) hands, and the sensor head.

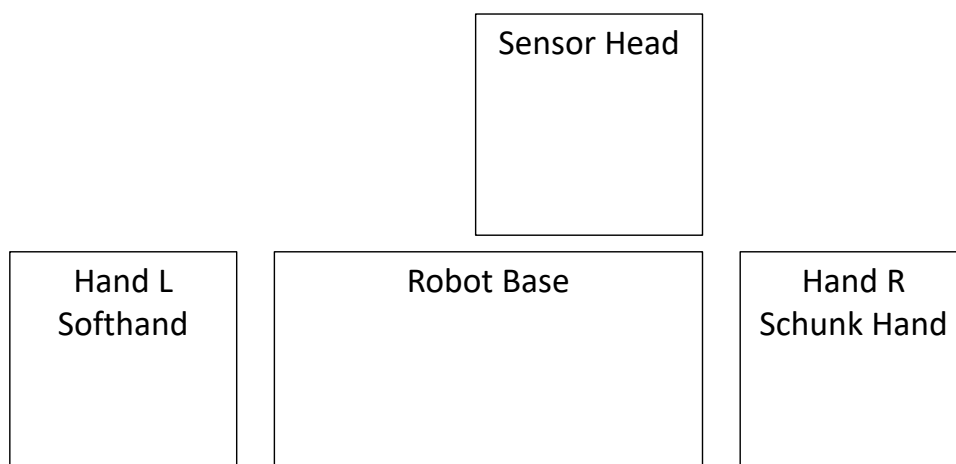


Figure 2: Modular Robot Setup.

The general implementation scheme for an automated import of this (always changing) robotic system is:

$$\text{URDF} \xrightarrow{(a)} \text{model} \xrightarrow{(b)} \text{digital twin}$$

where we (a) automatically import the URDF file and parse it to be compliant with the VEROSIM data structure. And (b) additionally automatically add necessary internal and external interfaces to the model, based on the robot description.

Mainly concerning the 'digital twin' model this is presented for the two robots used in the CENTAURO project in the following sections.

4.1.1 Different Control Devices

For a direct control of the virtual robot we implemented a set of control devices. Of course, the final system has to interface the exoskeleton and/or navigation ROS messages, but for testing and development some simpler control devices are feasible. As already presented in D4.2 we established an UDP-based interface to the exoskeleton which can still be used to control for example the upper body of a robot. Further developments regarding the exoskeleton now shifted towards the incorporation of XBotCore and widening the ROS interface. Another possible control device is the control software from UBO (*UBO Robot Control*) which is also used in several *Momaro* applications. Due to its ROS-based core and a defined interface (shown in D4.2) we were able to use this for a direct control of the *Momaro* system also in VEROSIM. This also led to various possibilities of using ROS and scripts to custom interface with the system. Besides these high level interaction schemes we also implemented a direct gamepad control of the CENTAURO robot which is presented in Sec. 4.2.1.

4.1.2 Different Control Schemes

Our control scheme utilizes the two analog gamepad sticks mapping their position to desired velocity of the robot. We have implemented several mappings. The operator may choose either (a) to control the translation and rotation independently, or (b) move the robot in a car-like mode, where the one gamepad stick is used to apply the steering angle, whilst the other stick the translation velocity. In this second mode, the robot isn't able to rotate instantly, but it enables the operator to rely on his habits driving a car and so do less control errors. We also implemented a third control mode, which we call caterpillar-mode (c). Here we map the two gamepad sticks onto velocities of two virtual caterpillars left and right of the robot. This mode enables the complete motion of the robot like the first mode and is a bit more intuitive (cf. Fig. 3).

We implemented these modes in order to give the most flexibility for the operator to choose the preferred mode. Apart from that, it is possible to evaluate the operator performance using different motion modes in simulation and present the best one for real-life scenarios.

4.1.3 Momaro

The *Momaro* robot is a robot built by UBO. We are able to import the URDF file of the robot and simulate its motions. Usually the robot motion is planned and executed by algorithms implemented by UBO. Despite that, as an option for the operator, we have implemented several control strategies for the robot using a gamepad, as presented before.

As *Momaro* is represented by an URDF model, we use the VEROSIM importer for reading the URDF file and generate a VEROSIM model including the rigid bodies, joints and motors for rigid body simulation. We have performed the necessary adaption for the corrected leg geometries within the original URDF file, such that the import is completed automatically; manual adaption steps are not necessary.

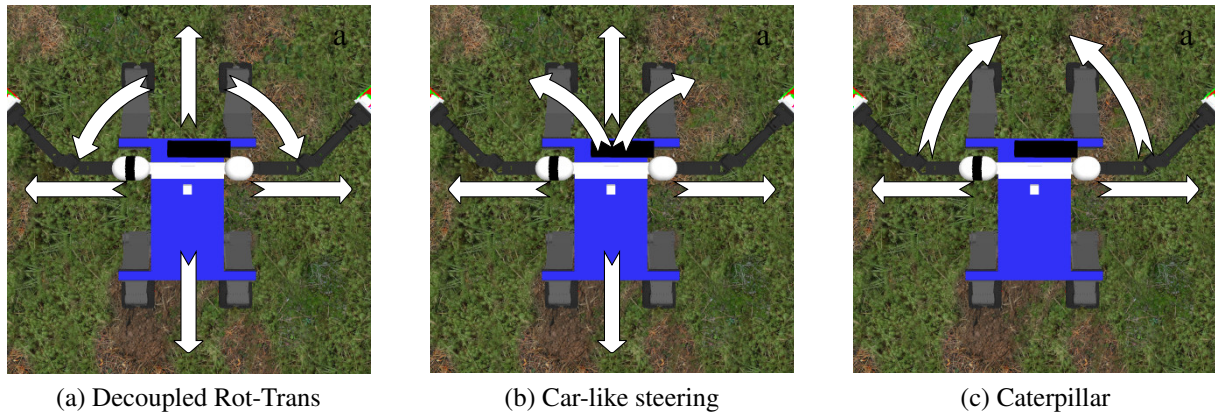


Figure 3: Different control strategies (top view)

The Momaro robot is equipped with four wheels (one each leg) which are able to rotate in the vertical axis. In order to continue the automated import, we have written an adaption script that integrates the different control modes into the model.

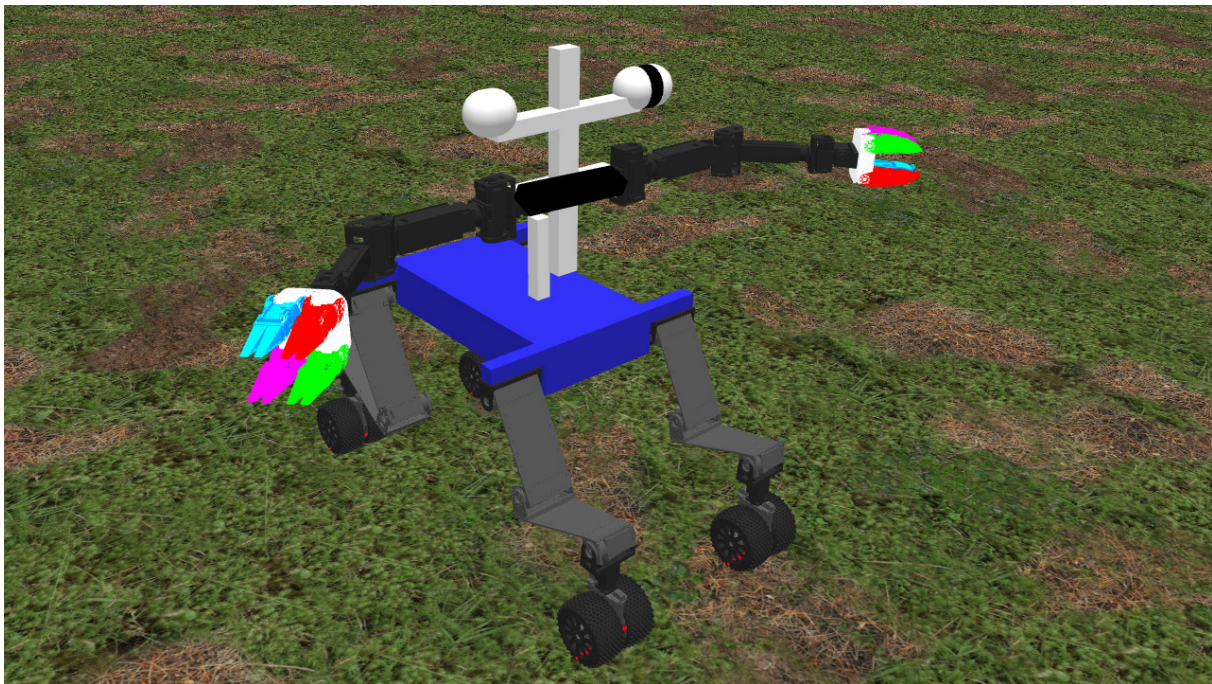


Figure 4: Robot Momaro with complete hulls.

Like the Momaro robot, the CENTAURO robot will also have four wheels enabling equal motion capabilities; therefore we can reuse the import and steering functionality with the CENTAURO robot as well.

4.1.4 Schunk Hand

As with the most components, there exists a URDF model of the Schunk Hand. This hand will be mounted on the right arm of the Centauro robot. This human-like hand has five fingers and 20 degrees of freedom. The nine motors included into the hand are able to move each finger independently. However, several joint groups (usually a finger is a group) are connected together to one motor enabling synchronized motion of these joints.

The mechanical connection of these joints is represented in the URDF file by a mimic-entry, where the scale and offset of the joint motion is noted with respect to the reference joint. We adopted our URDF importer for handling this kind of description. The resulting imported VEROSIM model generates for each of the joints of the group a motor. These motors share a common input and force synchronizes motion as required by the URDF model.

In addition we generate a ROS subscriber compatible with the descriptions coming with the ROS package *schunk_svh_driver*. Using *svh_sim_test*, provided by this package, we have validated the functionality of the imported VEROSIM model, which is now ready to be used as a digital twin.

4.1.5 Schunk Hand and Exoskeleton

The communication with the exoskeleton is not final yet, but for now, the communication based on UDP packets is available. In parallel to the ROS based interface, in VEROSIM we implement an UDP handler in order to provide a simulation of the Schunk Hand for usage with the exoskeleton at this stage.

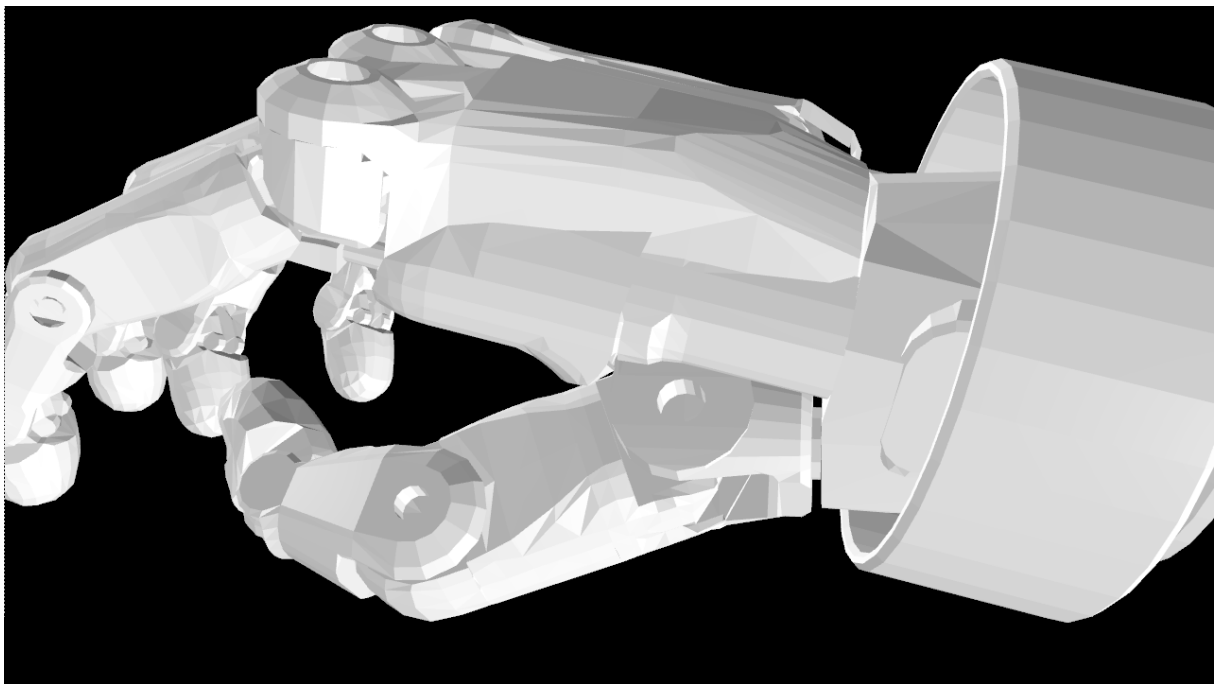


Figure 5: Schunk Hand imported from URDF description into VEROSIM.

4.1.6 Centauro

The transfer from Momaro to Centauro is based on the consistencies (+) and differences (-) of these two systems.

- (+) "same" sensor head setup
- (+) externally accessible via ROS (using similar ROS messages)
- (-) internally controlled by *UBO Robot Control* vs. *IIT XBotCore*
- (-) different hands

Of course the import procedure is the same as before, we still use an URDF file to import the model into VEROSIM, as we can see in Fig. 6

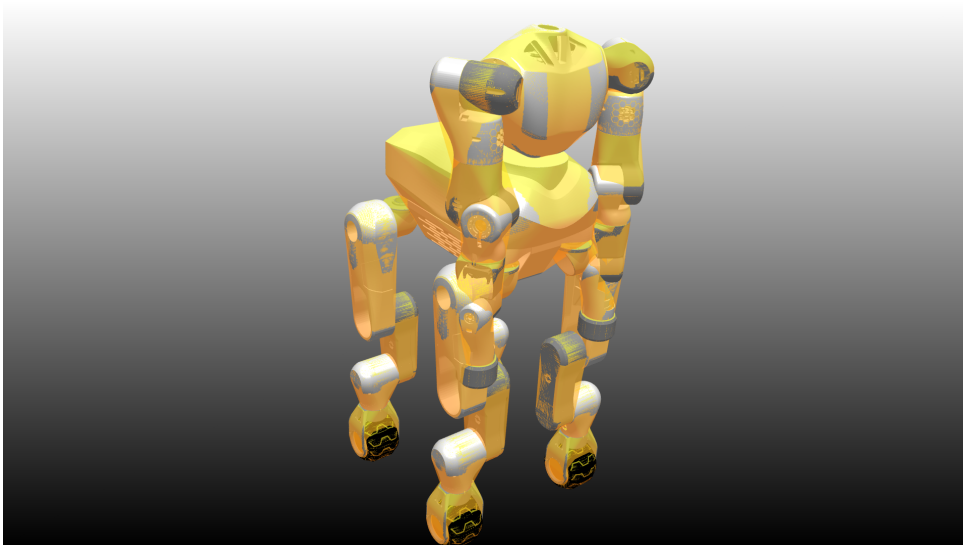


Figure 6: Centauro model.

The main systematical transition is done by using the XBotCore framework, which will be discussed in detail in the next section.

4.2 XBotCore

XBotCore is a modular library developed by IIT with the focus on the communication via EtherCat and controlling robots based on this communication bus. IIT has already demonstrated the principle using Gazebo as simulation system. Doing this, many of the control algorithms may be implemented and tested before the real robot is ready to use.

Using XBotCore as one integration tool we can abstract a lot of issues with this interface. The overall holistic setup of integration is shown in Sec. 7 but for now we can already say that most control and feedback data transmission will be piped through XBotCore, whereas sensor data will be directly transmitted via ROS. Furthermore, XBotCore can then provide additional ROS nodes for all non-real-time critical control and feedback tasks of the robot.

4.2.1 ROS

Besides the ROS interface, to directly interact and control arbitrary robots in the simulator, presented in Sec. 3.1, we additionally implemented further ROS message types associated with XBotCore, namely *ADVRJointState* and *ADVRJointCommand*. These can then directly be addressed by external tools without using the XBotCore interface. For the ease of use we also implemented a direct interface for simple navigation tasks. Based on the NAV ROS msg *centauro_msgs/DrivingMovement*, we implemented a Publisher of this message type which can directly be connected to any input device used in the simulator. Thus, we can easily test and access driving movements with a gamepad, joystick, or any other input device of choice.

For a direct visualization of the robot's *TF2/tf* tree we can use the according ROS message and automatically generate a tree of frames using blocks as we can see in Fig. 7. The published frame tree of the robot is gathered with a *tf* subscriber and parsed into multiple hashes of the frames. Afterwards, the frame tree is recursively parsed into the VEROSIM frame tree and for visualization purposes each frame is represented by a box with non-uniform x/y/z lengths. This was used as a proof of concept of using a direct control via XBotCore in a simulator, visualized in a second simulator instance.

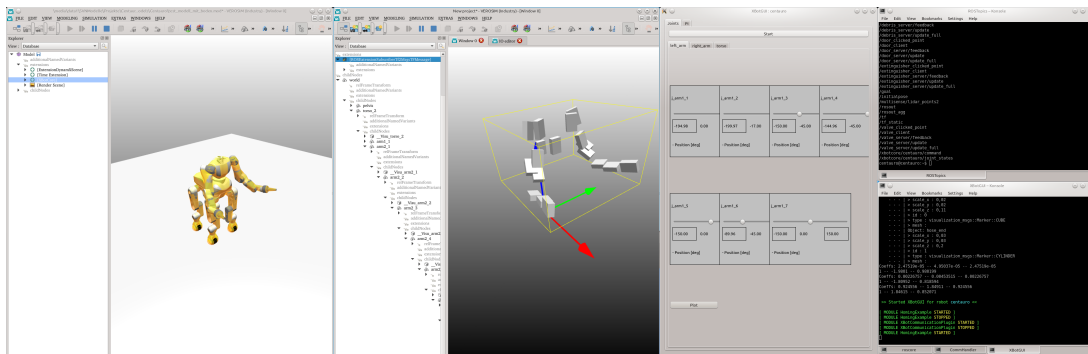


Figure 7: Automatically generated 'tf' frame tree.

4.2.2 VEROSIM

The XBotCore library is modular structured and allows the user to add control extensions or other functionality by the software pattern 'Dependency Injection'. In particular it allows the user to drop in implementations for the interfaces of *XBot::IXBotJoint*, *XBot::IXBotIMU*, *XBot::IXBotFT*. Doing this allows the user to replace the robot under control by a simulated one.

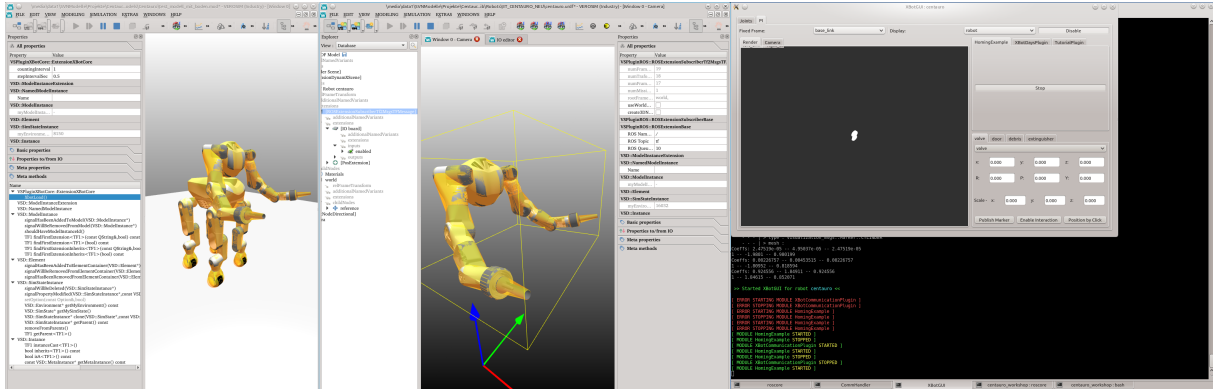
After the CENTAURO Workshop, where we have got deep insight into the XBotCore structure, we were able to implement the needed interfaces using VEROSIM as simulation system. Here we also provide the developers with a real-time simulation of the CENTAURO robot. In addition, we establish a link between XBotCore and the other components used with Centauro.

In particular this means a one-to-one mapping of

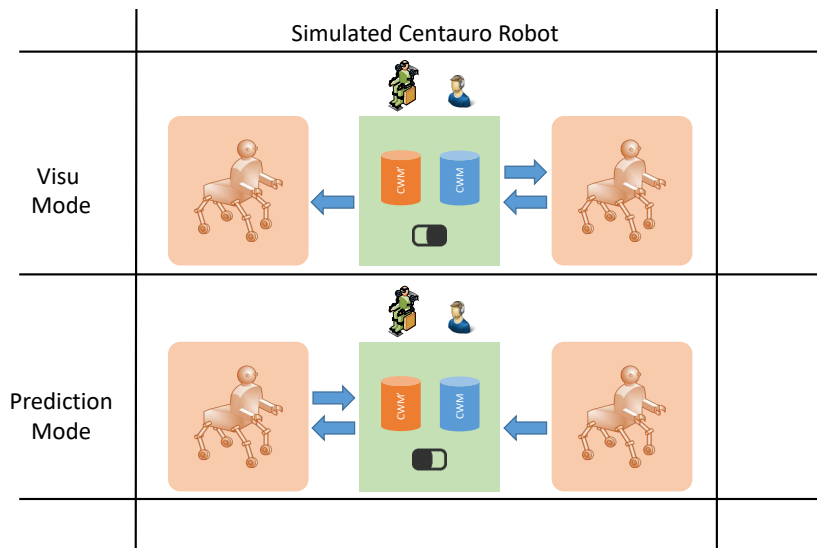
- Joints,
- Links,
- Force Torque (FT) Sensors, and
- Inertial Measurement Units (IMUs),

whereas especially the FT sensors and the IMUs are still under development.

For testing and evaluation of current developments we use the simulated CENTAURO robot in the prediction mode. Represented by Fig. 8b we use for example the XBotCore plugins (right) to directly control the robot (or its digital twin) on the left, while visualizing the current robot’s state in another instance of VEROSIM (middle).



(a) The system in action: VEROSIM (with XBotCore) + VEROSIM Visualization + CommunicationHandler + XBotGUI



(b) Concept of using the simulator twice, as a digital twin and its use in visu or prediction mode

Figure 8: Using the simulator: Centauro direct control + visualization of JointStates.

5 Visualization and Rendering

In general, we already presented the rendering pipeline for example in D4.2. Also here in D4.3 we use ROS as the commonly accepted middleware to transmit generated and processed data. As described in Sec. 3.1 we primarily use standard ROS sensor message types, accompanied by custom message types for processed vision data.

For the visualization and rendering three main aspects are important:

- visualization of "raw" data
- visualize preprocessed sensor data
 - optimized for an immersive (stereoscopic) presence of the 1st person operator,
 - enhance the operability by useful visualization metaphors, like Head-up-display or Projective VR,
 - combined to textured meshes

As we have already shown the visualization of raw data (in terms of images, point clouds, or the robot's state) in the EU review and follow up integration meetings (cf. Fig. 9 and Fig. 10 accordingly). Now, we want to focus here on continuous expansion of this concept.

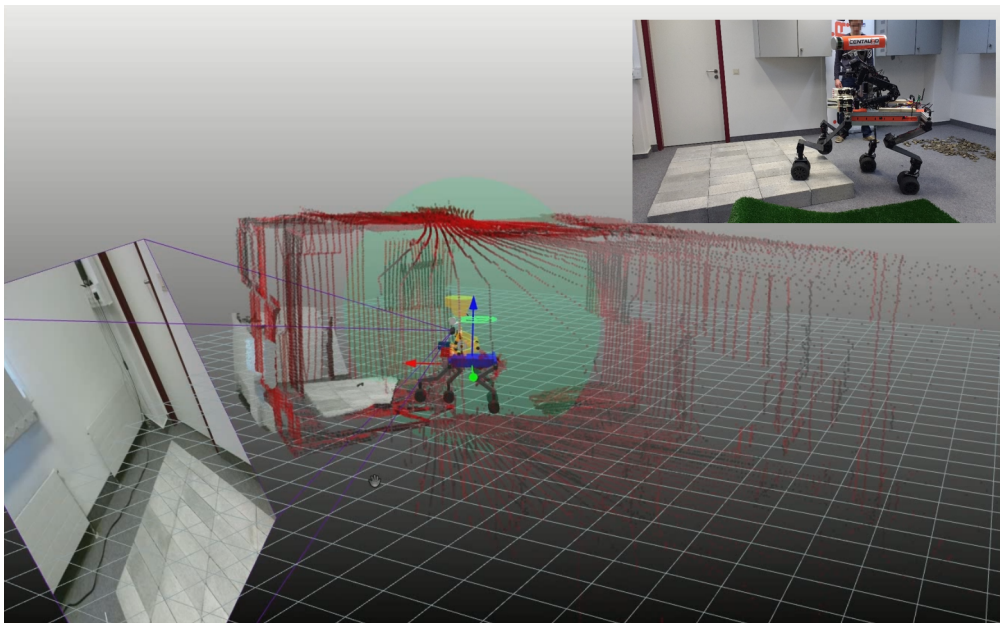
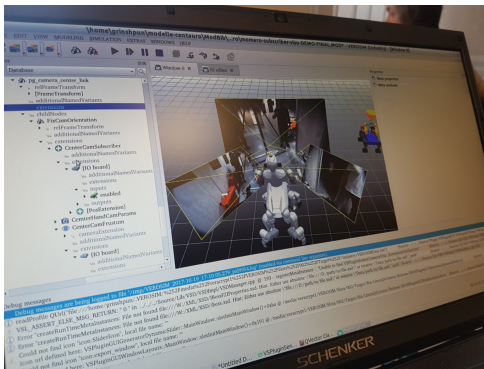
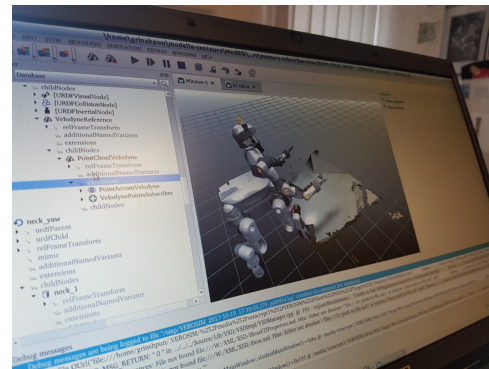


Figure 9: Review Meeting Momaro live visu of PointCloud, Images, JointStates, Tracer.

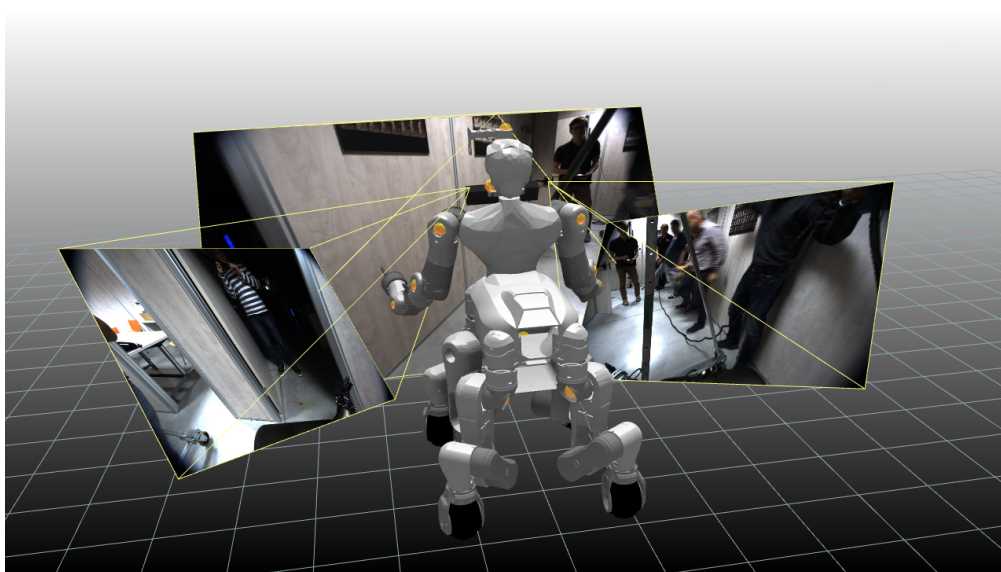
Besides the sensor message raw data, we extended the capabilities of VEROSIM for visualizing more data types in a natural and easy to use way. Transmitted via ROS messages we are now able to visualize different processed sensor data in the simulation system.



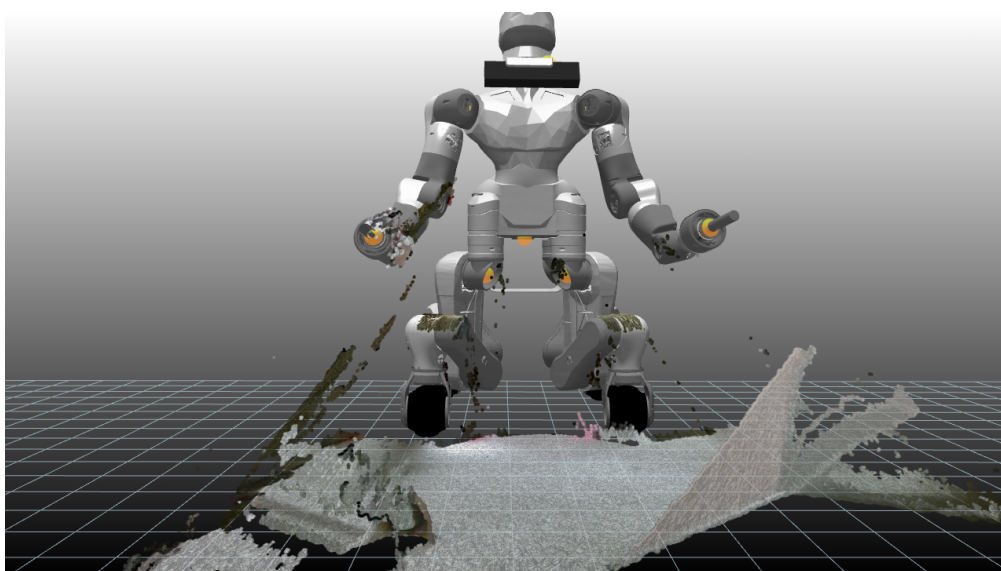
(a) Centauro live visu of Images



(b) Centauro live visu of PointCloud



(c) Centauro live visu of Images



(d) Centauro live visu of PointCloud

Figure 10: Visualization of real data of the CENTAURO robot.

5.1 Operator Visualizations

Intuitive visualization of data is often the key element to an enhanced user experience. Thus, we extended frameworks to generate *overlays* and *billboards*.

Overlays: As one can see in Fig. 13a one can generate a user- or application-oriented Head-up Display (HUD) to visualize internal or external data, like battery, parameters, images, maps, plots, compass, etc. This user interface can then be used in the final mission in an individual setup based on the operator and the chosen I/O devices.

Billboards: Additionally, environment or object information can be projected into the 3D scene to visualize additional information. We implemented (a) traversability cost map visualization, and (b) path visualization. The cost map can be seen in Fig. 11 and represents a three-dimensional overlay of a dynamically generated heightmap, which will be presented later in Sec. 6.2, with a color map of traversability costs. This overlay should enable the operator to directly see drivable and critical areas more intuitively.

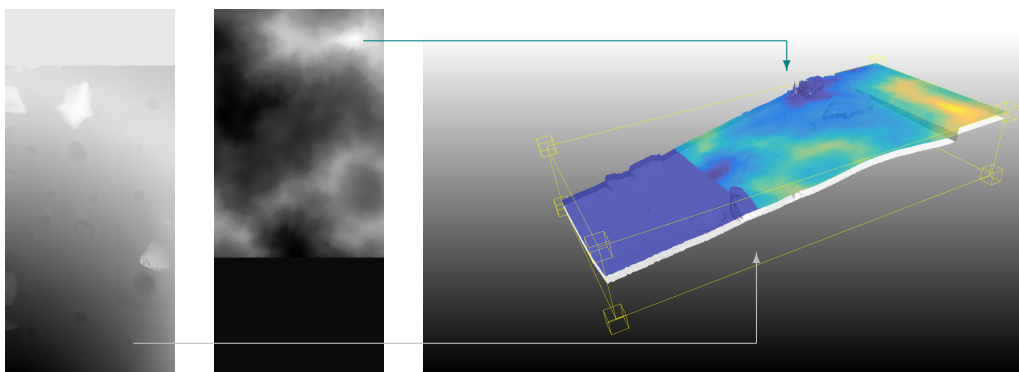


Figure 11: Rigid body 3D environment, generated via a ROS height map with a color overlay of a traversability cost map.

Furthermore, we also implemented the (online) visualization of planned paths. This path visualization can be seen in Fig. 12c. Transmitted via ROS msgs (*centauro_msgs/HeightMap* and *nav_msgs/Path*) the data is preprocessed by project partners, and finally rendered in the simulator.

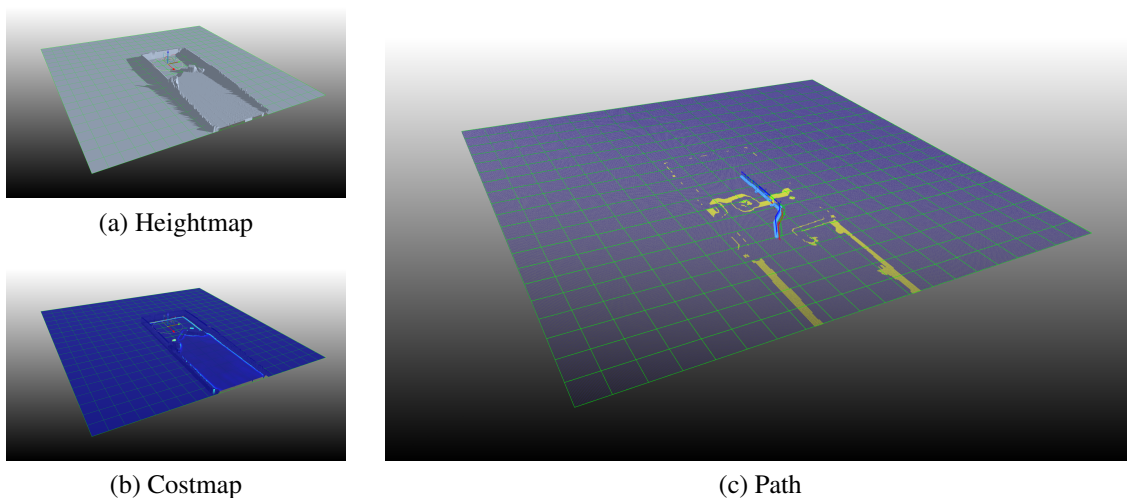
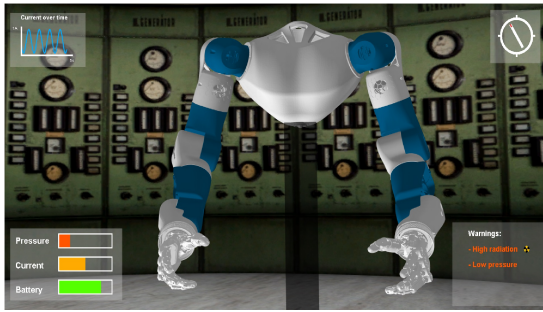


Figure 12: Visualization metaphors.

Besides the aforementioned visualization one can also use billboards for detected objects,

rendered at the site of occurrence (see Fig- 13b).



(a) Head-up Display (HUD)



(b) Projective VR "Billboards"

Figure 13: Simulation-based operator interface and visualizations.

These billboards represent a pose $P(p, O)$ and an information string s . They are automatically oriented to the camera viewpoint, and also fade-in and out by a predefined distance. The used ROS msg type is here the customized *centauro_msgs/ModelPose*. Due to the ROS interface we can already connect any ROS transmitted data (and of course data from internal simulator frameworks) to generate such visualization metaphors.

5.2 Textured Mesh Rendering

Based on developments of UBO, who achieved a textured mesh renderer as a standalone tool for the review meeting, we have decided on the following integration concept:

1. RWTH integrates the code of UBO used at the review meeting for rendering Kinect v2 data directly into VEROSIM,
2. LIU developers a textured mesh polyhedron renderer node,
3. RWTH interfaces the LIU node, utilizing previous integration of UBO's code.

All in all, we want to achieve a ROS network, where the sensor head publishes ROS messages (Velodyne PointCloud + Kinect PointCloud + Kinect RGB image + 3 RGB Camera images), one ROS node (by LIU) processes this data prior to rendering in the simulator.

Additionally, RWTH has to provide the possibilities to render this textured mesh on normal monitors as well as stereoscopic VR glasses (like the HTC Vive).

5.2.1 Kinect Renderer

The integration of UBO's textured mesh renderer into VEROSIM is currently under heavy development. The general conceptual idea is to directly incorporate the given code into the simulator directly. This code involves only data from the Microsoft Kinect v2, generating a pointcloud based mesh with projected image textures onto it. Based on these developments we will extend the concept to the unit sphere rendering in the following. As one can see in Fig. 14 we already achieved a preliminary integration of UBO's code for rendering Kinect v2 data.

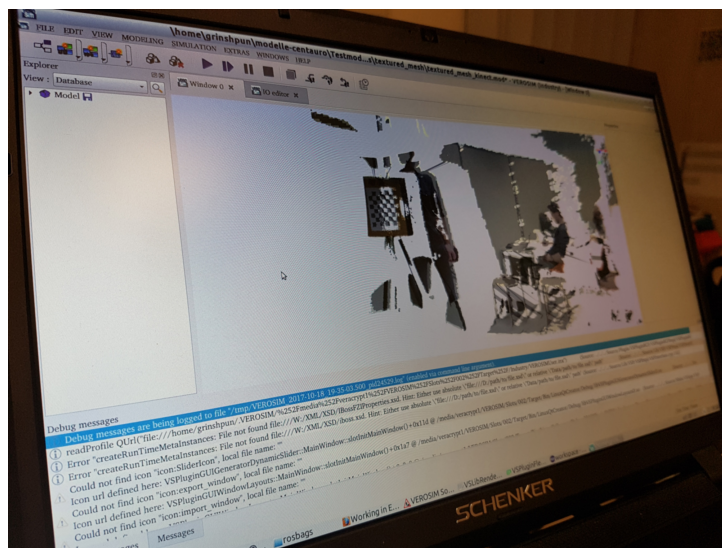


Figure 14: Visualization of a textured mesh.

5.2.2 Generalized Renderer

The generalized renderer represents a unit sphere rendering approach of LIU. This will be a standalone ROS node that handles (and fuses) the pre-processing of sensor data from the Velodyne, Kinect, and the three RGB cameras. The consortium decided on the following customized ROS msgs types:

- *centauro_msgs/Polygons*
- *centauro_msgs/SubMeshTexCoords*
- *centauro_msgs/SubMeshVerts*
- *centauro_msgs/TexturedPolygonMesh*
- *centauro_msgs/TextureMesherInput*

where the **input** of the system is represented by the *TextureMesherInput* data type and consists of the pointcloud, image, and camera info data. The **output** of the system (and thus the input to the simulator) is represented by the *TexturedPolygonMesh* which especially comprises the pointcloud, vertices, texture coordinates (u,v) and images. These can then directly be rendered in the simulator (or any other renderer used).

5.2.3 Stereoscopic Rendering in General

The integration of stereoscopic headsets, like the HTC Vive, requires a holistic integration in the simulator. Thus, *OpenVR* and *SteamVR* are used to interface with the given SDKs and could directly be used in VEROSIM. For Windows this integration is completely finished and showed very good and performant results. The Linux SDKs are unfortunately not supported completely yet but we are currently working on implementation details. First results can be seen in Fig. 15 where we use the HTC Vive under Linux to render the current robot model state.

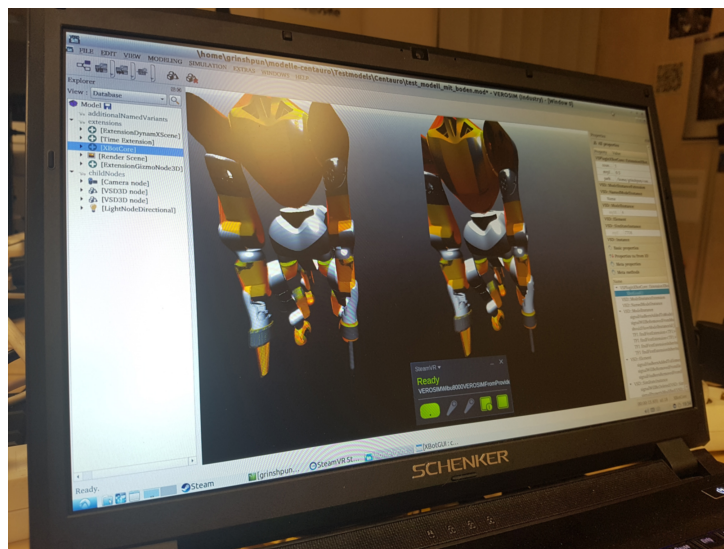


Figure 15: Stereoscopic rendering using HTC Vive.

If there will be still challenges during the integration phase we already set up a backup concept of using a Linux and Windows VEROSIM in parallel, where we utilize Core and ROS functionalities under Linux, and the rendering under Windows. The current set of all properties can then be synced in between these systems.

6 Simulatable Environment Model

The environment model is one main aspect of the CENTAURO system. On the one hand, we want to use the simulation prior to the final evaluation to conduct the defined benchmark tasks (of M30 and M42 evaluation). On the other hand, we want to use a dynamic perceived environment (build by sensor data of the real robot) which can be switched into, to test and evaluate planned actions. Thus, we will present those "static" and "dynamic" environments we have instantiated so far in the following.

6.1 Static Environment

In general, static environments are not meant to be static, but to be in a predefined state. Only on the broader scope, these environments are not dynamically created during runtime, but are predefined. Thus, we have an environment that does not change over time, but nevertheless some parts of it can be moved, modified, etc. We implemented a set of environments that are associated with WP8 Integration. We used the M30 and M42 benchmark (real) environments and tasks to recreate these in the simulator. With this we can use the simulator for an earlier testing and benchmarking in the real environment.

To have a holistic database for all these parts we developed a *Model Library*. There we can encapsulate all robots, environments, items, and tasks. Additionally, we can generate evaluation or benchmark scenarios by point and click. In the following, we first present the Model Library itself (cf. Sec. 6.1.1) before presenting M30 (Sec. 6.1.2) and M42 (Sec. 6.1.3) evaluation.

6.1.1 Model Library

The Model Library is a project specific conglomeration of models. In CENTAURO we defined

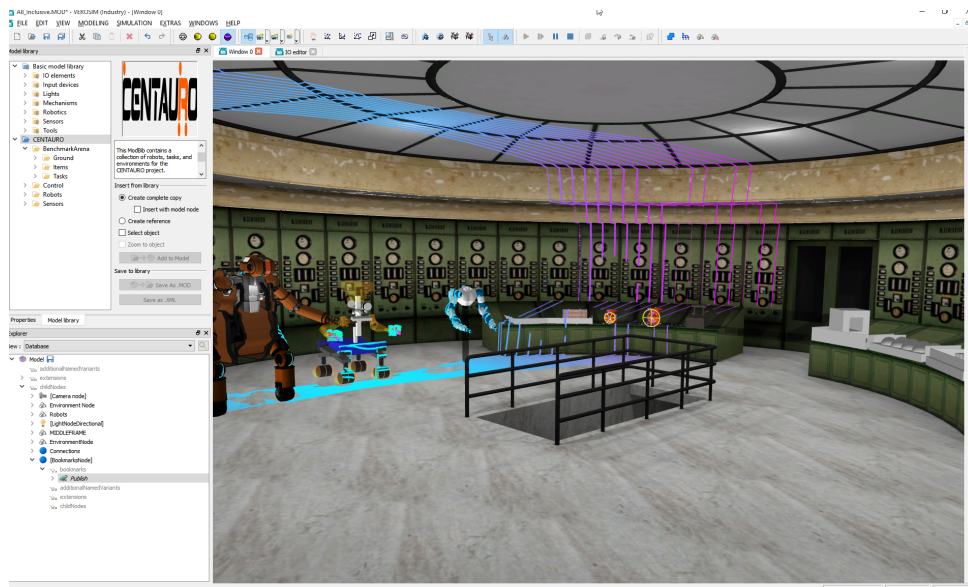


Figure 16: ModBib Concept: Environments, Robots, Sensors, Items/Tasks.

the four model types:

- robots,
- environments,

- items and tasks, and
- sensors.




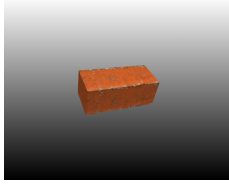

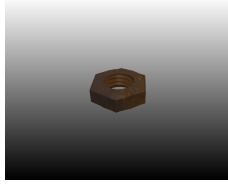


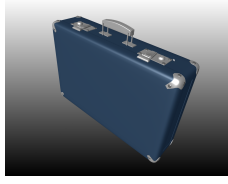
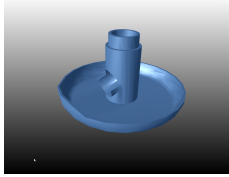
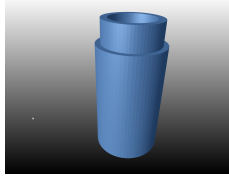
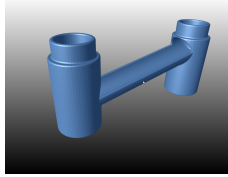
Of course the different robot models presented in Sec. 4.1 are gathered here, as well as the different sensor types of the final sensor head, like:

- Velodyne (point cloud),
- Kinect v2 (point cloud + RGB image),
- wide angle RGB camera.

Additionally, we group the setup in (a) environments and (b) items/tasks. Thus, we can generate a holistic setup by first choosing an overall environment and afterwards filling it with items. Additionally, we can save this composition using relative model paths in one integrated model file.

Besides the aforementioned robots and environments we also have some "normal" objects developed for further tasks. Tab. 1 presents some of these models. Specific models w.r.t the integration protocol can be found in the appendix in Tab. 2, Tab. 3, Tab. 4, and Tab. 5.

Table 1: ModBib: Collection of sample objects for navigation tasks (a,b,c) or standard manipulation tasks (d-i) up to very filigree manipulation tasks with a marble run (j,k,l).

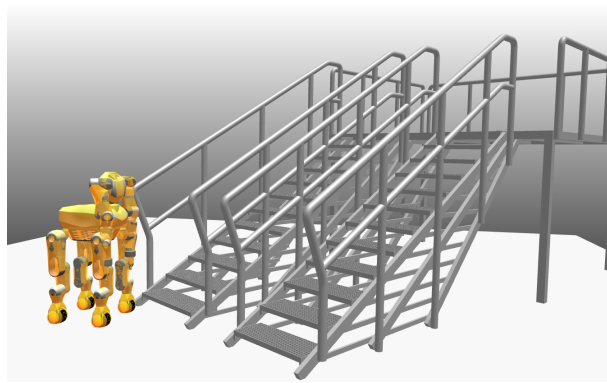
Nr.	VEROSIM Render Image		
a,b,c			
d,e,f			
g,h,i			
j,k,l			

6.1.2 KHG M30 Evaluation

The evaluation of a first integrated CENTAURO system will take place in M30 at the KHG test facility. Based on images and CAD drawings we re-modeled all necessary evaluation tasks defined in D8.1, which are also presented in the appendix in Tab. 2. One rough example is shown in Fig. 17 where we remodeled the steps and platform present at KHG.



(a) Image



(b) VEROSIM

Figure 17: KHG environment example: platform and steps.

Besides the ability to use all of these developments during the M30 evaluation meeting, we can also use it prior to this for testing.

6.1.3 USAR M42 Evaluation

The final CENTAURO system evaluation is expected to be at the USAR test facility in Ahrweiler. As presented in previous reports this facility comprises a so-called "disaster road" including damaged buildings. We have gathered point cloud data of the facility with a *Zoller+Fröhlich ZF IMAGER 5006* high resolution laser scanner, with a resolution (set to high) of $H = 0,036^\circ$ and $V = 0.036^\circ$, $f = 10.000\text{pxl}/360^\circ$, an ideal object distance of $l > 5\text{m}$ and a beam divergence of $d = 0,22\text{mrad}$.

Some 360° intensity images of this scan accompanied by some real images can be seen in Fig. 18.

The according point clouds currently still needs pre-processing and filtering before we can reconstruct a holistic environment model of the disaster road. A first impression can be seen in Fig. 19 where a VEROSIM pointcloud image can be seen. The conversion of point cloud data into a rigid body based simulation environment is currently under development.

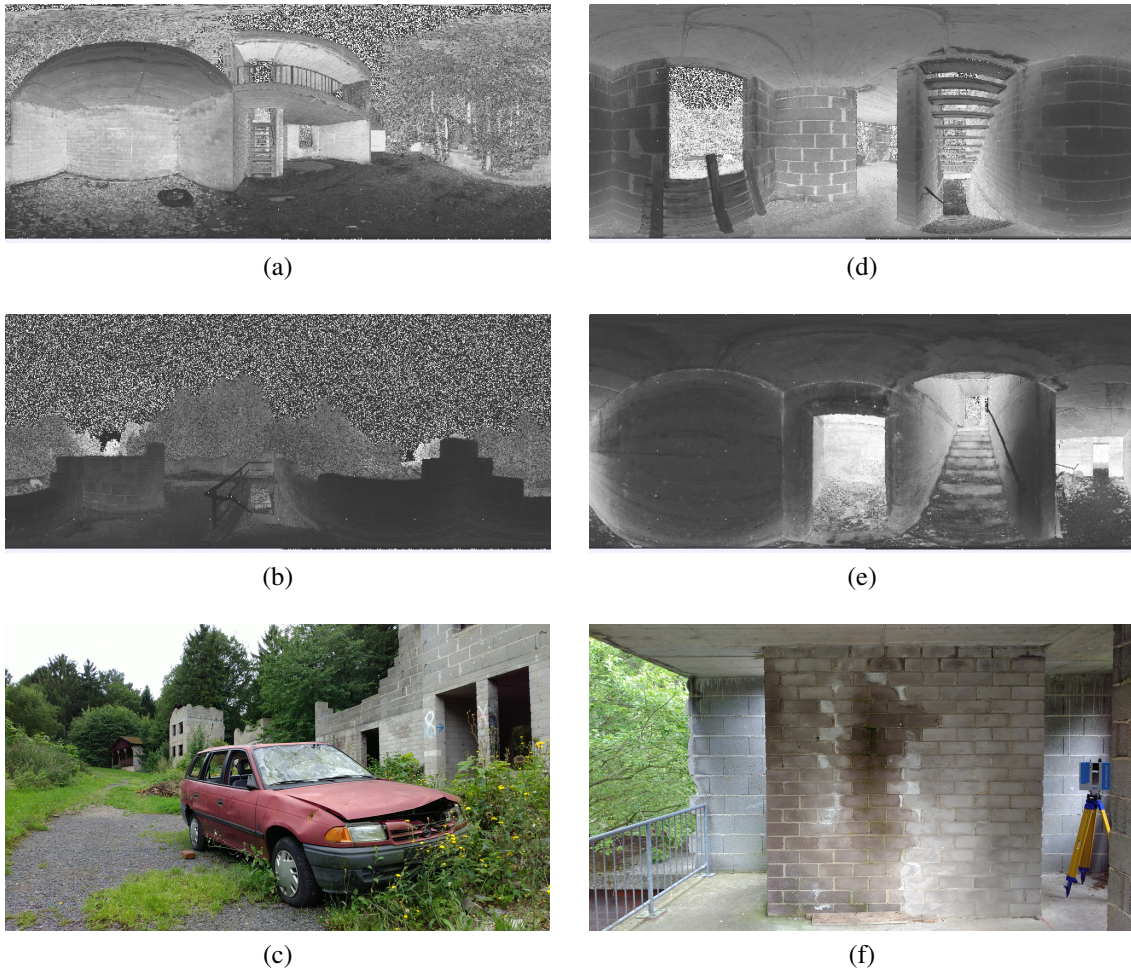


Figure 18: M42 evaluation camp: USAR Ahrweiler "disaster road" 360° intensity images, outdoor (a-c) and indoor (d-f).

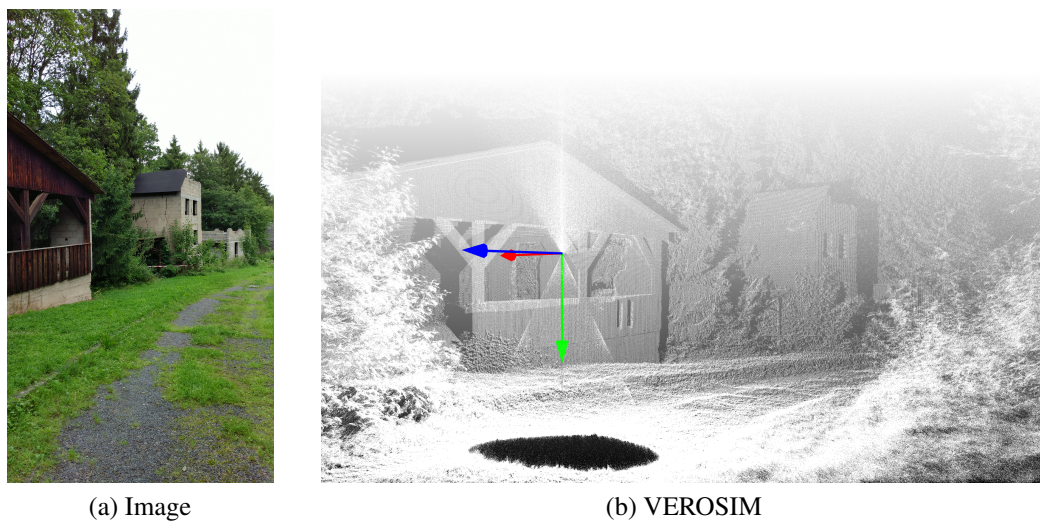


Figure 19: USAR disaster road: sample point cloud in VEROSIM.

6.2 Dynamic Environment

The dynamic environment comprises the two different aspects of

1. ground, and
2. items.

The generation of this environment is of course limited to the sensors used.

6.2.1 Heightmap

We use preprocessed sensor data by UBO which provide the simulator a so-called *HeightMap*. This map is generated by a ROS node of UBO and transmitted via a custom ROS message type namely *centauro_msgs/HeightMap*. Afterwards, this heightmap is imported as a VEROSIM geometry and dynamized into a HeightField rigid body.

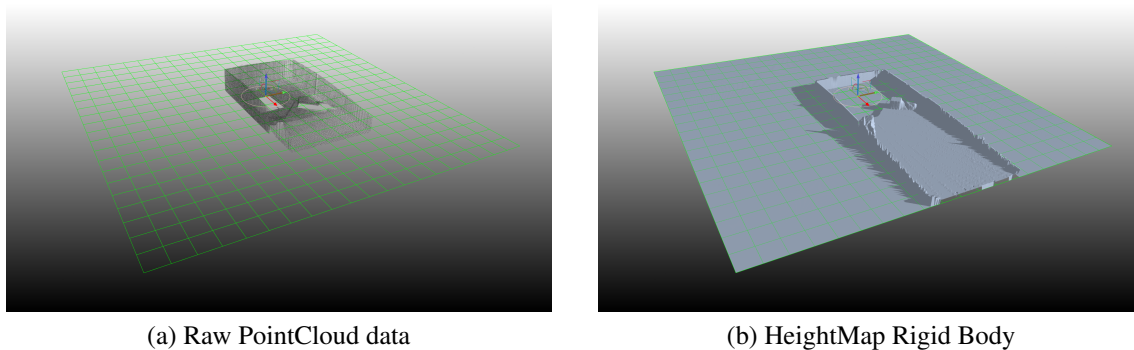


Figure 20: Heightmap.

6.2.2 Template-based Model Insertion

The model library items should also be used for "template based model insertion" based on the object perception tasks. Besides the pure visualization of objects in the scene based on Sec. 5.1 we want to insert an according model at the place of recognition. For example, a hammer recognized on a workbench will be replaced in simulation by a dynamic rigid body model of an according hammer from the model library. Thus, we can grasp the hammer in simulation instead of just visualizing its point cloud. The transmission of information is still done via the ROS msg *centauro_msgs/ModelPose*.

6.2.3 Real-Time Linux

Some complex dynamic simulation models need to be executed in a real-time simulation environment to guarantee a hard upper limit to scheduling cycle times. Especially applications, which comprise physical hardware components besides the simulated environment, impose such requirements.

Figure 21 depicts a setup with different computing nodes working in close coordination to implement a simulation system coping with the inclusion of a set of slave nodes taking the roles of real-time controllers to an actuator (KUKA LWR iiwa), a sensor (Velodyne 3D laser scanner) and a simulated model, that might also be replaced by a physical counterpart.

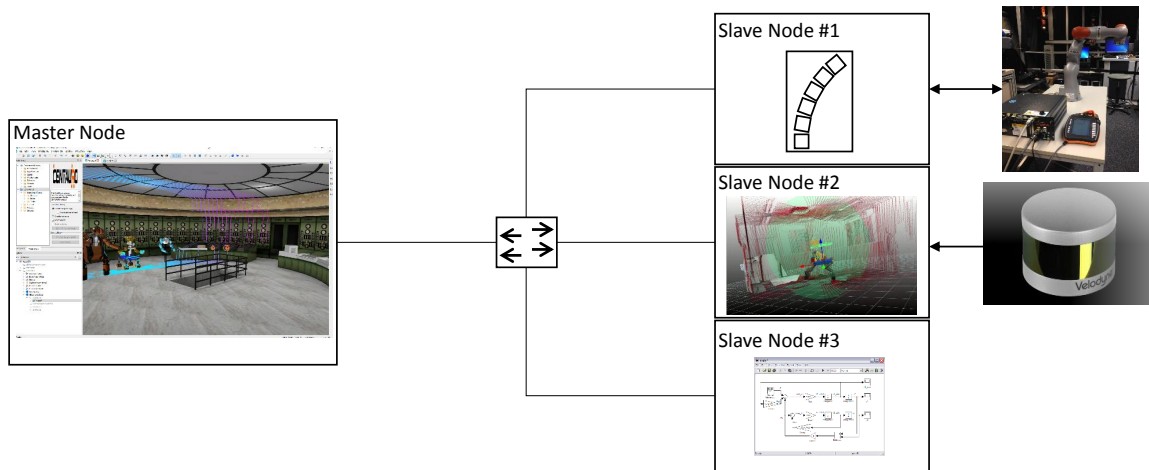


Figure 21: Sample configuration of Master and Slave nodes implementing a complex simulation environment with physical actuators and sensors.

Since the master node is not necessarily a real-time system, especially due to the fact that it is meant to act as an user interface to the simulated environment and therefore contradicts timing requirements, such a separation is essential to guarantee stable operation of the hardware components. As a consequence the master node fuses the simulation results from each slave node into the central simulation model. In the example above, the simulation environment on the master node includes a simulated robot that mirrors the behaviour of the physically attached robot. The dedicated slave node executes a simplified simulation model to calculate trajectories requested by the master including either kinematic or dynamic behaviour. Likewise, a simulated laser scanner is included that represents the acquired 3D pointcloud from the physically attached laser scanner. The sensor simulation on its dedicated slave preprocesses the measurements and generates the pointcloud data to be accessed by the master.

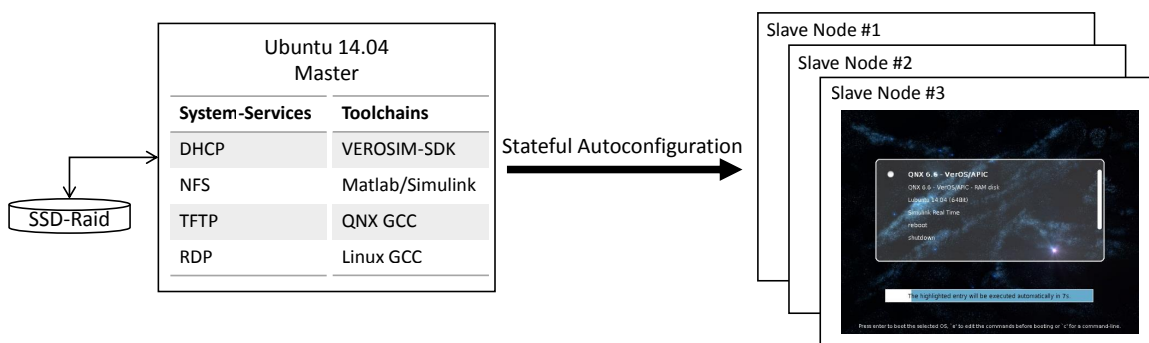


Figure 22: Structure of the automated setup for stateful autoconfiguration of slaves.

The automated setup and stateful configuration was implemented as shown in Figure 22. The master node provides a set of services and tool chains on a fast storage system that is also accessible by each of the slave nodes via network file system (NFS) and trivial file transfer protocol (TFTP). That way the Master Node distributes the operating systems and all the data for the simulation models to the slaves. During the boot process of each slave, it acquires a dynamic IP address from the master and fetches an operating system image using TFTP, e.g. QNX or a Linux Kernel with real-time capabilities. As soon as the Kernel is running the user land is mounted via NFS, at which point the simulation software and models are also available to be executed. The master needs to determine, depending on the simulation to be run, what

role each slave node has to take up and to afterwards start the slave processes.

7 Overall Setup

The overall setup (cf. Fig. 1) comprises the four possible scenarios:

1. **Direct control** of the **real** robot.
2. **Direct control** of the **digital twin**.
3. **Visu Mode**: Direct Control of the real robot visualizing the robot state in simulation in parallel.
4. **Prediction Mode**: Directly control and pause the real robot, switching into virtual reality to evaluate planned action there first, before executing them in reality.

Especially the prediction mode is part of this deliverable and the final evaluation scenario is based on it. Thus, we will focus on this mode in the following.

7.1 Interfaces and Integration

Based on the interfaces and integration partners defined in Sec. 3, the integration concept is shown in Fig. 23.

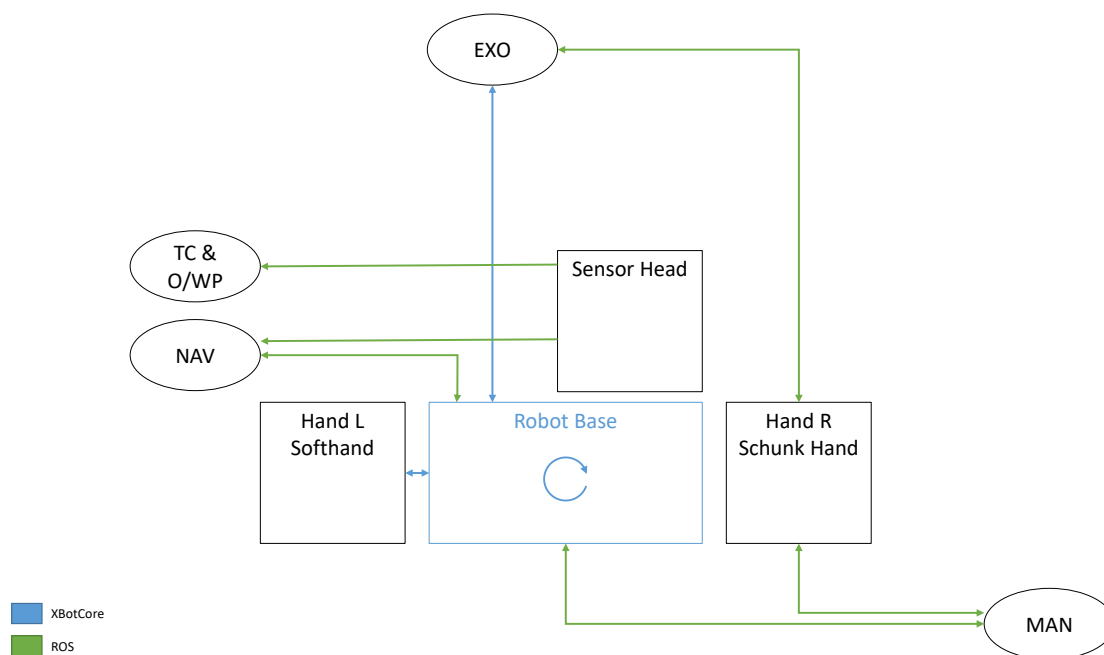


Figure 23: Interface Concept based on Fig. 2, using XBotCore (blue) and ROS (green).

As we can see, most interfaces are based on ROS, where most of the used message types can be found in Sec. 3.1. Everything that is directly integrated into XBotCore should now be ready to work also in the simulator system. The digital twin has to replicate this scheme to be used exactly like the real robotic system.

7.2 Switch into Prediction Mode

The "Switch" of directly control the real robot to directly control its digital twin requires different steps (see also Fig. 1). These steps were conducted in cooperation with UBO as a first conceptual draft how this could be accomplished:

1. Pause the real system (in a safe state)
 - (a) Stop the real system listening to commands
 - (b) Stop the real system publishing joint states/ sensor data
2. Switch the simulation from visualization mode to a dynamic rigid body simulation
3. Stop the digital twin listening to joint states
4. Construct a virtual world (rigid body simulation)
5. Use the rigid body model of the digital twin
6. Start the digital twin
 - (a) Start the digital twin listening to commands
 - (b) Start the digital twin publishing joint states/ sensor data

These steps result in the *Prediction Mode* where we use the digital twin identically to the real system, using the same input devices, the same commands, resulting in the same kind of feedback from the system.

To physically execute this "Switch" we need to take all interfaces into account. Simply said, we just have to redirect all messages from and to the robot to the simulator. In Fig. 24 this scenario is described in more detail. Besides the general ROS commands which can be redirected cutting the connection of "NimbRo Interface Software" (by UBO), additional switches are required for the exoskeleton (by SSSA) and the simulation itself. For the exoskeleton SSSA would have to redirect the UDP packets to the operator station instead of the mobile robot. For the simulator we exchange the robot model and use the modular I/O board where we can easily toggle an enabled/disabled flag for all relevant components.

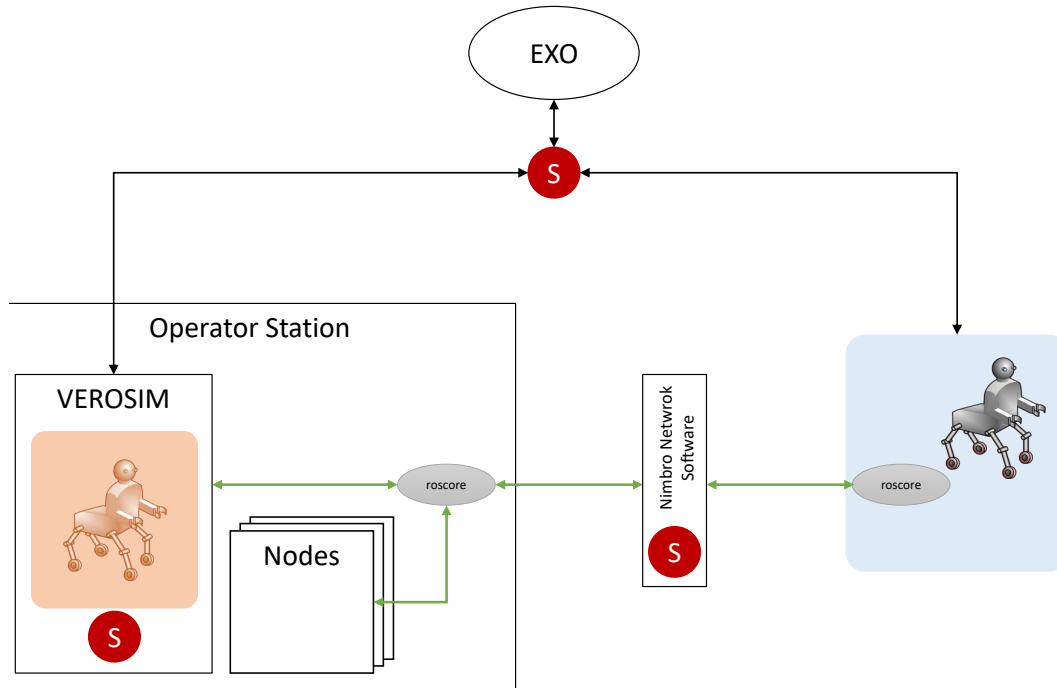


Figure 24: "Switch" (S) in between *Visu Mode* and *Prediction Mode* w.r.t. Fig. 1.

8 Conclusions

Taking everything into consideration, we now can present a holistic system of robot and environment model accompanied by a set of input devices, integration of core middleware (XBotCore and ROS), and useful visualizations.

This encompasses different robot models (digital twin vs. visualization model) and different simulator modes (direct control vs visualization mode). Furthermore, the integration of ROS, UDP, and XBotCore as message passing middleware, as well as the implementation of all necessary simulator internal connections (of joints, sensors, etc.) to and from ROS were established. Additionally, in terms of rendering different aspects were covered: i) the (stereoscopic) rendering in general, the rendering of raw and processed sensor data, up to specialized types like textured meshes; ii) new user interfaces, like head-up-displays and projected visualization metaphors were integrated to promote the ease of use visualizing data in the operators view or at the place of occurrence. Furthermore, benchmark scenarios and objects (real or fictional) were modeled, based on the specification requirements for M30 and M42. Finally, the online conversion of sensor data to rigid body environment models has been implemented to generate simulated worlds based on the robot's percepts.

The next step is to evaluate on-line performance of this holistic system, whereas we already conducted concepts of integration and 'switching', as well as anticipatory actions for the critical components of stereoscopic rendering on Linux and the swap of rigid body simulation to a real-time capable QNX server.

9 Appendix

In the following the model library for the M30 and M42 evaluation scenarios is presented.

Table 2: ModBib: Locomotion.


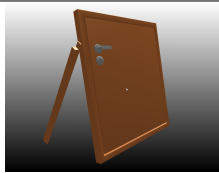

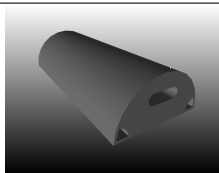

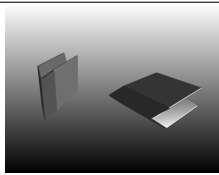
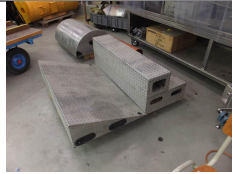
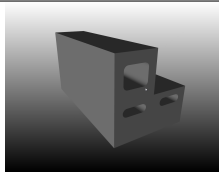

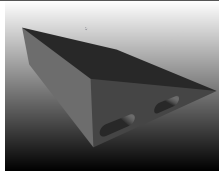

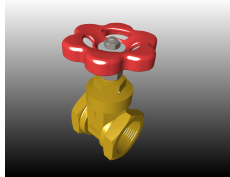
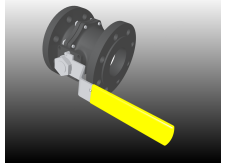

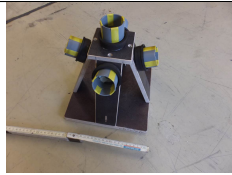
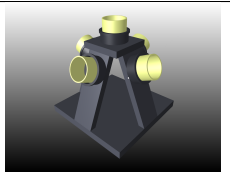
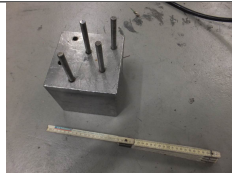
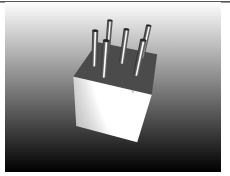
Nr.	Name of task	Real image	VEROSIM	MS
1	Narrow Passages			
1a	Opening and going through a regular door (Door equipped with a handle, unlocked)			M30
1b	Add closing mechanism and use of a key to unlock			M42
2	Obstacles			
2a	RoboCup Rescue 3D step field			M30
2b	Add debris			M42
2x	KHG obstacles			M30
2x	KHG obstacles			M30
3	Stairs			
3a	Walking up and down on a regular stairs with min. 45° (straight, change direction at landing)			M30
3b	Debris on the stairs			M42
4	Ramp			
4a	Walking up and down on a ramp of min 30°			M30
4b	Walking up and down on a ramp of min 45°			M42
5	Gap			
5a	Overcoming a gap of min. 30cm	✓	✓	M30
5b	Overcoming a gap of min. 60cm	✓	✓	M42

Table 3: ModBib: Manipulation 01.

Nr.	Name of task	Real image(s)	VEROSIM	MS
1	Valve			
1a	Opening and closing a valve			
1b	Different locations, different diameters			
				
2	Pipe Star			
2a	Inspection			
2b	Direction			
2c	Extraction			
2d	Insertion			

References

- [1] European Commission Directorate General for Communications Networks, Content and Technology Components and Systems: Robotics. Grant Agreement 644839: CENTAURO—Robust Mobility and Dexterous Manipulation in Disaster Response by Full-body Telepresence in a Centaur-like Robot, 2014.

Table 4: ModBib: Manipulation 02.

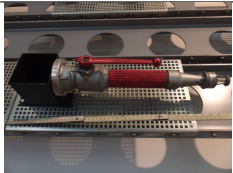
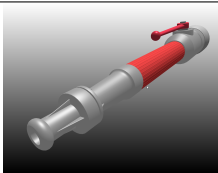
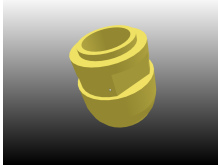
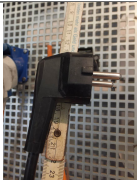
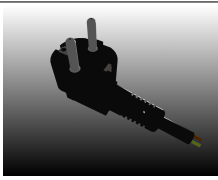

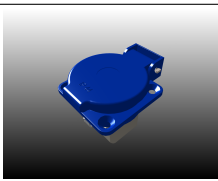

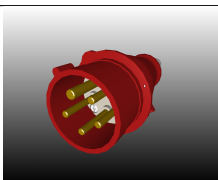

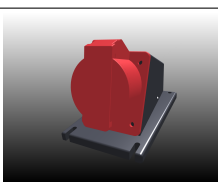






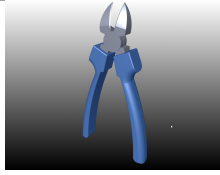
Nr.	Name of task	Real image(s)	VEROSIM	MS
3	Fire Hose Connection			
3a	Connect mobile hose to a stationary part			M30
3b	Connect two hoses (both mobile)			M42
4	Power Connection			
4a	Connect mobile hose to stationary part 220V			M30
				
4b	Connect two hoses (both mobile) 220V, 440V			M42
				

Table 5: ModBib: Manipulation 03.

Nr.	Name of task	Real image(s)	VEROSIM	MS
5	Take a Sample			
5a	Follow a surface with a sensor	✓	✓	M30
5b	Smear test	✓	✓	M42
6	Fix a Cable			
6a	Snap hook			M30
6b	Shackle			M42
7	Use Power Screw Driver			
7a	Screw is already partially in wood, use power screw driver			M30
7b	Attach a piece of wood to some static wood. Use power screw driver			M42
8	Drill a Hole			
8a	Use a single handed driller			M30
8b	Use a two-handed larger version			M42
9	Cut a Cable/Pipe			
9a	Use a cutting tool			M30
9b	Secure part that is being cut			M30