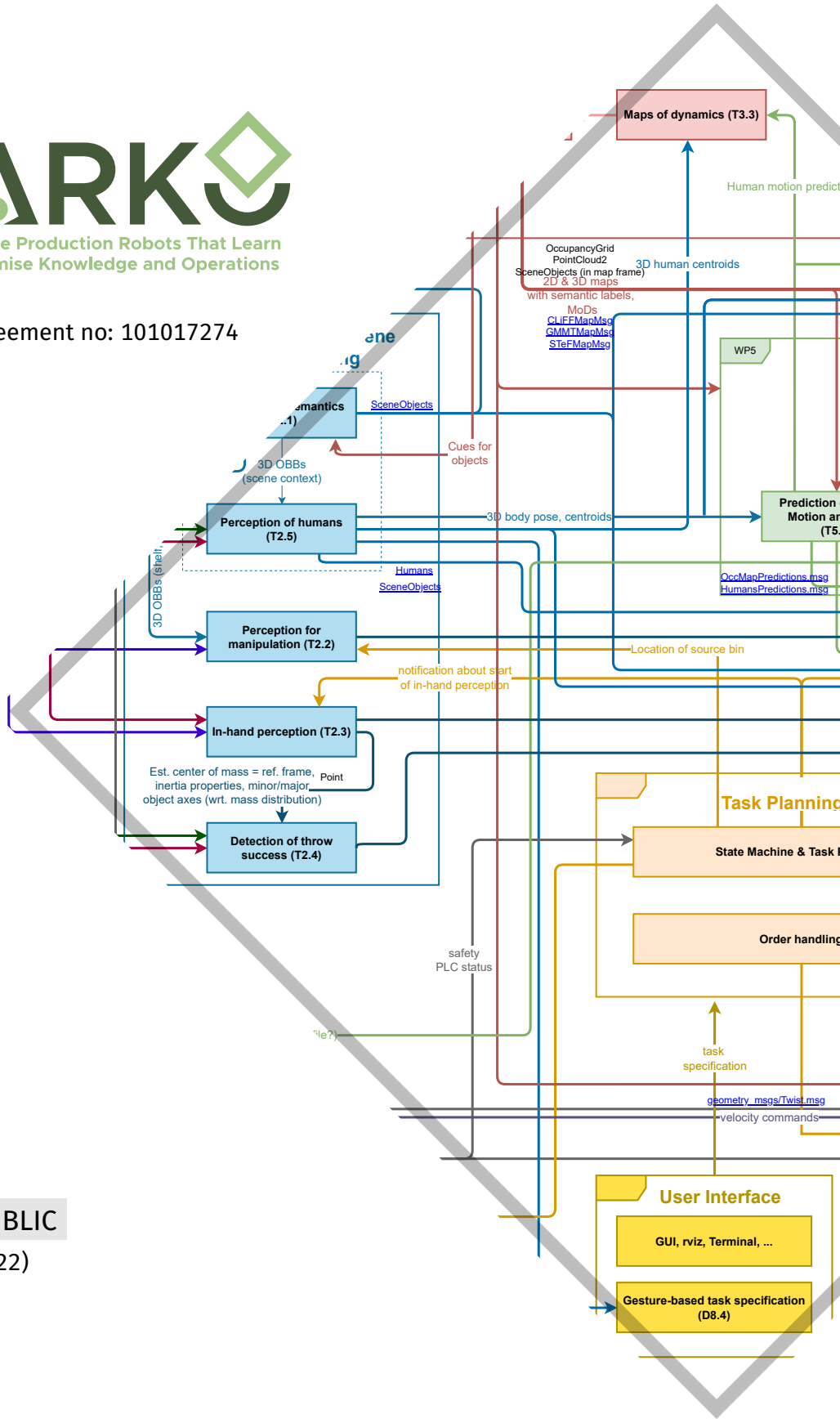
DARKO

Dynamic Agile Production Robots That Learn and Optimise Knowledge and Operations

H2020-ICT-2020-2 Grant agreement no: 101017274



DELIVERABLE 8.2
System architecture

Dissemination Level: PUBLIC

Due date: month 17 (May 2022)

Deliverable type: Other

Lead beneficiary: ORU

1 Introduction

This document covers the design and implementation plan for the DARKO system architecture. It should be seen as an executive summary of the overall system architecture that will be used throughout the project, which will continuously be updated as the development progresses and serve as documentation for the DARKO development teams internally. We outline the general software architecture, tools for deployment of components, and testing procedures using a simulation environment.

2 Architecture

We have developed a system architecture mostly based on the open-source Robot Operating System (ROS) middleware. As a consequence, DARKO's system architecture is inherently component-based, built around the principles of re-usability and low coupling. As of now, we are exclusively using ROS 1 in favour of ROS 2; and specifically, a combination of ROS Melodic and ROS Noetic. Most of the components are using Melodic, in part to ease integration with existing software provided with the robot platform from Robotnik (which is currently only available for Melodic). The consortium is discussing, and plan to work towards, a partial or full switch to ROS 2 later in the project, in order to ensure the longevity and easy uptake of the software we develop.

Given the nature of the project, on top of the middleware, we model our overall system architecture as composed of functional layers.

At the bottom, an *ability-level*, encapsulating the basic robot functionalities like sensing, control, localisation, path planning, pick and place operations, etc. This layer includes modules in particular from WP2, WP4 and WP6; facilitating low-latency, closed-loop control of individual lower level abilities of the robots.

On top of this, we have an *information layer* that encompasses more long-term and context-specific knowledge and abilities, enabling robots for safe and efficient operation in shared environments. This layer enables the robot to collect and refine knowledge about the environment it is operating in, as well as functionalities allowing to utilise this knowledge. Hence, this layer comprises primarily modules developed in WP3 and WP6.

Finally, a *risk-aware execution layer* includes modules from WP7 to assure that the system will execute its tasks in a safe way.

A high-level view of this composition is shown in Figure 1.

The overall architecture is presented more concretely as a flow chart in Figure 2. This flow chart captures the main directions of data flow on a conceptual level, and abstracts from the actual software packages to be integrated in DARKO. The flow chart is organised around work packages rather than the layers described above, but the risk-aware execution layer comprises the modules in WP7 as well as the beige Task Planning box which is closely connected to WP7, the information layer comprises the mapping and map server modules in WP3 as well as the global motion planning in T6.3, and the ability layer includes the remaining modules, which all act directly on live data.

While Figure 2 captures the functional architecture that guides the development and integration in DARKO, Figure 3 is an automatically generated snapshot of the status of software running on the live robot platform, and their run-time connections.

3 Usage and deployment

Within the project, we have agreed to use a git-based workflow for source code management. We have a project-central repository hosted at a private GitLab instance at ORU. A

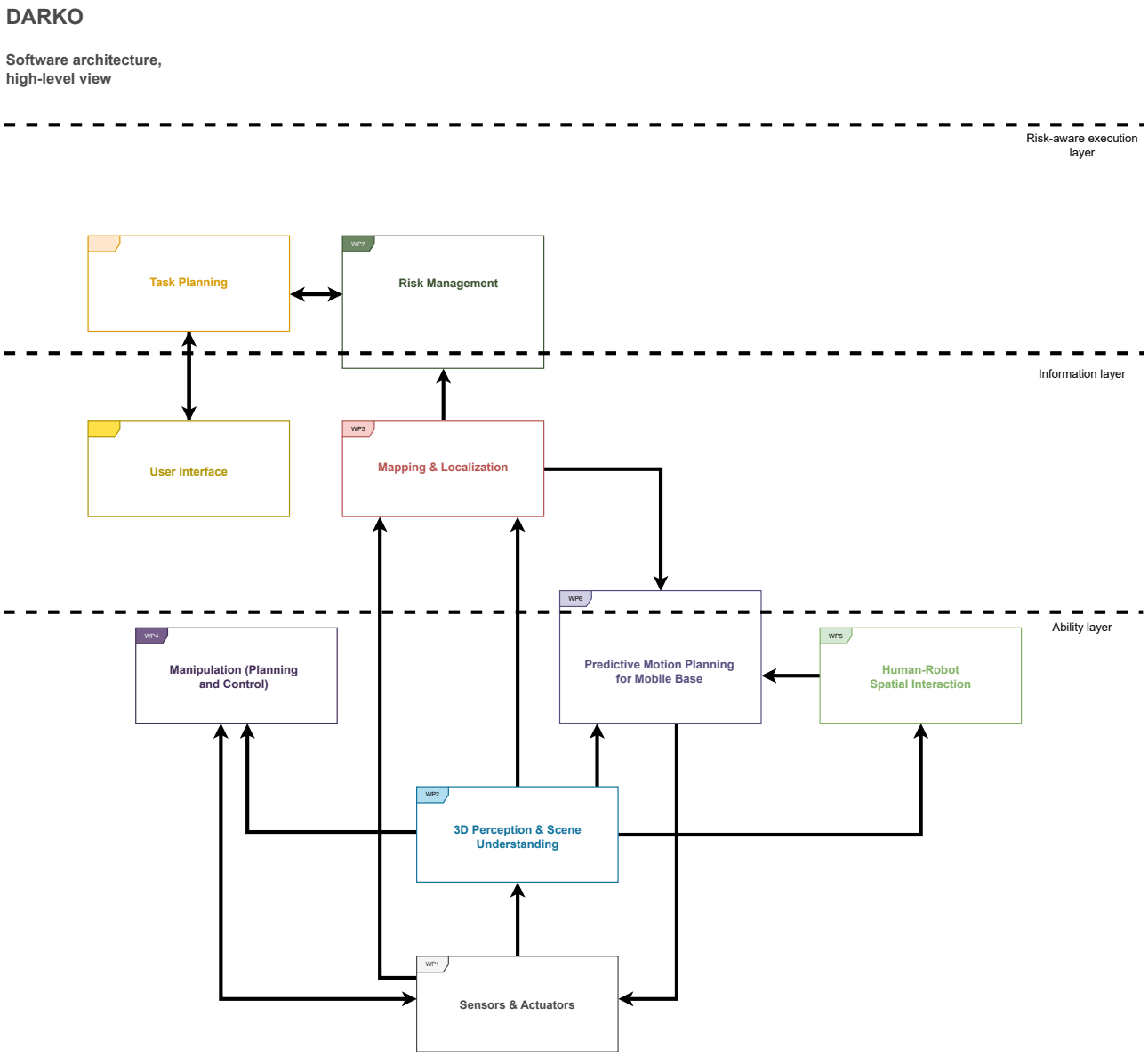


Figure 1: High-level view of the DARKO software architecture.

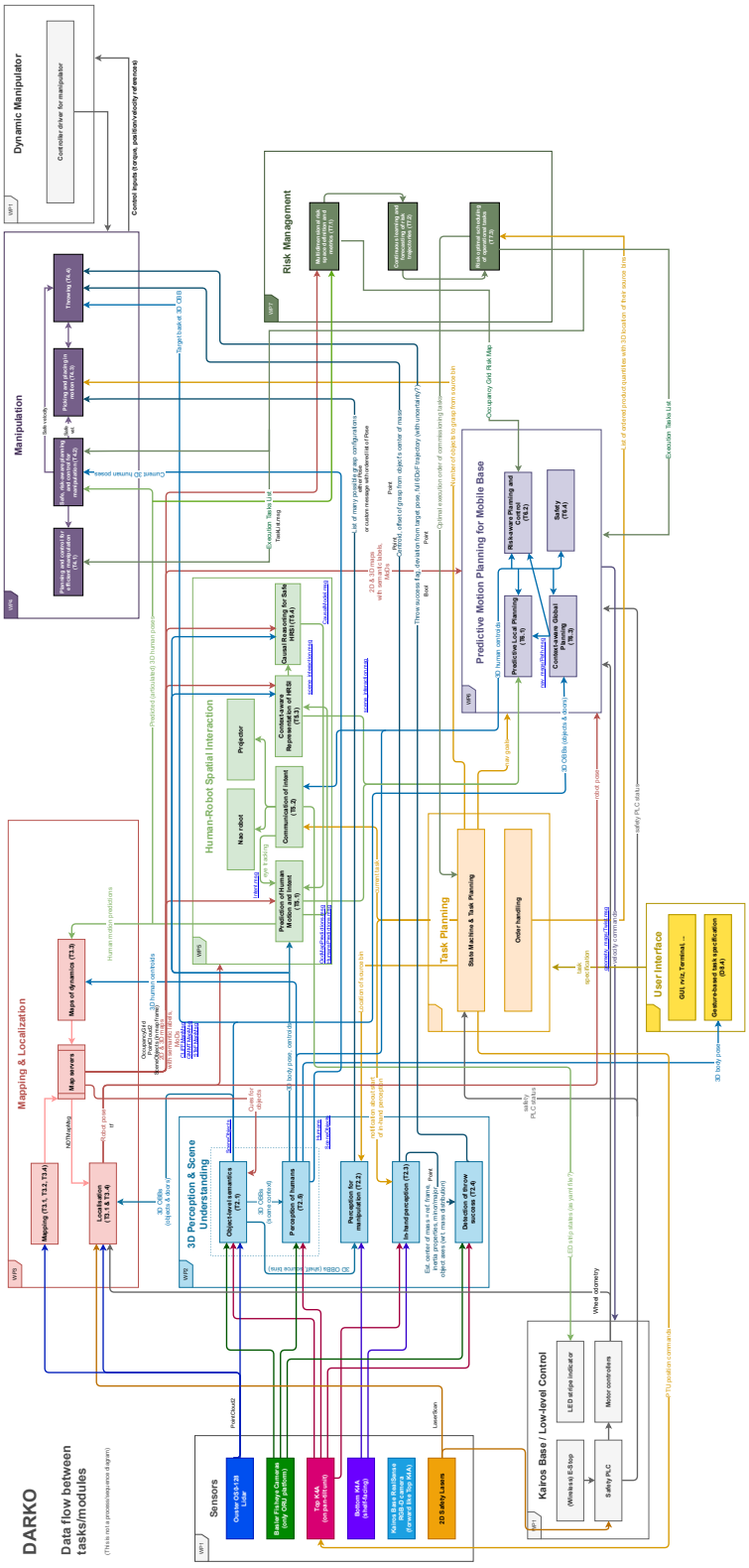


Figure 2: Overall flow chart of the main modules involved in DARKO's software architecture, and the types of data passed between the modules as input and output.

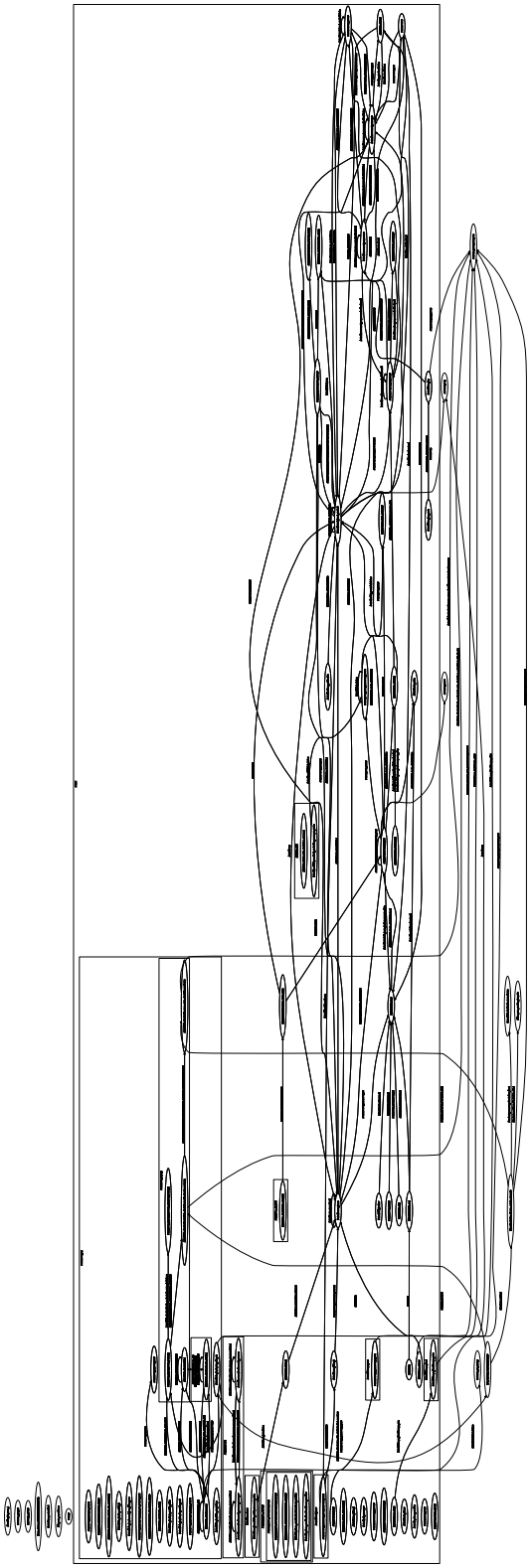


Figure 3: Snapshot of the ROS computation graph, filtered for the most relevant software nodes running live on the robot, and their connections.

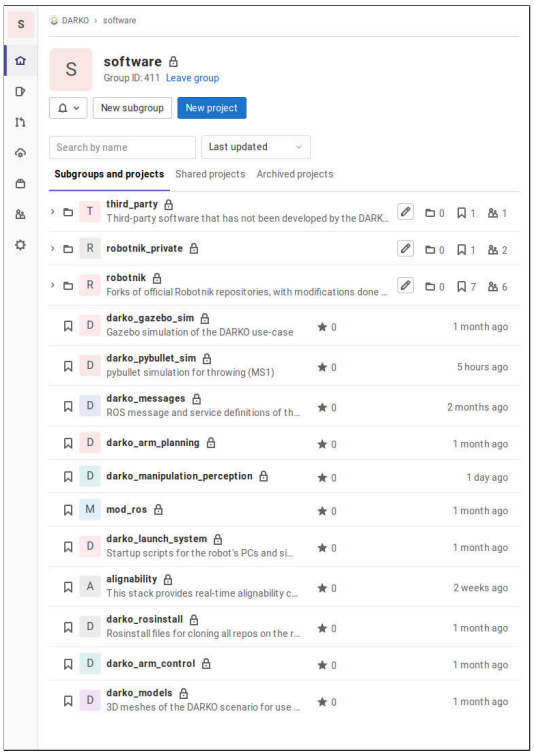


Figure 4: DARKO software repositories hosted at the project-internal GitLab instance at ORU.

list of currently available DARKO software packages hosted there is captured in Figure 4. These repositories includes further links to packages hosted elsewhere, such as public GitHub repositores. We also maintain a separate public-facing GitLab space.

Certain software components from industrial partner Bosch, which include intellectual property that needs to be protected, are shared with partners in binary form as ROS Debian packages (.deb) or Docker images. Selected components will be open-sourced at a later point during the project once the necessary release processes have been executed.

In order to synchronise the software modules installed and running on the three DARKO robot platforms in the consortium, we maintain *rosinstall* files to be used for cloning all repositories at the correct version, in an organised way. Likewise, the DARKO simulator that we develop for facilitating testing and integration (see Section 4), will use a similar rosinstall workspace description.

We make use of wiki pages in the DARKO GitLab space to share documentation about the overall workflow, accessing and operating the robot platforms, as well as usage of certain software modules (see Figure 5.) We track progress, especially towards project milestones, using the built-in issue tracker in GitLab, which makes it easy to keep track of outstanding issues, deadlines, and responsibilities.

The DARKO robots are connected to a virtual private network (VPN) via vpn.aass.oru.se, to which project developers can connect in order to easily work directly on the robot without being physically present. This setup makes it easy to collaborate towards deployment of DARKO software (including debugging and refinement) even in an internationally distributed project such as this.

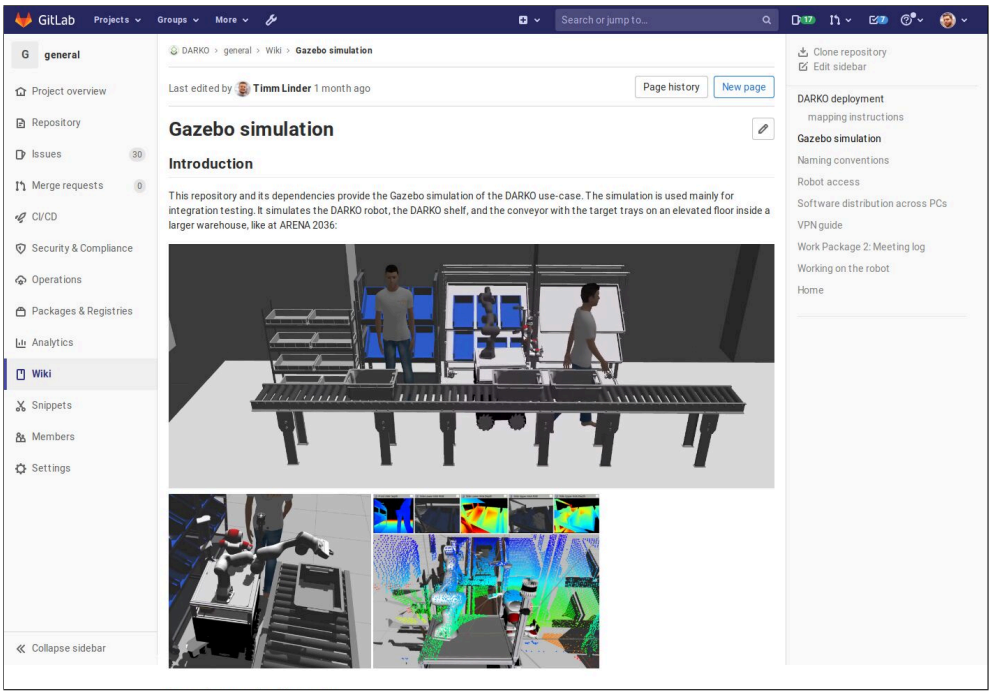


Figure 5: The DARKO project-internal wiki

4 Testing and simulation

Gazebo simulation for integration testing

For integration testing of the DARKO software, we have developed a fully ROS-integrated simulation using the Gazebo simulator. We are currently using Gazebo 9 in a Docker container running ROS Melodic, which matches the software versions run on the real robot's main computer.

The Gazebo simulation approximately replicates the initial DARKO robot platform with its sensor setup along with a Franka Panda manipulator and a standard gripper. The sensors are placed in the same locations as on the real robot. While we replicate the most important sensor characteristics, such as type of camera lens (rectilinear vs. fisheye), or 3D lidar resolution, we do not accurately model noise effects since the goal of the simulation is not to train or evaluate models for perception. Instead, the focus is to replicate a typical workflow from picking of objects from a source bin by moving both the mobile base and the manipulator, initiating a throwing action, and thereby placing the object into a target bin. For this purpose, we simulate the BSH shelf replica, the boxes and trays as well as a conveyor belt using CAD models. Items for picking are represented using simplified primitives, since accurately modeling plastic bags etc. and the resulting contact forces for gripping is out of the scope of this simulator. Both static and walking humans have been added to the simulated warehouse scene to enable testing of components from work package 5.

To simplify testing of components in both simulation and on the real robot, we have matched the ROS topic names, namespaces and TF frame ID as closely as possible between simulation and hardware.

For certain components that are computationally complex, such as the perception pipeline, we also provide drop-in mock components (with the same ROS interface) that e.g.

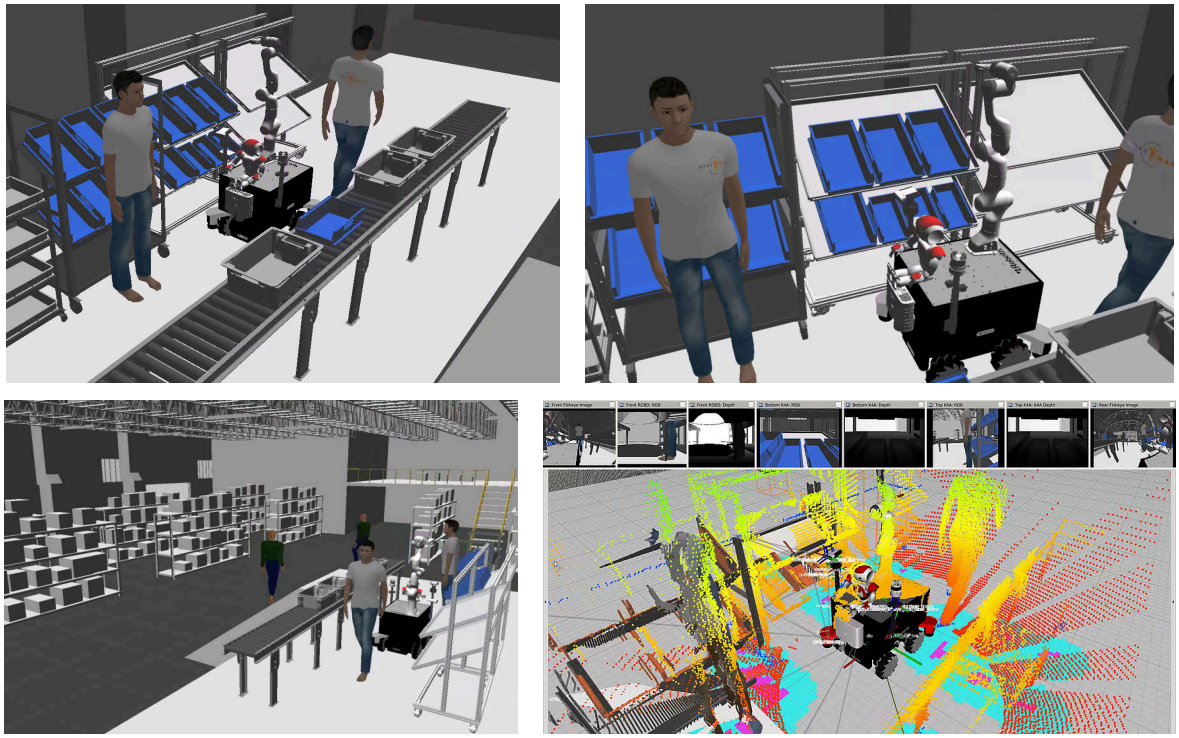


Figure 6: Gazebo simulation of the DARKO use-case in a simulated warehouse with robot platform, BSH shelf replica, conveyor and static and walking humans. The bottom right picture shows the resulting visualization of RGB-D, lidar and fisheye sensor data in Rviz using the same sensor arrangement as on the real robot.

provide ground-truth human and object locations to downstream components, without requiring each developer of those components to run the full perception stack on his or her development computer. This way, the simulation can run in approximately real-time.

Physics-based simulation for throwing using pyBullet

Physical effects such as accurate multi-contact forces, nonlinear object flying dynamics are notoriously hard to model correctly in Gazebo. Due to this and for historic reasons, a separate physics-based simulator has been developed for the throwing task. This simulator is based upon pyBullet (a Python wrapper around the Bullet physics engine) and currently not coupled with the Gazebo simulation. A picture of the throwing simulation with the mobile manipulator is shown in Figure 7 (bottom). The simulation script takes target box position relative to the mobile manipulator as argument, calculates batches of valid throwing trajectories within seconds and simulate the minimium-time one.

Since in this phase, the simulation is for the purpose of finding valid joint space throwing configuration in a reliable and computationally efficient manner, we make a number of assumptions; such as known accurate flying dynamics, perfect grasping and trajectory tracking. In future work, we will relax the assumptions and extend our work to close the sim-to-real gap, and to replicate the joint states from the pyBullet simulation into Gazebo.

5 Milestone 1

As DARKO's Milestone 1 we have implemented a concrete deployment of the software architecture. Going beyond the simulation that was originally planned, we have implemented a demonstration that includes prototype versions of most of the modules from the DARKO work packages live on the initial robot platform (from D1.1). Some impressions of the milestone demonstration can be seen in Figure 7.

The milestone demonstration includes live versions of

- basic mapping,
- basic navigation,
- human pose tracking (tracking of centroids, detection of full body pose),
- short- and long-term human motion prediction,
- prototype object-level semantics (detecting the blue box for picking),
- integration of perception (Bosch), grasp pose generation (ORU), and grasp execution (UNIPI) via the defined interfaces (although the grasp pose is a hard-coded dummy value since we do not yet have a trained model for the target BSH objects),
- alignment prediction and verification (for estimating localisation risk),
- creating and publishing long-term maps of dynamics,
- mock-up modules for context-aware human-robot spatial interaction and causal discovery.

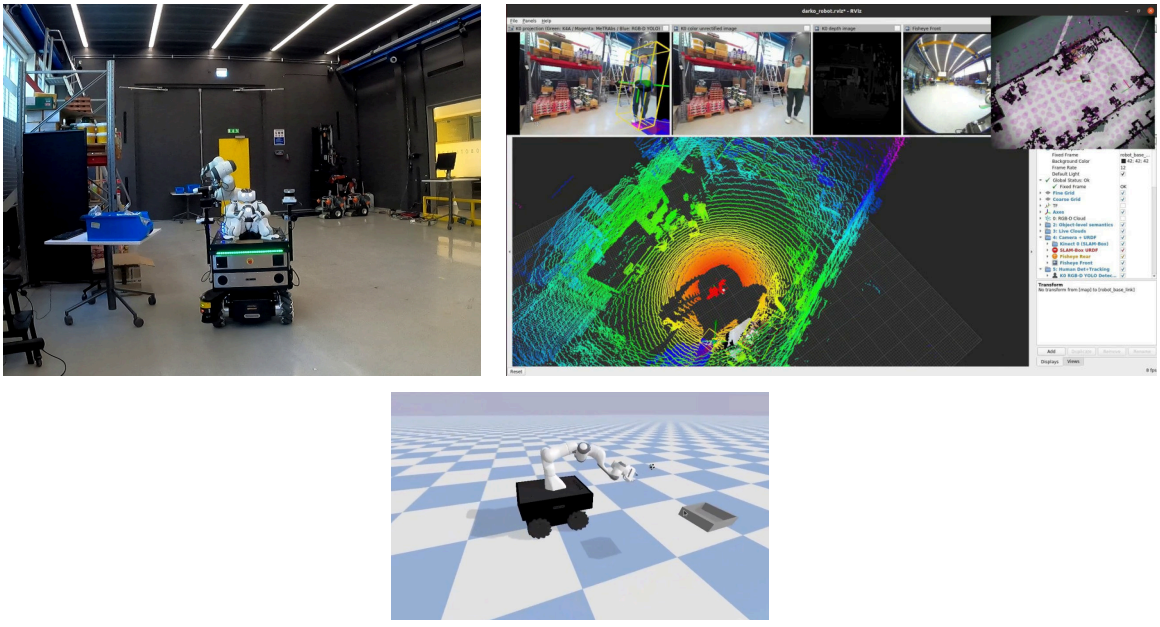


Figure 7: Snapshots from the MS1 demo, with live integration of deployment, human-aware navigation, and prototype manipulation components on the DARKO robot platform, plus physics-based simulation of throwing using the pyBullet simulation.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101017274