



Sustainable and reliable robotics for part handling in manufacturing

Project no.: 610917
Project full title: Sustainable and reliable robotics for part handling in manufacturing
Project Acronym: STAMINA
Deliverable no.: D1.3.7
Title of the deliverable: STAMINA test and evaluation report M32

Contractual Date of Delivery to the CEC: 31.07.2016
Actual Date of Delivery to the CEC: 30.07.2016
Organisation name of lead contractor for this deliverable: Peugeot Citroën Automobiles S.A.
Author(s): Arnaud Chazoule, Volker Krüger, German Martin, Alexander Schiotka, Lazaros Nalpantidis, Cesar Toscano, Ron Petrick, Matthew Crosby
Participants(s): P01, P02, P04, P05, P06, P07
Work package contributing to the deliverable: WP1
Nature: R
Version: 1.0
Total number of pages: 43
Start date of project: 01.10.2013
Duration: 42 months – 31.03.2017

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 610917

Dissemination Level

PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract:

This deliverable provides a report on the tests performed during the third test sprint. This performance report will provide a structured feedback to the following R&D phase and prepare the next test sprint.

Document History

Version	Date	Author (Unit)	Description
0.1	08.06.2016	A. Chazoule	Initialization
0.2	30.06.2016	V. Krueger	Complete overhaul
1.0	27.07.2016	A. Chazoule	Some minor modifications + publications

Executive Summary	4
1 Introduction: Objectives of the third test sprint	5
2 Description of the test sprint environment	5
3 Integrating and testing a complete solution	8
3.1 Picking	8
3.1.1 Preparation phase.....	8
3.1.2 Target for the third test sprint.....	9
3.1.3 Results.....	9
3.1.4 Contingency plan for picking-skill	10
3.2 Navigation	11
3.2.1 Improvements since last time	11
3.2.2 Test of mapping, localization and navigation in different environment	12
3.3 Calibration	14
3.3.1 Manual Camera Calibration	14
3.3.2 Automatic Camera Calibration	15
3.4 Skill Primitives	15
3.4.1 Skill-Primitive locate_bonn.....	16
3.4.2 Skill-Primitive registration	16
3.4.3 Skill-Primitive arm_motion	17
3.4.4 Skill-Primitive kittingbox_registration	17
3.4.5 Learning Primitives grasping_pose_learn, placing_pose_learn, object_train	18
4 SkiROS and Skills	20
4.1 Progress since last time	20
4.2 Preparation of SkiROS for use	22
4.2.1 On-site functional testing	22
4.2.2 STAMINA system configuration.....	22
5 Configuring the Logistic Planner	23
5.1 Mapping the kitting zone and constructing the logistic world model	23
5.2 Teaching the docking positions	32
5.3 Retrieving the skills from the robot	32
5.4 Creating the kitting order	32
6 Mission Planner Integration	34
7 Complete system test	34
7.1 Introduction	34
7.2 Complete System Functional Testing	35
7.2.1 Evaluations of the Skills within the complete system	37
7.2.2 Evaluation of the Skill-Primitives within the complete system	39
7.3 Error Recovery Capability	41
8 Conclusions	41
9 Appendix	42

Executive Summary

The agile development methodology applied in STAMINA relies on iterative and incremental test sprints for managing the integration, testing and structured feedback to RTD. Four test cycles have been planned during which all partners will be focused on the test in order to provide flawless runs and sustainable feedback to the R&D partners.

The present document is the **test and evaluation report for the second test cycle**. At this stage, the focus lies on the generalization and combination of the previous developments.

The document is decomposed as follows: the reader will find a general description of the test sprint organization in the [introductory section](#). The rest of the document is structured in relation with the test sprint scheduling, following a bottom-up approach: [Section 3](#) describes the integration and testing of the complete system including skills (navigation, picking, placing); [Section 4](#) presents the integration of the skill planner; [Section 5](#) demonstrates how the Logistic planner works and is configured by the logistic technician; [Section 6](#) describes the integration of the Mission Planner. Finally, [Section 7](#) discusses the system test on a complete run and identifies future work for the next test sprint. The reader will find in the [Appendix](#) two papers submitted for publication summarizing the experiments for picking, placing and vertical integration.

Note that this document is identical to Deliverable D1.3.3 (CO, Confidential version).

1 Introduction: Objectives of the third test sprint

The third test sprint of the STAMINA project took place from M32 to M36. The overall target of this test sprint according to Task 1.3 in the DOW is as follows:

On a third and fourth testing cycles test the focus will be on the evaluation of the complete system namely through usability testing. These usability tests will target both the end users, which include blue-collar workers and forklift drivers but also the systems integrator. The evaluation of the setup time and the engineering efforts required to setup and run the STAMINA system constitute an important foundation for the future exploitation of the project.

In this test sprint, we have for the first time tested the *complete* system in the relevant environment. Already last time, all skills were integrated into the complete system, but due to the limited robustness of each of the skills and the SkiROS system, it was not possible to successfully execute complete missions.

In this test sprint, we have therefore focused on tested the execution of complete missions and generated large amounts of statistics about detectable, undetectable and predictable errors. Concrete and detailed user-testing will be done in an upcoming test sprint shortly before the delivery time of this report. Those test results will be summarized in deliverable D5.3.

To facilitate the evaluation we have identified relevant performance indicators to quantify the functional performance of the system, its reliability, its effectiveness and resilience to disturbances.

The report is structured as follows: In Sect. 2, we will give an overview of the test sprint environment at PSA. Sect. 3 will summarize a) the skill improvements since last time and b) summarize what preparations were specifically needed to execute tests. This will reflect the effort of starting up a STAMINA robot system within a new environment. Sec. 4 and 5 will focus on the configuration of the task and logistic planners, respectively. Again, this will give an insight into what efforts are needed to setup a system within a completely new environment. Section 7 will summarize the experimental results.

2 Description of the test sprint environment

PSA has created a new replica of a small logistics kitting zone (see picture below) with containers and shelves for testing the basic robot skills (navigation, part localization, picking, placing). Figure 1 shows four isles with a number of pallets and shelves.

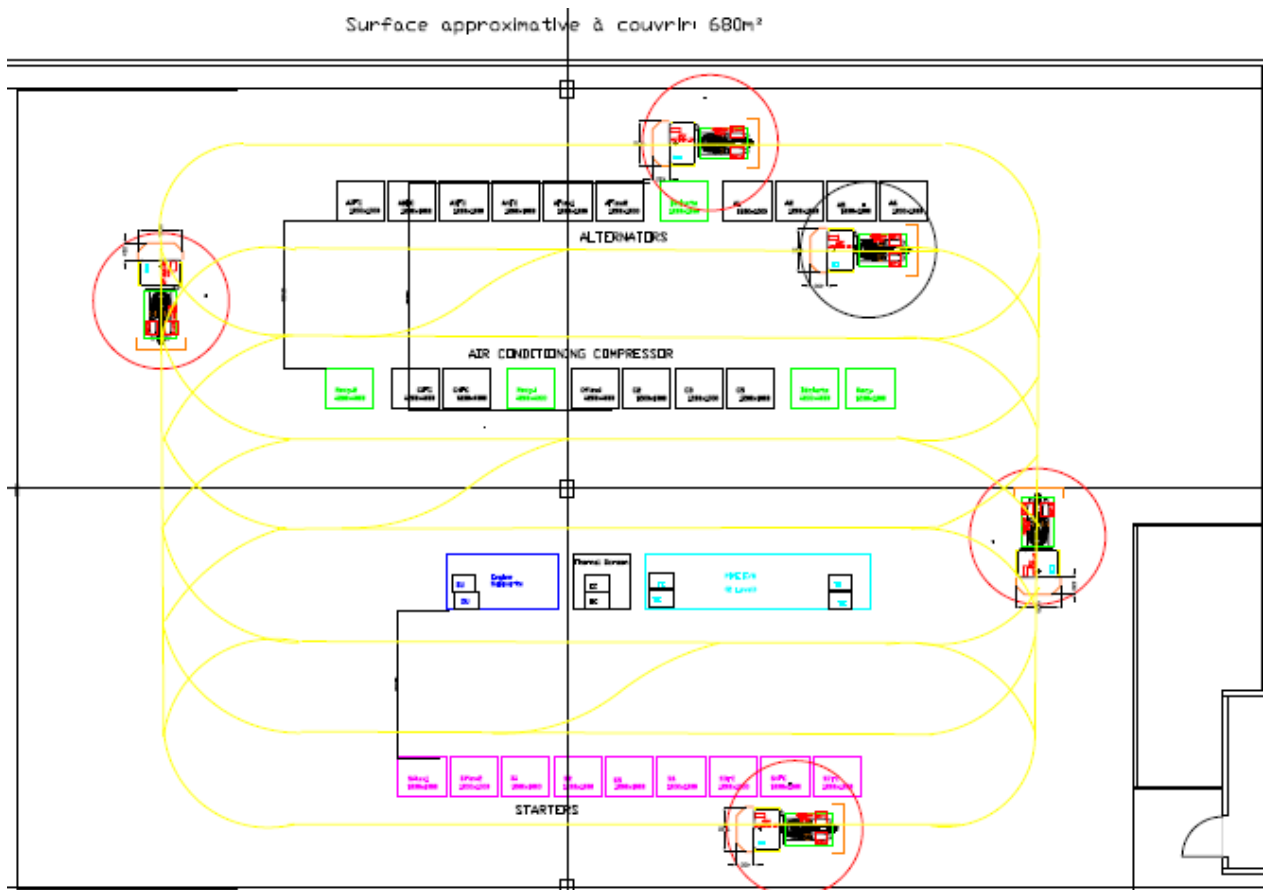


Figure 1 shows the layout of the kitting supermarket for the third test sprint.



Figure 2 shows the STAMINA robot at PSA...

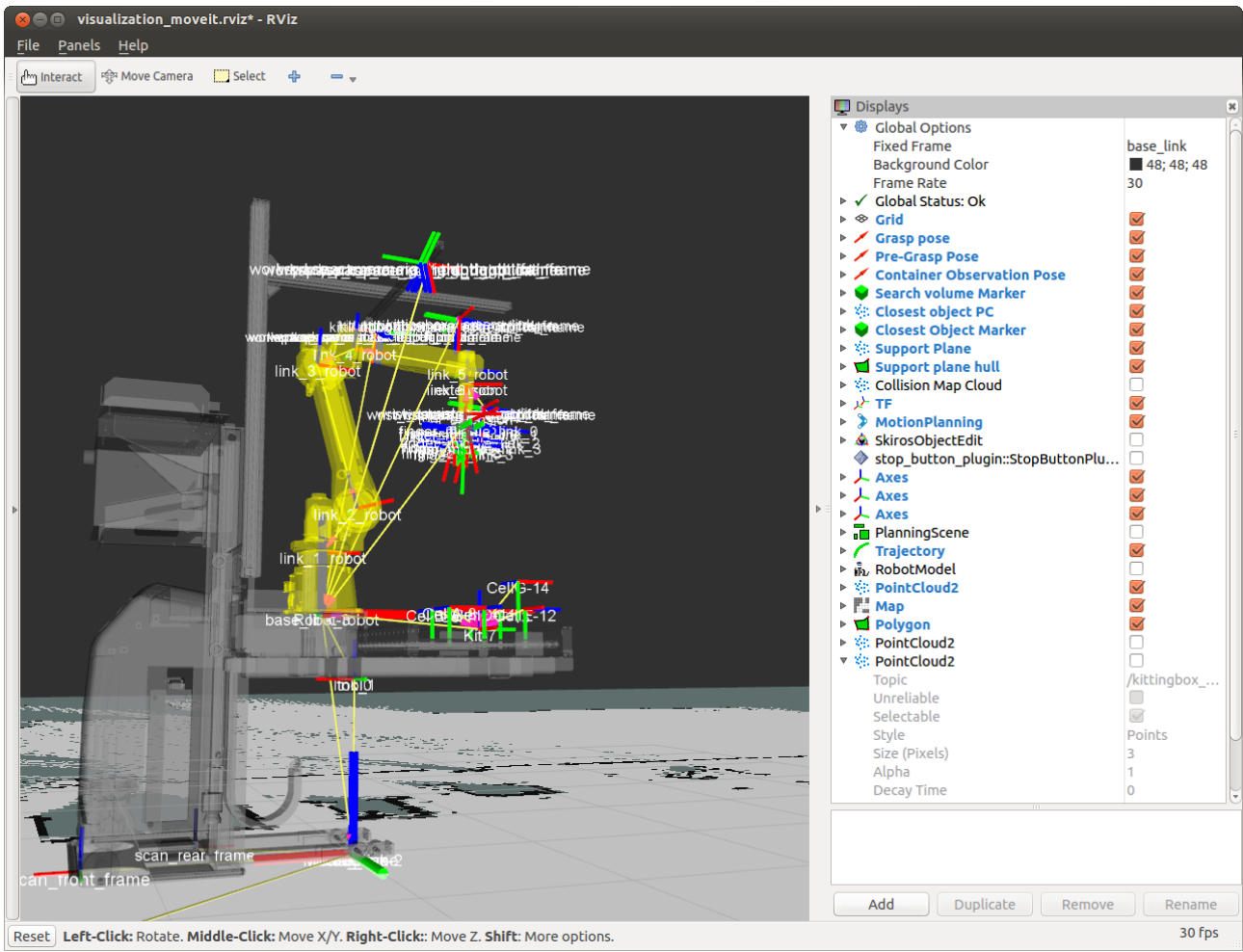


Figure 3 ... and its RViz virtual model for simulation

3 Integrating and testing a complete solution

Our goal to this test sprint was to deliver and test a complete system using all skills tested separately into the second tests sprint. In the following section, we will discuss topics that are specific to the different skills, including

- improvements,
- shortcomings,
- failures,
- preparations needed *before* arriving to the test-site,
- preparations needed *at* the test site.

The picking skill was a great challenge and in spite of great efforts, we had to resort to a contingency plan by using a limited picking skill.

3.1 Picking

3.1.1 Preparation phase

In preparation to the 3rd test sprint UBO focused on mitigating integration issues of the picking pipeline as part of the skill architecture. The conflict originated in different assumptions made by two independent software components – the skill architecture (AAU) and the part training and locating software (UBO) – about the base frame of reference. As a result, objects could not be localized for picking at the BAS. Additionally, motion planning was affected as a result of the mismatch. In the months leading up to the 3rd test sprint UBO attempted to solve the problem remotely. However, owing to a number of hardware limitations on the experimental site at UBO the issue could not be reproduced and, hence, the solution could not be immediately found.

When on site at BAS, UBO managed to resolve the conflict between the software components. In addition, UBO performed a few successful tests before the start of the 3rd test sprint on a stationary platform.

UBO has also continued the development of the bin-picking skill. For this purpose, a method for container detection and pose estimation of engine pipes using RGB-D data has been developed. The new method is based on RGB-D edges and is illustrated in Figure 1.

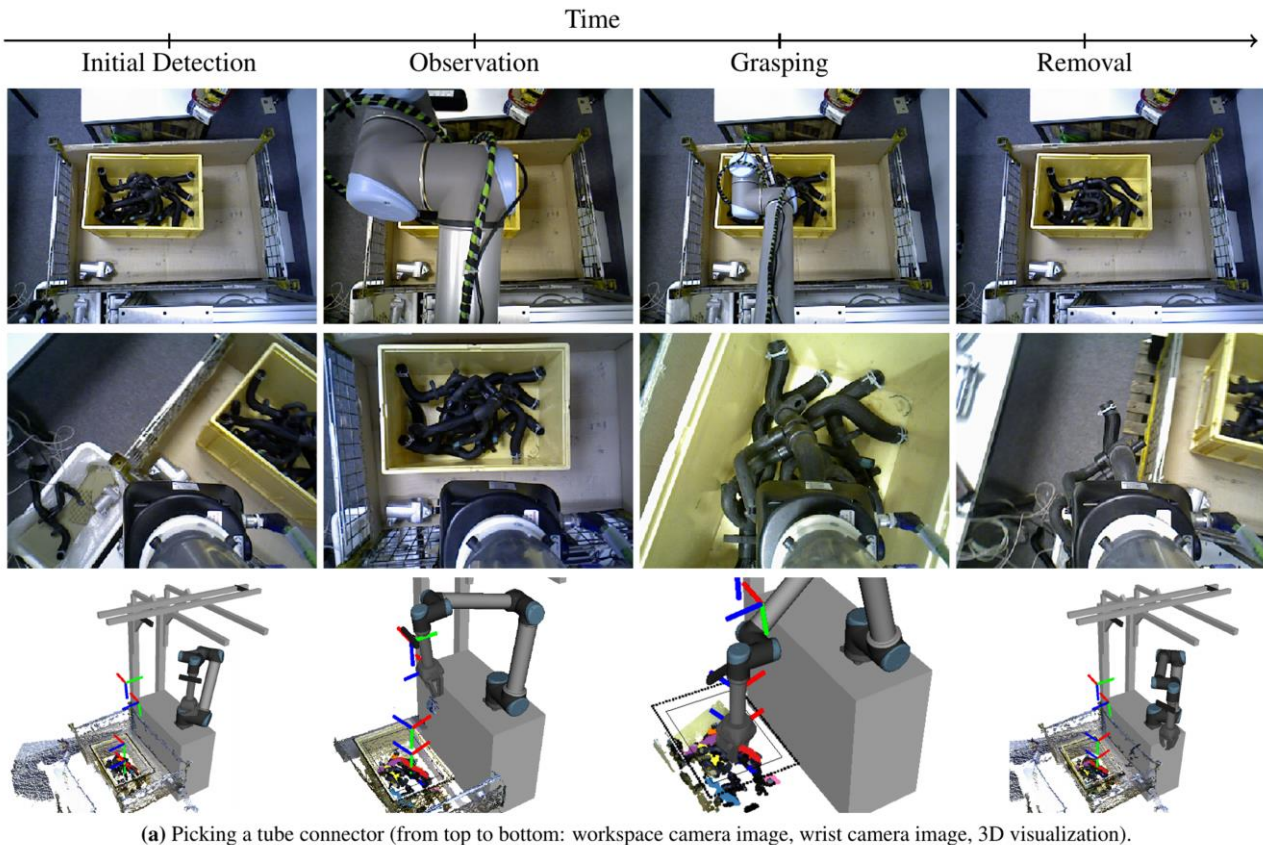


Figure 1: Box detection and bin-picking of engine pipes

3.1.2 Target for the third test sprint

The goals of the 3rd test sprint for UBO were:

- to test and evaluate the performance of the training and picking skills independently from other software components with the final demonstrator at PSA (stationary tests);
- to test and evaluate the performance of the training and picking skills in conjunction with the other software components (e.g. navigation, placing skill, etc.) with the final demonstrator at PSA.

The tests were conducted in allocated 2-hour timeslots during each day of the test sprint. The results of the experiments were categorised in the binary fashion (i.e. success of failure) for each test component depending on the outcome. Due to time limits no evaluation of the container detection and pose estimation of engine pipes could be performed.

3.1.3 Results

The following results were obtained during the third test sprint with the final demonstrator platform at the end-user site of PSA.

3.1.3.1 Stationary tests

UBO performed systematic tests with four different objects: the alternator, compressor, starter and the engine support. UBO taught the system grasping poses and performed 5 picking tests for each object placed in different poses. The tests were carried out with the pose of the mobile platform

fixed with respect to the pallet, i.e. the platform stayed immobile between the training and testing stages. All of the tests were successful.

Test results for training individual parts

Alternator	OK
Starter	OK
Engine support	OK
Compressor	OK

Test results for picking individual parts

Alternator	OK
Starter	OK
Engine support	OK
Compressor	OK

3.1.3.2 Integration tests

In integration tests the trained parts were placed in different pallets. It was observed that the perception of the parts was sensitive to the changes in their orientation: the pose of an object trained from one side of the pallet would not be correctly estimated if the robot approached the pallet from the other side.

Motion planning was another source of errors. Variability of the platform pose relative to the pallet means that objects would appear at distant positions which complicates the computation of a motion plan to reach them. This was not observed in the UBO lab, since the platform is stationary (typically 5 planning attempts suffice).

The related second problem concerns the segmentation. The initial segmentation is performed from the workspace camera, which in the UBO lab is centred above the pallet. If this is not the case, there is a chance of failure for parts with protuberances: the perspective of the camera might result in significant self-occlusions leading to the object segment being split in two.

3.1.4 Contingency plan for picking-skill

Early on it became apparent that differences between the STAMINA robot hardware and the lab setup at Bonn might be problematic. Therefore, a contingency plan was worked out.

At AAU, a new pick skill was designed, `pick_aau`. It was modelled using multiple skill primitives, see Fig. 3. Hardware related primitives like `OpenGripper`, `CloseGripper` or `MoveArm` constitute implementations of drivers are not discussed in further details.

More interesting, however, are the generative primitives like `Locate`, `Locate kit`, `Register` or `PlanMove` which may contain more complex algorithms.

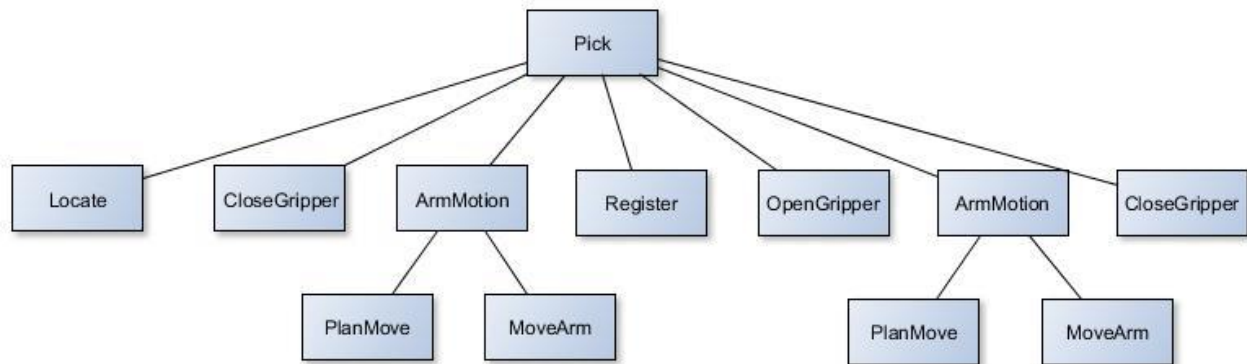


Figure 4 shows the sequence of primitives composing the `pick_aau` skill.

Perception and motion planning primitives are the key aspects in the kitting pipeline. Due to the modular structure of SkiROS, all implementations of primitive can be exchangeably used in multiple skills. Our new motion planning primitive, e.g., has been without further development successfully integrated into the picking skill as well. The procedure is logically the same as the `pick_bonn` skill. Moreover, the `Locate` and `Register` modules use exactly the same algorithms used in the `pick_bonn` skill. The only major difference is the `arm_motion` module (i.e., the Motion Planning module) that differs from the `pick_bonn`, because it uses a different approach, see Sect. 3.4.3. At the moment, `pick_aau` primitive is more reliable because the motion planning is working better compared to the one used in the `pick_bonn`. Nevertheless, as `pick_aau` is just a temporary contingency replacement of the `pick_bonn`, it doesn't implement all the features supported in the latter. In particular, it is possible to approach the objects only with one view, with the wrist camera pointing to the front of the pallet.

3.2 Navigation

3.2.1 Improvements since last time

In order to prepare the navigation skill for the new test sprint ALU-FR implemented new features and ran extensive tests in a new environment at PSA in Rennes. The following subsections summarize all modifications for the software components of the navigation skill with regard to the test-sprint. Moreover, subsection 2.4 embraces a description of preceding tests in the environment of the upcoming test-sprint.

3.2.1.1 Backward driving

Thus far the navigation module only allowed forward motions of the robot which can result in inefficient movement behavior on the predefined graph-structure. For an improvement of the path execution time ALU-FR developed the feature of backward motions. This makes direction changes on every point of the graph possible. It also allows a directional control of the robot's orientation through turning points on the graph. Through this feature an expert-user can set the robot's orientation for parts of the predefined graph.

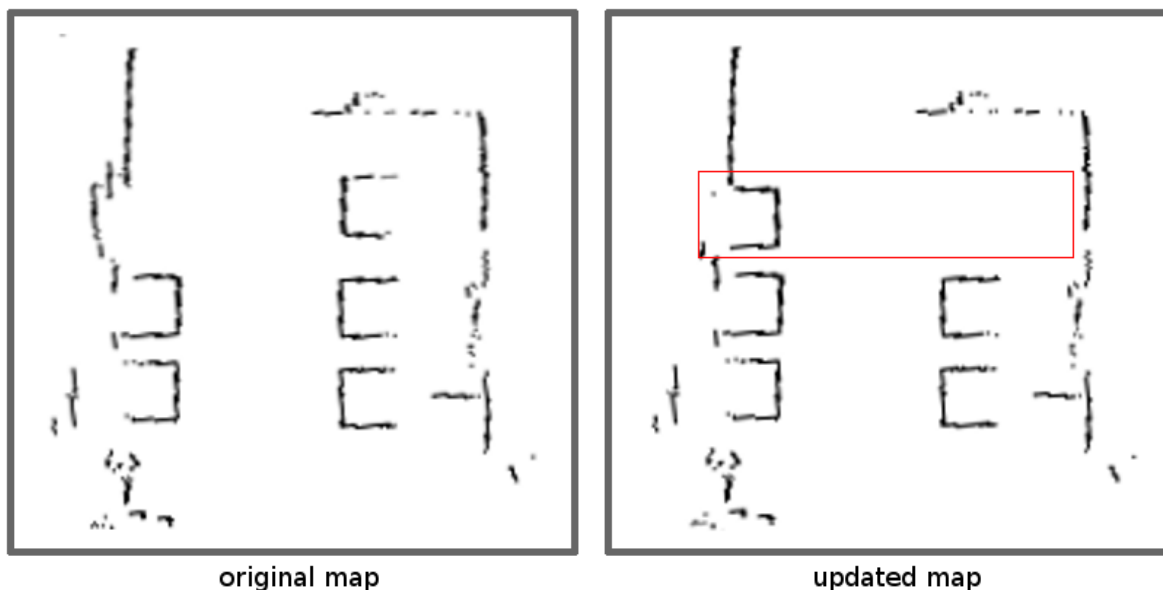
3.2.1.2 UI Improvements towards ease of use

In order to maximize the ease of use by an expert end-user ALU-FR improved the visualization of the graph-structure and the robot.

In particular this concerns the robot contour, naming of waypoints and motion arrows for the robot while moving.

3.2.1.3 Remapping of designated areas

In order to minimize the effort of an expert user for mapping of partly changed environments ALU-FR implemented a remapping of selectable areas. The technique performs mapping with known poses while it only updates cells in an occupancy grid that are located in a preselected area. This has the advantage that it is not necessary to remap the whole area. A user can specify this area or it can be specified by higher level software components. For example, if the world model registers that a pallet got replaced or shelves got reconfigured, it can request the remapping of an area. We create a local grid map and each map-cell represents the probability of occupancy of the environment. We update the map by integrating new sensor measurements from the LiDAR. For this we perform ray casting for each laser beam on the grid map. When saving the map we merge the original and the local map. An example of an original map and an updated map is shown in the figure below. On the left the figure shows the original map and on the right an updated version where all cells in the red bordered area of the grid map are updated.



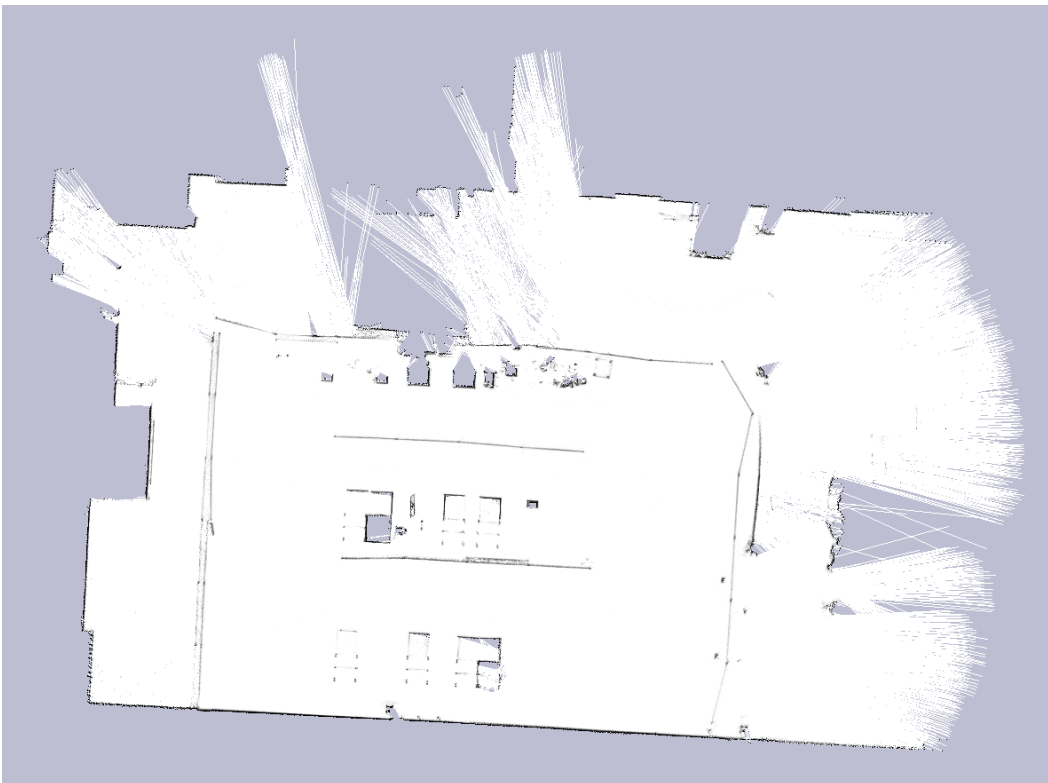
3.2.2 Test of mapping, localization and navigation in different environment

The challenges for the navigation module were twofold. Firstly the bigger area for the test-sprint can lead to more laser-beams hitting the hardware's maximum range. This can cause inaccuracies for the localization module in specific areas of the map where the robot mostly senses features that are further away than the maximum range of the laser scanner.

Secondly the test sprint area is surrounded by a fence which has a thin lower edge. Depending on the robots pitch-angle the 2d laser scanner can either sense the fence or the features behind it. This can also lead to inaccuracy regarding localization.

The globally consistent grid map of the environment in Rennes is visualized in the figure below. It shows the fence around the test-area as well as the outer walls of the hall. Especially on the right edge of the figure there are mainly laser-measurements that hit the max-range of the on-board laser-scanner.

After constructing a navigation-graph based on the globally consistent map ALU-FR conducted experiments to ensure a high localization precision. The procedure was to observe the uncertainty of the localization while navigating over the whole graph. As a result there was no situation where the tracking of the robot's position failed or lead to an unusual uncertainty in the pose estimate.



The next step was to test the navigation. ALU-FR performed multiple runs of arbitrary start- and end-poses of the robot where the driving behavior and the accuracy of reaching the end-pose were observed. After extensive testing ALU-FR occasionally observed an undesirable navigation behavior which described oscillations of small amplitude. This behavior became apparent on straight parts of the predefined graph. Since the robot doesn't correct its rotation on the goal position the described behavior lead to a rotational error. This can practically lead to inaccurate positioning in front of shelves regarding rotation.

After adjusting both the frequency of the controller and the rotational acceleration ALU-FR didn't observe any further undesirable navigation behavior. Instead the robot moved in a reliable and smooth manner to all randomly chosen goal-points on the navigation graph.

With the abovementioned parameter adjustment ALU-FR was able to increase the velocity of the robot while navigating. This applies to all parts of the map.

3.3 Calibration

3.3.1 Manual Camera Calibration

The camera calibration for all vision-related tasks was done manually so far. It is presently a very time-consuming task as the view of the camera has to be validated objectively in the RViz User Interface by aligning it the robot model and to the other cameras, see Figure 5. Without experience the tilting and rotation of the camera becomes a cumbersome job. Luckily, this procedure usually has to be done only once during the setup phase of the robot. Nevertheless, the calibration for all cameras can take several hours and has to be done before any of the cameras can be used by the skills. An initial accurate calibration is crucial and has been carried out for the 3rd test print by BAS before the other partners arrived.

Calibrating the cameras once in the beginning has a considerable disadvantage: The movement of the robot during the picking tasks can result into a de-calibration of the cameras, which has the effect that the camera views do not align well anymore with the real environment. This might lead to subsequent errors in other modules like motion planning, picking and placing. Errors due to calibration are generally hard to recognize during the runtime as it might manifest in various seemingly unrelated failures like misplaced object detection, collision during arm movement or failed motion planning.

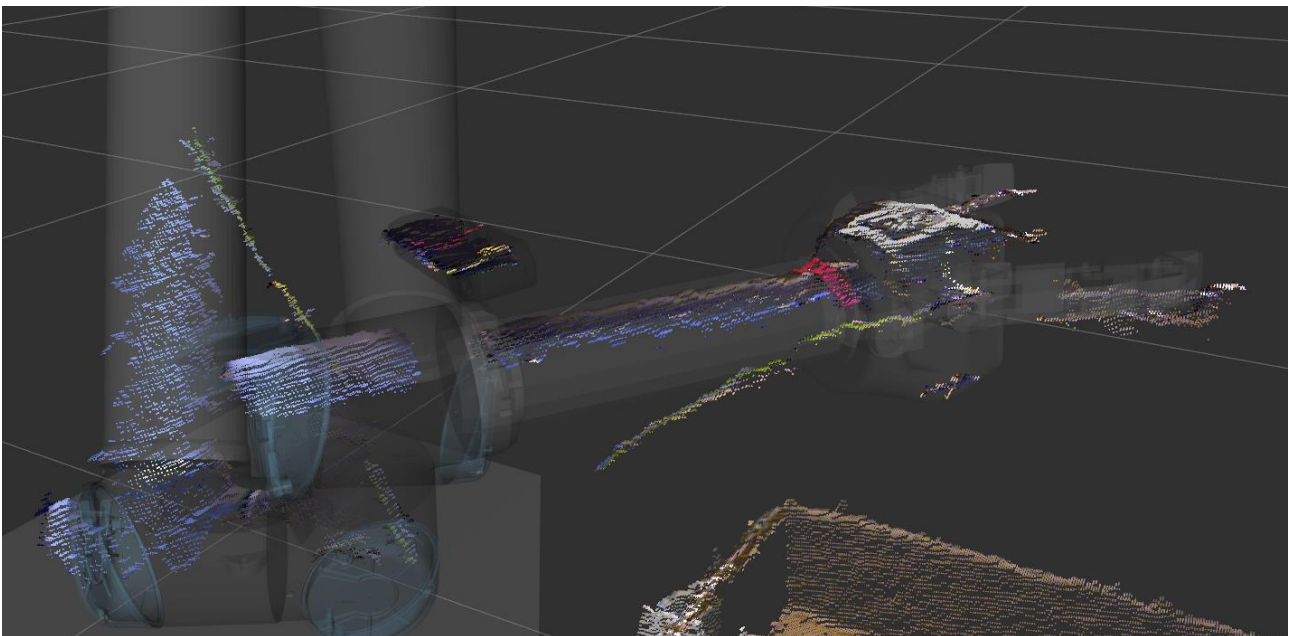


Figure 5 shows the manual calibration step where the point cloud of the robot arm, as captured from the RGBD camera, has to be aligned with the physical robot arm.

3.3.2 Automatic Camera Calibration

AAU started working on an automatic, easier-to-use camera calibration. Most of the openly available camera calibration systems include a checkerboard and the manual movement of the arm to pre-defined calibration points. Furthermore, an accurate measurement between a calibration marker and the end-effector has to be conducted, which is neither simple nor accurate, as the points of measurement lie within the robot arm. The aim of the novel approach is to calibrate the cameras *without* dedicated markers and in form of an *online* procedure so that the system can remain calibrated all the times. In its present version, however, markers are still needed.

Two different systems have been implemented: one for the wrist camera calibration and another one for the static workspace cameras. The calibration of the camera attached to the robot arm uses the classical approach of adjusting the view with respect to another calibration static camera using a checkerboard pattern: The user has to put the checkerboard in a position where it is visible from both cameras. A user interface then indicates when the checkerboard has been found and starts the calibration automatically. After a few seconds, it returns the estimated camera pose which is used to update the robot model. The calibration of the static cameras on the gantry does not need the use of a checkerboard and instead uses a fixed position marker on the gripper. The used Aruco marker is similar to a QR code and can be attached anywhere on the gripper (see Figure 6). The position of the marker with respect to the end-effector does not need to be known which improves usability considerably. The required theory has been developed by AAU and will be submitted at the upcoming ICRA conference.

Except for four arm poses that have to be programmed, the entire calibration procedure runs automatically. This procedure can easily be repeated anytime and takes less than 10 mins for one camera.

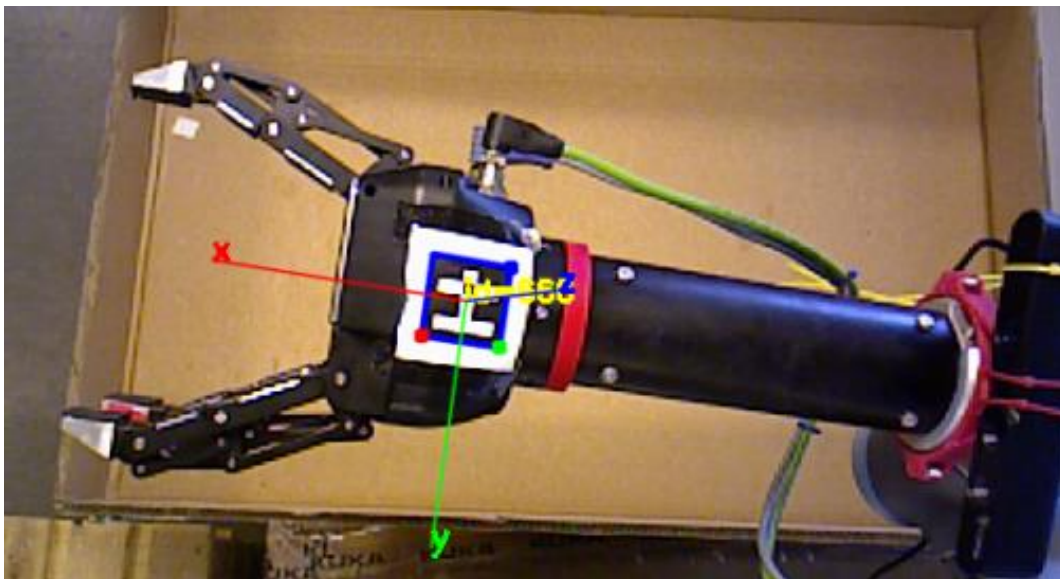


Figure 6 shows an Aruco maker on the gripper. The exact location of the maker on the gripper is unknown and calculated automatically during the calibration process.

3.4 Skill Primitives

A big effort since last time went into improving the skill-primitives. Skill-primitives are components the normal user usually does not have access to. They are used by the skill-programmer and are used as building blocks for building skills. The most important ones are discussed below. Their performance is evaluated in Sect. 7.

3.4.1 Skill-Primitive locate_bonn

This skill primitive is one of the perception components in the picking pipeline (see Sect. 3.1) and locates the horizontal support surfaces of pallets. In addition it segments the objects on top of this support surface, selects the object to grasp (object candidate being closest to the pallet center) and computes an observation pose to take a closer look at the object for recognition and localization.

In order to find potential object candidates, we then select the most dominant support plane, compute both convex hull and minimum area bounding box, and select all RGB-D measurements lying within these polygons and above the extracted support plane. We slightly shrink the limiting polygons in order to neglect measurements caused by the exterior walls of the pallet. The selected points are clustered (to obtain object candidates), and the cluster being closest to the center of the pallet is selected to be approached first. After approaching the selected object candidate with the end effector, the same procedure is repeated with the wrist camera in order to separate potential objects from the support surface. Using the centroid of the extracted cluster as well as the main axes (as derived from principal component analysis), we obtain a rough initial guess of the object pose. With the subsequent registration stage, it does not matter when objects are not well segmented (connected in a single cluster) or when the initial pose estimate is inaccurate.

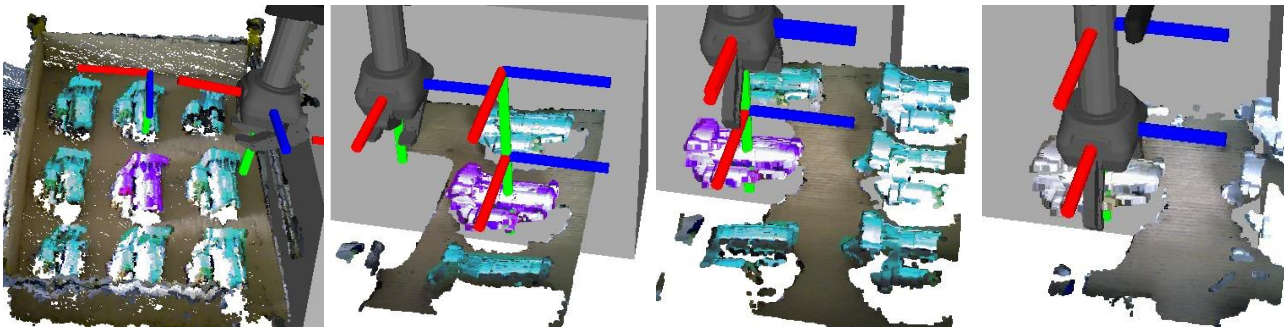


Figure 7 shows typical results of detection and localization. From left to right: workspace camera point cloud with extracted object candidates (cyan) and selected object (magenta), and wrist camera point clouds during localization, approach and grasping.

Details are provided in ¹

3.4.2 Skill-Primitive registration

The skill-primitive registration is used to accurately localize the part and to verify whether the found object is of the correct type. The initial part detection only provides a rough estimate of the position of the object candidate. In order to accurately determine both position and orientation of the part, we apply a *dense registration* of the extracted object cluster against a pre-trained model of the part:

In a training phase, we collect several views on the object. To optimize viewing pose selection we use a pose graph optimization techniques.

The pose refinement approach is then based on a soft-assignment surfel (“Surface Element”) registration: Instead of considering each point individually, we map the RGB-D image acquired by

¹ Holz, Dirk and Topalidou-Kyniazopoulou, Angeliki and Stückler, Jörg and Behnke, Sven
Real-time object detection, localization and verification for fast robotic depalletizing. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp.1459--1466, 2015.

the wrist camera into an MRSMAP and match surfels². This needs several orders of magnitudes less map elements for registration. Optimization of the surfel matches (and the underlying joint data-likelihood) yields the rigid 6 degree-of-freedom (DoF) transformation from scene to model, i.e., the pose of the object in the coordinate frame of the camera. The resulting segment observation likelihood is compared with a baseline likelihood of observing the model MRSMAP² by itself. We determine a detection confidence from the re-scaled ratio of both log likelihoods thresholded between 0 and 1.

A detailed discussion of this Skill-Primitive can be found in²

3.4.3 Skill-Primitive arm_motion

The skill-primitive arm_motion exploits many capabilities of MoveIt software. In order to achieve motions that are able to successfully place an object in compartments which in many cases are very confined, we have developed a planning pipeline by introducing two deterministic features which could be anticipated as planning reflexes. The need for that addition arises due to the stochasticity of the OMPL planners which makes them unstable as presented in preliminary benchmarking tests³. The Probabilistic Road-maps (PRM), Expansive-Spaces Tree (EST) and Rapidly exploring Random Tree (RRT) algorithms were evaluated. According to our results PRM performs better on the kitting task. However, its success rate is not desirable for industrial applications. Another deterrent factor on motion planning is the Inverse Kinematics (IK) solutions that derive from MoveIt. Although there exist multiple IK solutions for a given pose, MoveIt functions provide only one which is not always the optimal i.e. the closest to the initial joint configuration. We deal with this problem by sampling multiple solutions from the IK solver. The IK solution whose joint configuration is closer to the starting joint configuration of the trajectory is used for planning. The developed planning pipeline achieves repeatable and precise planning by introducing two *planning reflexes*, the *joint space linear interpolations* and the *operational space linear interpolations*. The first ensures that the robot's joints will rotate as less as possible which can be interpreted as an energy minimization problem. We solve this using linearly interpolating in the joint space of the robot between the starting and the final configurations. The latter results into a linear motion of the end-effector in its operational space. This is achieved by performing a linear interpolation between the starting and final poses of the end-effector. Furthermore, the spherical linear interpolation (slerp) is used for interpolating between orientations. This linear motion is desirable for going into the narrow compartments of the kitting box. The path that is created from the two reflexes is evaluated for (1) collisions, (2) constraint violations and (3) singularities. If any of those happen the pipeline employs the more sophisticated PRM algorithm for solving the planning problem.

3.4.4 Skill-Primitive kittingbox_registration

The kittingbox_registration skill-primitive is responsible for locating the kitting. The kitting box registration is generally used as part of any skill, that needs to physically interact with the kitting

² Stückler, Jörg and Behnke, Sven, Multi-resolution surfel maps for efficient dense 3D modeling and tracking, Journal of Visual Communication and Image Representation Vol. 25 (1), pp.137--147, 2014.

³ Polydoros, Athanasios S and Großmann, Bjarne and Roviada, Francesco and Nalpantidis, Lazaros and Krüger, Volker Accurate and Versatile Automation of Industrial Kitting Operations with SkiROS Conference Towards Autonomous Robotic Systems (TAROS), pp 255--268, 2016.

box, e.g. the placing skill. For the pose estimation of the box, an additional workspace camera with a top view on the kitting area is used. The kitting box detection is challenging because its appearance changes drastically when objects are inserted. This would require to distinguish between the box itself and the objects that were inserted, which is a challenge on its own. Instead, a classic engineering-based solution leads to a highly robust kitting-box detection: The kitting box registration is done based on a 2D edge detection on a cleaned and denoised depth image. The feature-edges are the upper-rims of the box (see Figure 5, bottom right) as these features hardly change due to objects in the box. A pixel-wise temporal voting scheme is used to generate from the point cloud the potential points belonging to the kitting box upper-rim. After mapping these back to 3D space, a standard Iterated Closed Point (ICP) algorithm is used to find the kitting box pose (see Figure 5).

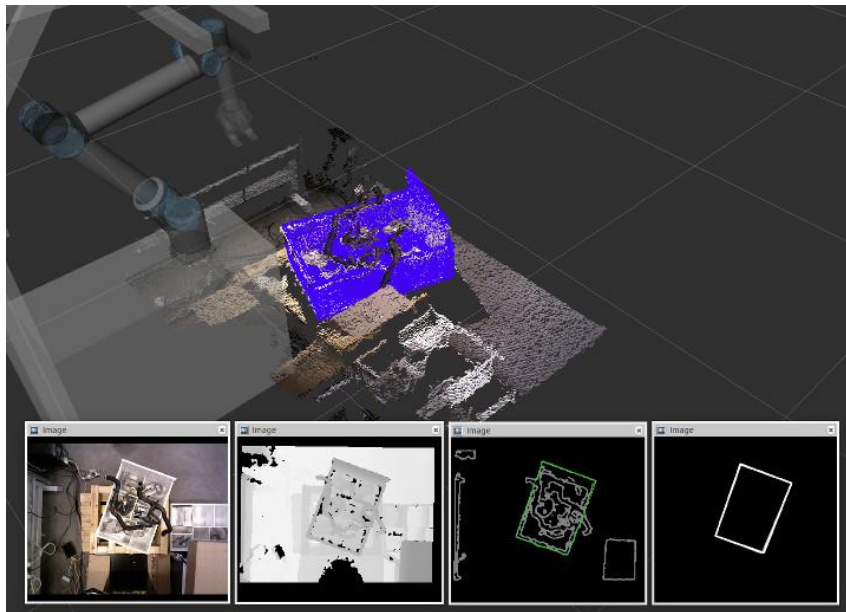


Figure 8 shows the the CAD model of the kitting box aligned with the visual data from the camera. The bottom images show some of the processing steps, incl. a 2D segmentation.

3.4.5 Learning Primitives `grasping_pose_learn`, `placing_pose_learn`, `object_train`

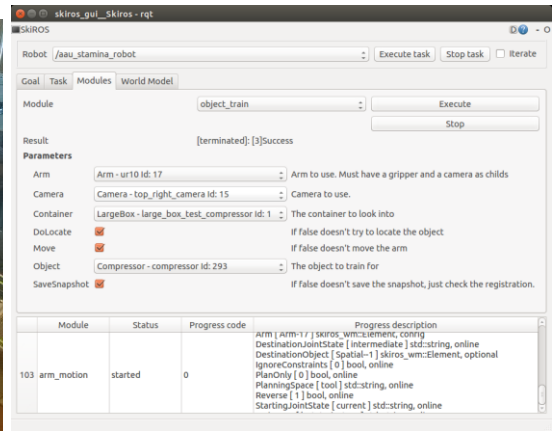
Learning primitives are needed to extend the robot's knowledge base with the important information about the environment, in an initial configuration phase. For every single object the robot has to learn the information that is relevant for physically interacting with it for the kitting task:

1. the object shape
2. the grasping pose
3. the placing pose

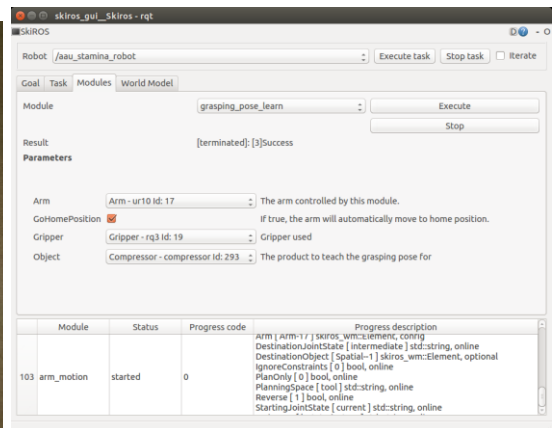
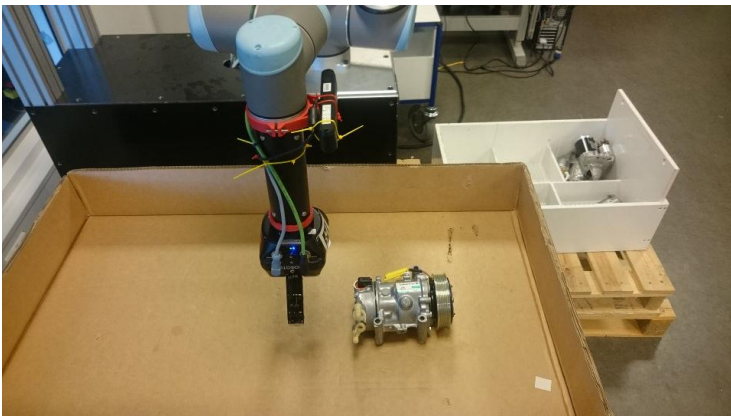
These functions can be executed *before* arriving to the test site.

The teaching phase is partially automated and consist in the following procedure:

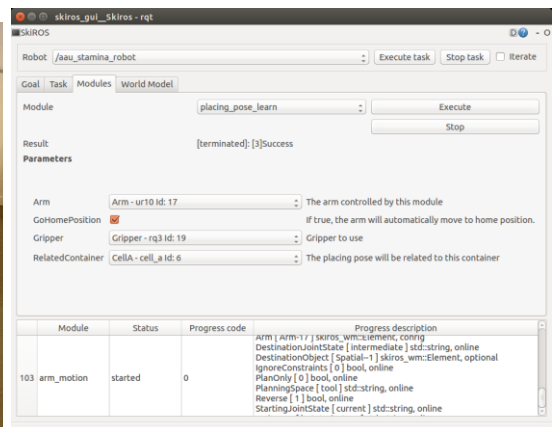
1. Place an object on the container, and run the object train primitive specifying the object's type. The primitive locate the object, move the arm and register the snapshot using the wrist camera.



2. Move manually the robot arm to the object and run the grasping pose learn module to register the grasping pose



3. Bring the arm back to home position and pick the object using the pick skill
4. Move manually the robot arm in a kit compartment and run the placing pose learn module



The execution of all four steps takes in average 4 minutes and has to be done once for each single object.



Figure 9 shows the training of an object at the test site.

4 SkiROS and Skills

In this section, we briefly discuss

- the progress in SkiROS since the last test sprint.
- the preparation needed *before* arrival at the test site
- the preparation needed *at* the test site.

The SkiROS system (“Skills for ROS”) is the software platform that provides the infrastructure for running the skills and for controlling the hardware. Deliverables D3.1 and D3.2 summarize the SkiROS system. A complete account of the SkiROS framework has been submitted as a book chapter to Springer: *Robot Operating System (ROS) The Complete Reference (Volume 2)*⁴.

4.1 Progress since last time

As outlined in D4.2.1 the kitting technician is the worker that will have to interact with the robot on the shop floor. Thus, the most important extensions of the SkiROS system are focused on improved human-robot interaction for the kitting technician:

1. Addition of skill primitives for fast teaching of new objects, incl. how to grasp and how to place them (see Sect. 3.4.5).

⁴ http://events.coins-lab.org/springer_ros_book_v2.html

2. Development of an integrated task planner on the robot: During the 2nd test sprint, the robot was receiving pre-planned sequences of skills. Now, the robot receives goals, and plans by itself the correct sequence of skills. In case of a pre- or post-condition failures, the robot is able to re-plan autonomously in order to be able to continue the mission.
3. Development of a GUI for an easier overview and run-time monitoring of the system. The GUI present several tabs to
 - a. edit the world model: This feature is needed for the Kitting-technician to correct the world model knowledge if needed, e.g. upon an undetected error such as an object falling out of the gripper during picking (see Sect.7.2.1).
 - b. allow single module testing: Of use for the kitting technician in case of an error with a skill.
 - c. task monitoring and run-time control: used by the kitting technician for run-time status control
 - d. goal assignment: use by the kitting technician to manually insert a task goal.
4. Tighter integration between the robot and the logistic planner world model: The robot now updates constantly to the logistic planner with low level information, such as its position in the factory. The logistic planner can also query the robot to get other recent updates. This allows the kitting technician to have access to the run-time status of potentially several robots even from a central location.

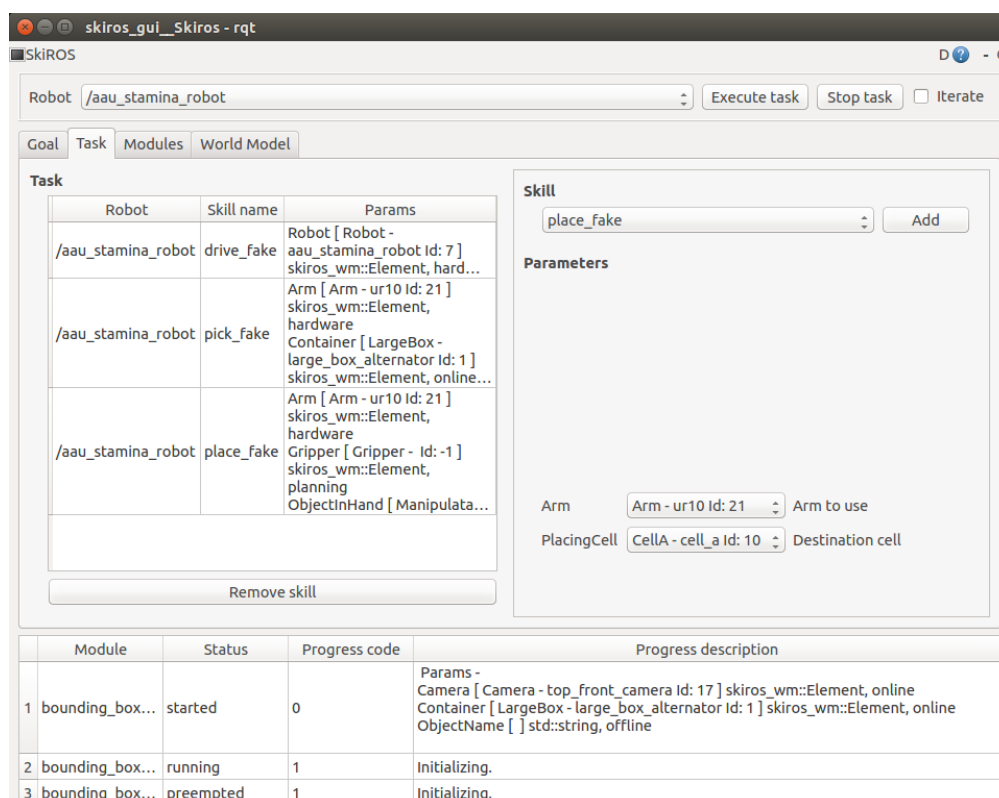


Figure 10 show the SkiROS GUI with a simple drive-pick-place task. On the top there are buttons to command execution and stop. On the bottom it is possible to monitor the progress.

4.2 Preparation of SkiROS for use

The following two sub-sections will discuss what preparation steps have to be executed to prepare SkiROS for use. Here, we distinguish between two cases:

1. Tasks that have to be done on-site but only once, such as **functional testing of system components** that might show site dependent behavior. This type of testing is needed in order to verify the components within the real environment, i.e., those components that were so far only tested in the lab-space. Once the system is perfectly matured, these tests would not be needed anymore. However, given the experience with robotic systems under changing environments, it might be worthwhile to consider a suite of unit-tests that can be run in each new environment.
2. Tasks that have to be done on-site every time the system is started up within a new environment. This includes **system configuration, training maps and potentially objects**. This task has to be executed by the kitting technician when setting up the STAMINA system, e.g., at a new site.

4.2.1 On-site functional testing

In preparation for 3rd test sprint AAU focused on:

- Add feature to pick from left and right sides
- Setting-up and testing the wireless to connect the robot with the logistic planner, the latter running on a stationary workstation
- Alignment of the robot ontology with the logistic planner: This consisted in extending the robot ontology with the names of parts that were newly introduced into the logistic planner. This is a task that could have been done off-site but was done on-site this time due to last minute changes.
- Test interfaces between the robot and the logistic planner (SOA-interfaces). This in particular consisted in solving some minor issues:
 - Several wrong feedback codes from the robot to the logistic planner were corrected
 - A minor set of parameters in the robot launch file were re-configured to have the robot ready to receive goals from the logistic planner
- Test and refine the integrated task planner. In particular:
 - It was noted that when the robot arm is not at home position, a safety rule must prevent the platform from navigating. This safety rule required some minor modifications to the pre-conditions of the drive skill.

4.2.2 STAMINA system configuration

The following tasks have to be executed by the kitting technician when setting up the STAMINA system at a new site.

- Teaching – for every object the sequent information has to be learned from the robot:
 1. the object shape
 2. the grasping pose
 3. the placing pose

The teaching phase is partially automated and has been discussed in Sect. 3.4.5.

- Teach the 3D environment map and draw the rail where the robot should navigate. This procedure was already discussed in D1.3.2 and it has not changed.

5 Configuring the Logistic Planner

In this section we discuss all topics related to the vertical integration of the robot, functionality implemented by the Logistic Planner (see deliverables D4.1.1, D4.1.2, D4.2.1, D4.2.2 and D4.2.3). This section is an incremental update from the earlier deliverable D1.3.2. It includes the connection with the SkiROS framework running on the robot, the connection with the Mission Planner, the creation and management of the logistic kitting zone's model (logistic world model) and the connection with the MES of PSA. All these elements were put into use and tested. The reader will find throughout this section screenshots of the interface of the logistics planner for comprehension purposes.

5.1 *Mapping the kitting zone and constructing the logistic world model*

The model of the kitting zone is again achieved in two phases. In the first phase, the physical objects identified in Deliverable D.1.1.3 (parts, small boxes and large boxes pallets) were categorized and characterized in terms of code references, names and physical structure. Data about racks were gathered from separate documents supplied by PSA. However, some of the physical objects available in the kitting area didn't correspond to these previously modelled object types. In these cases, new types of racks and large boxes were created within the Logistic Planner.

For each object type (parts, small boxes, large boxes, racks and conveyors), the following data were introduced in the Logistic Planner:

- reference (system identifier) and name (textual friendly name);
- width, depth and height of the external volume.

For racks and large boxes, their internal organization was identified in terms of levels and cells within each level: a rack is organized in levels and each level can have several cells containing instances of a given type of small box; a large box is also organized in levels and each level organized in cells containing usually instances of a given part type). This modelling is consistent with the ontology identified in the previous Section.

The outcome of this modelling can be seen in the Logistic Planner, in its "Type of objects" area (see the next following Figures).

LogPlanner / Type of Objects / Small boxes Alerts: 0 / Access to ROS: OK

+ add type of small box

Smallbox id	Width	Depth	Height	Actions
06432	396	594	314	
04322	297	396	214	
06422	396	594	214	
BNA20	400	600	200	
ANC20	300	400	200	

Figure 11 shows the different types of objects: Small boxes.

LogPlanner / Type of Objects / Parts Alerts: 0 / Access to ROS: OK

Part id	Width	Depth	Height	Actions
Engine pipe 9684566480	0	0	0	
Engine pipe 9686423080	0	0	0	
Engine pipe 9673633180	0	0	0	
Engine support 9672950980	0	0	0	
Engine support 9657457980	0	0	0	
Engine support 9674030280	0	0	0	
Engine support 9684613880	0	0	0	
Engine support 9675508280	0	0	0	
Engine support 9682776880	0	0	0	
Engine support V758078180	0	0	0	
Thermal shield 9804088180	0	0	0	
Thermal shield V759560680	0	0	0	
Thermal shield V759560580	0	0	0	
Alternator 9803750980	0	0	0	
Alternator 9666997980	0	0	0	

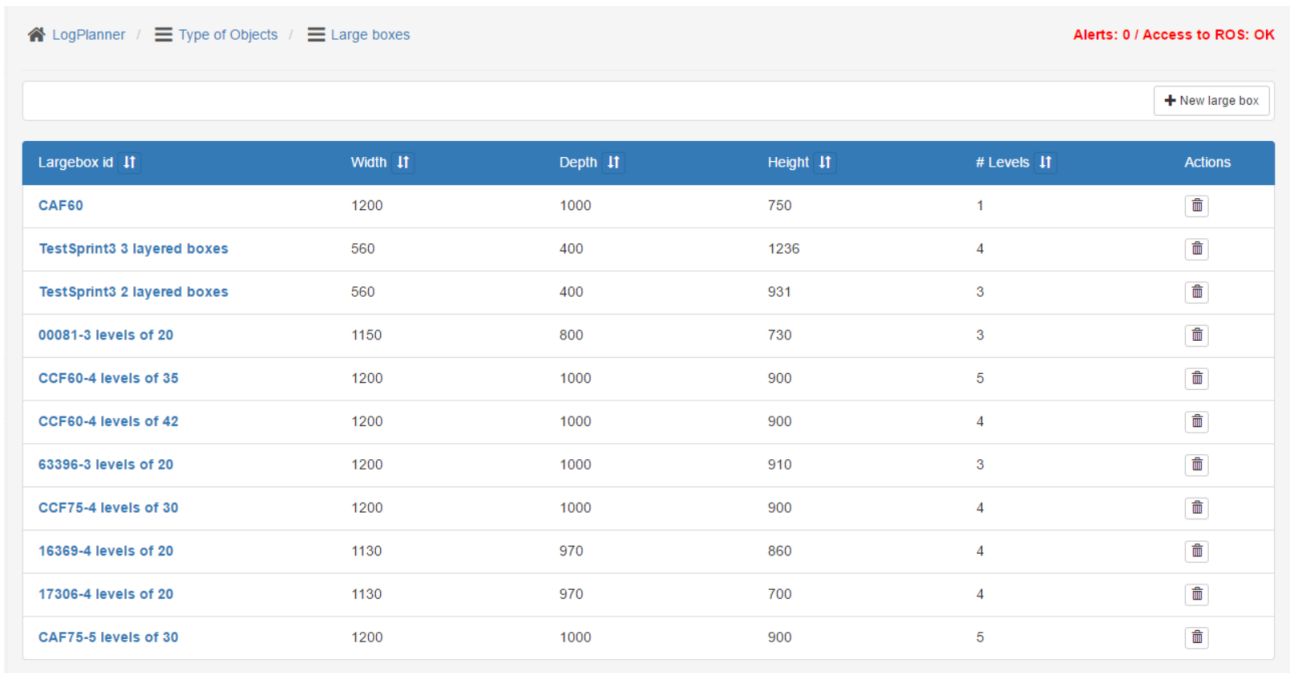
Figure 12 shows the different types of objects: parts.

LogPlanner / Type of Objects / Racks Alerts: 0 / Access to ROS: OK

+ New rack

Rack	Width	Depth	Height	# Levels	Actions
TestSprint 2130X530	530	2130	1480	1	
TestSprint 2870X1225	1225	2870	1425	3	
TestSprint 2870X1410	1410	2870	1425	3	
TestSprint3 1210 x 2800	1210	2800	1940	4	
TestSprint3 1410 x 2800	1410	2800	1940	4	

Figure 13 shows the different types of objects: Racks.



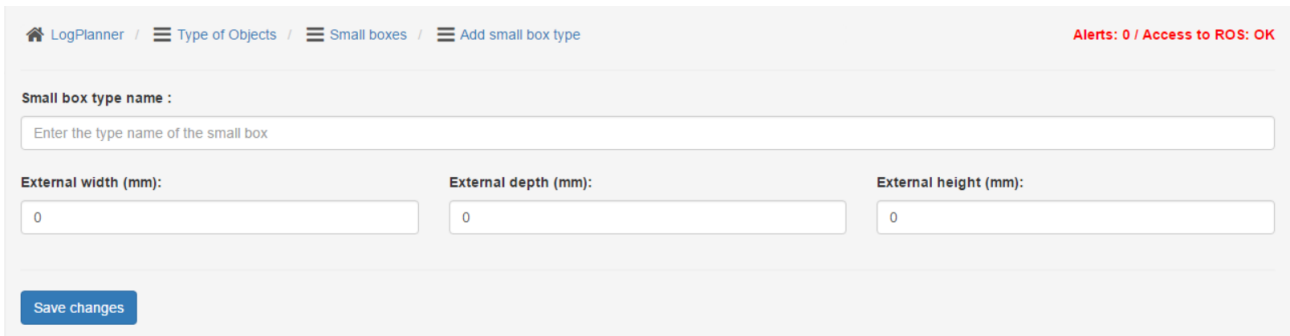
LogPlanner / Type of Objects / Large boxes Alerts: 0 / Access to ROS: OK

+ New large box

Largebox id ↑↓	Width ↑↓	Depth ↑↓	Height ↑↓	# Levels ↑↓	Actions
CAF60	1200	1000	750	1	
TestSprint3 3 layered boxes	560	400	1236	4	
TestSprint3 2 layered boxes	560	400	931	3	
00081-3 levels of 20	1150	800	730	3	
CCF60-4 levels of 35	1200	1000	900	5	
CCF60-4 levels of 42	1200	1000	900	4	
63396-3 levels of 20	1200	1000	910	3	
CCF75-4 levels of 30	1200	1000	900	4	
16369-4 levels of 20	1130	970	860	4	
17306-4 levels of 20	1130	970	700	4	
CAF75-5 levels of 30	1200	1000	900	5	

Figure 14 shows the different types of objects: Large boxes.

The modelling of new type of objects (small boxes, large boxes and racks) in the kitting zone is achieved by the screens presented in the following Figures.



LogPlanner / Type of Objects / Small boxes / Add small box type Alerts: 0 / Access to ROS: OK

Small box type name :

Enter the type name of the small box

External width (mm): External depth (mm): External height (mm):

0 0 0

Save changes

Figure 15: Type of Objects: Creation of a new type of small box.

LogPlanner / Type of Objects / Racks / Add rack type Alerts: 0 / Access to ROS: OK

Step 1 Step 2

rack type name : # Functional levels: # Cells per level:

External width (mm): Level width (mm):

External depth (mm): Level thickness (mm):

External height (mm):

Front view Side view

external depth

horizontal offset

vertical offset

Figure 16: Type of Objects: Creation of a new type of rack.

LogPlanner / Type of Objects / Large boxes / Add large box type Alerts: 0 / Access to ROS: OK

Largebox type name :

Level height (mm): # Levels: # Rows: # Columns:

External width (mm): Internal width (mm):

External depth (mm): Internal depth (mm):

External height (mm): Internal height (mm):

Front view Top view

external width

internal width

Columns

Level height

Internal height # Levels

External height

Save changes

Figure 17: Type of Objects: Creation of a new type of large box.

In the second phase, the goal was to identify which objects actually exist in the kitting area (instances of the types known in the Logistic Planner), which type of containment relationship relate some of the objects and where they are located in space (i.e., where the objects are implanted in space). This modelling work was done by the user within the Logistic Planner in the following way:

1. A bitmap image of the kitting area was provided to the Logistic Planner⁵. This image was produced by the robot while navigating through the kitting area (Section 3.1) and identifies visually the things that were detected in the area. Retrieval of this bitmap image is automatic

⁵ In the test sprint, a map of the kitting area is generated by the robot after the mapping phase (see Section 3.1). However, the methodology supported by the Logistic Planner also supports the usage of an image generated from an Autocad image, specifying in 2D the location of each physical object.

and provided by the robot itself by pressing “Get new Map” button (see following Figure). This image also specifies the reference point of the map, signalled as a red cross in the Logistic Planner’s representation (lower left corner in the Figure). The coordinate system of the robot and of the Logistic Planner is assumed to be the same (x increases to the right, y increases to the top). As showed below, the location of racks and large boxes is easily deduced and these areas allow the user to subsequently position instances of racks and large boxes. However, before going in that direction, a horizontal line in the robot map is specified by selecting two points, so that this line corresponds to a horizontal line (this must be done according to the user’s perception of the reality).



Figure 18: Implantation: Set Map.

2. Having the robot map horizontally aligned in the screen, the user specifies the area in it that corresponds to the kitting area (i.e., the root region). This is shown in the next Figure. The darker area is the root region.

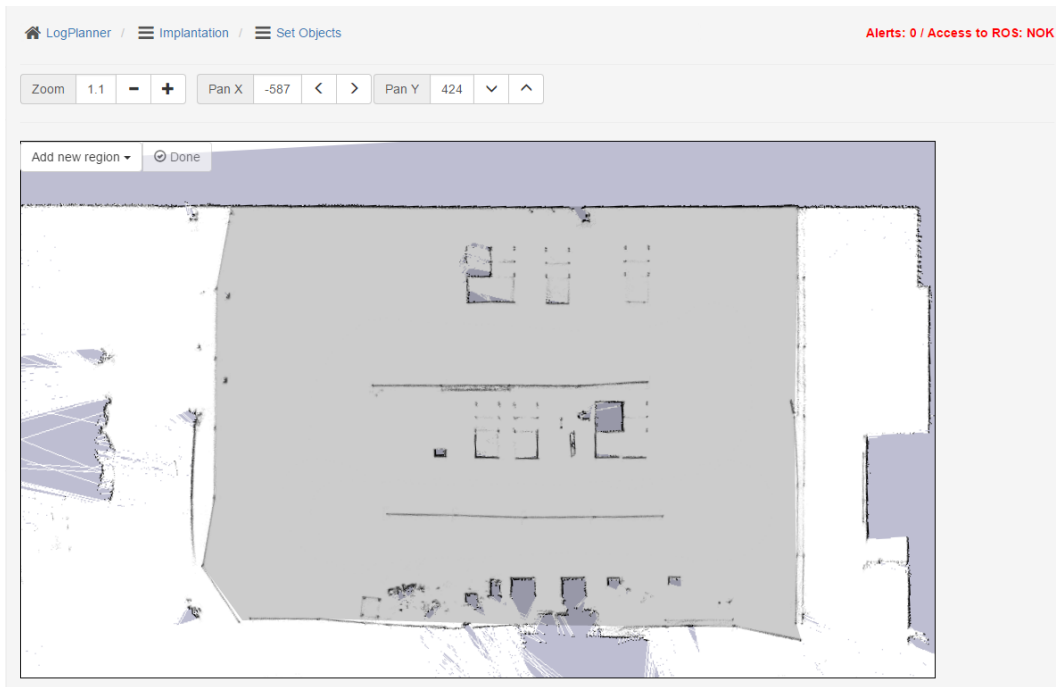


Figure 19: Implantation: Drawing of the kitting area (root region).

3. Taking the root region as reference, a container region is defined on top of the map (a container region is a region within the root region that contains physical objects). This is shown in the next Figure.

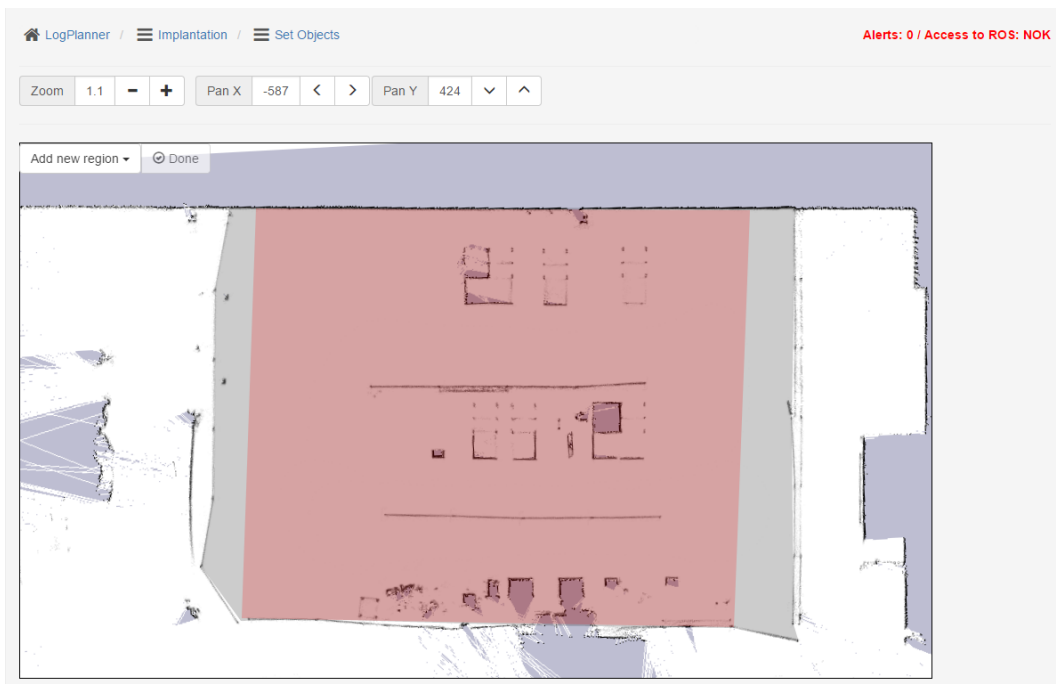


Figure 20: Implantation: Drawing of the container region.

4. An iterative process now begins where the user, according to the reality, instantiates objects representing the ones available in the kitting area (i.e., racks and large boxes) and locates

them in the proper place in the logistic map. The next figure shows the implantation of a rack (black rectangle with red arrows pointing the front of the object).

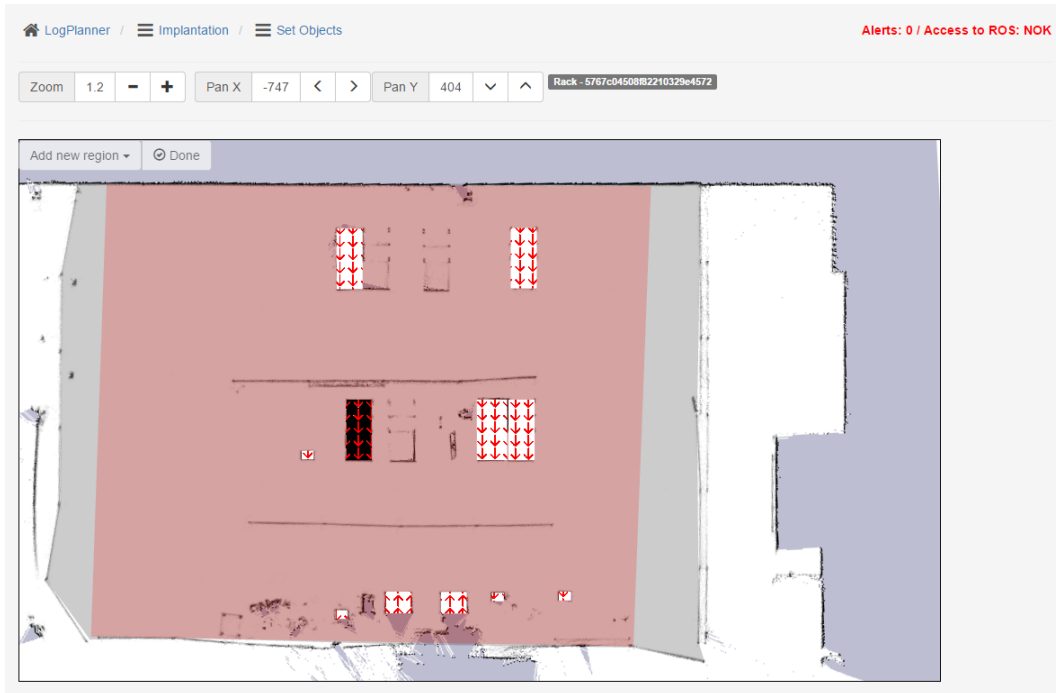


Figure 21: Implantation: Insertion of racks and large boxes.

The final outcome of this process with all boxes and racks marked is displayed in the following figure.

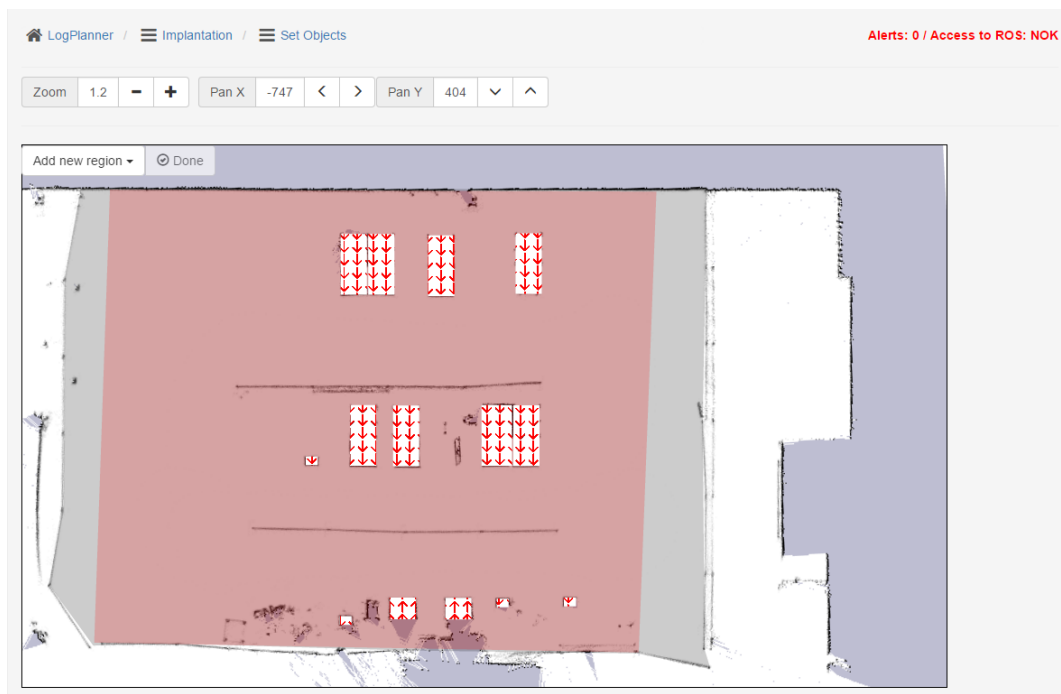


Figure 22: Implantation: identification of racks and large boxes in the logistic map.

5. The final step in the implantation process, allows the user to specify the following containment relationships:
 - which small boxes and parts are contained by which racks (next following Figure);

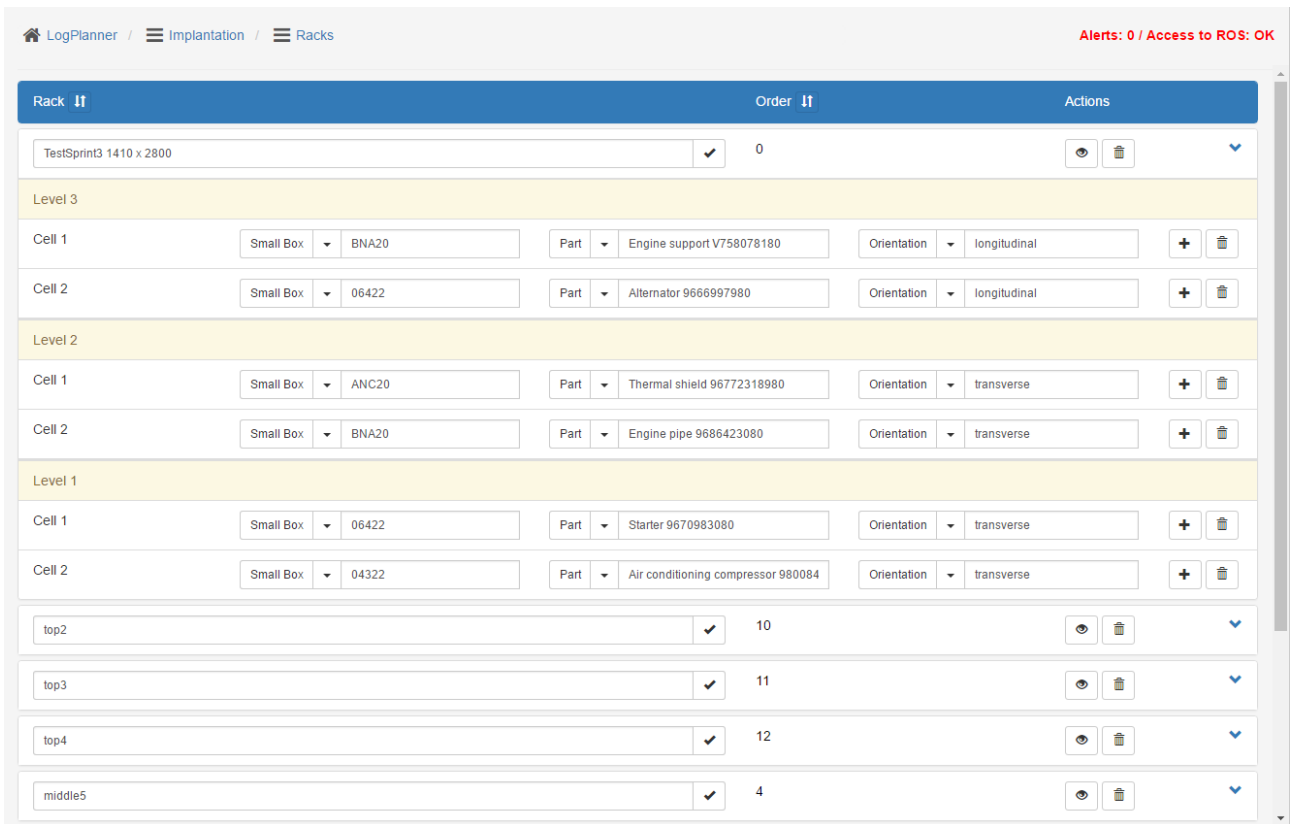


Figure 23: Association of small boxes and parts to a rack cell.

- which parts are contained by which small boxes (next Figure) if one wishes to change the previous association;

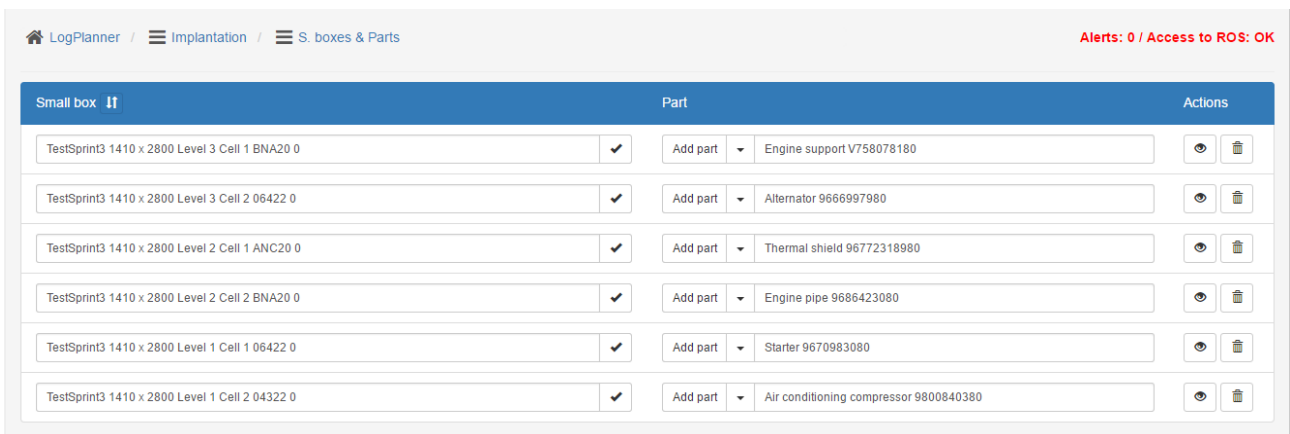


Figure 24: Association of parts to small boxes.

- and which parts are contained by which large boxes (next Figure);

LogPlanner / Implantation / L. boxes & Parts Alerts: 0 / Access to ROS: OK











Large box	# Levels	Part	Order	Actions
bottom1	3	Add part Thermal shield V75956068002	0	 
bottom2	3	Add part Thermal shield V75956068002	1	 
bottom3	4	Add part Thermal shield V75956068002	2	 
bottom4	3	Add part Thermal shield V75956068002	3	 
middle1	4	Add part Thermal shield V75956068002	8	 

Figure 25: Association of parts to large boxes.

The following figure shows a 3D view of the modelling done within the Logistic Planner. Eight racks (green colour) and six large boxes (orange colour) comprise the main objects. A small box containing parts was inserted on the each level of all the racks.



Figure 26: 3D view of the logistic world model.

Changing the map (i.e. retrieving a new instance from the robot) obliges the User to repeat all the modelling steps described above.

The outcome of this modelling in the Logistic Planner is the Logistic World Model that is provided to the Mission Planner and task Manager whenever required.

5.2 Teaching the docking positions

Teaching of the docking locations was part of the procedure for mapping the kitting zone into the map that the robot used for navigation. However, these locations are not used by the Logistic Planner: for each rack, large box and conveyor (furniture object), the Logistic Planner defines automatically a docking station as the space occupied by the robot in front of the furniture object, distanced by a security space previously configured in the Logistic Planner software. See the related discussion in Sect. 3.1.3.1.

Following Figure shows the docking stations (red rectangles) in a 3D view of the logistic world model.

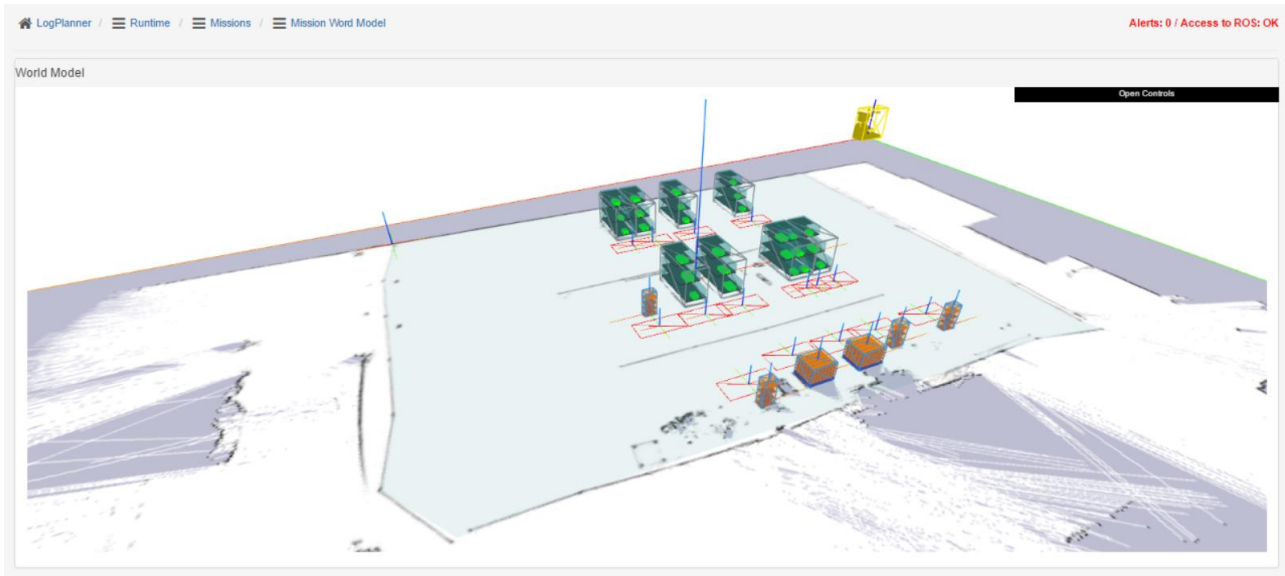


Figure 29: 3D view of the logistic world model identifying in red the docking stations.

5.3 Retrieving the skills from the robot

The following Figure shows the skills that are dynamically retrieved from the robot. The two buttons on the right, under the Actions label, allow the skills to be retrieved from the robot and to be sent to the Mission Planner.

Robot	P Station	Is Alive	Location (X,Y,Z)	Orientation (R,P,Y)	Actions
stamina_mobile_base	Yes	false	(0.0, 0.0, 0.0)	(0.00, 0.00, 0.00)	[Buttons]

Figure 29: Skills implemented by robot

5.4 Creating the kitting order

A kitting order is an instruction telling the system which kit is to be built and which parts are to be placed on which cells within the kit. It also specifies the car that will receive the kit. As such, a kitting order has the following data items:

- kittingOrderId - system identifier and friendly name of the order;
- carSeqNumber – the sequence number of the car that will get the kit after its completion;

- kittingZoneld - system identifier and friendly name of the kitting area (kitting zone);
- kitTypeeld - system identifier and friendly name of the kit to be built;
- partsToPick – a list identifying the parts that should be placed in the kit. Each element in the list has the following data items:
 - partId - system identifier and friendly name of the part;
 - cellId - system identifier and friendly name of the cell within the kit to contain the given part;
 - quantity – number of parts to place in the cell.

Several kitting orders were manually defined targeting a range of different situations. As examples, the following are mentioned:

- Construction of a kit containing one part. The part is to be picked from a large box.
- Construction of a kit containing two parts. The parts are to be picked from two different large boxes.
- Construction of a kit containing three parts. The parts are to be picked from three different large boxes.

Each kitting order is contained by a computer file and is imported by the Logistic Planner through its user interface (see the following Figure).

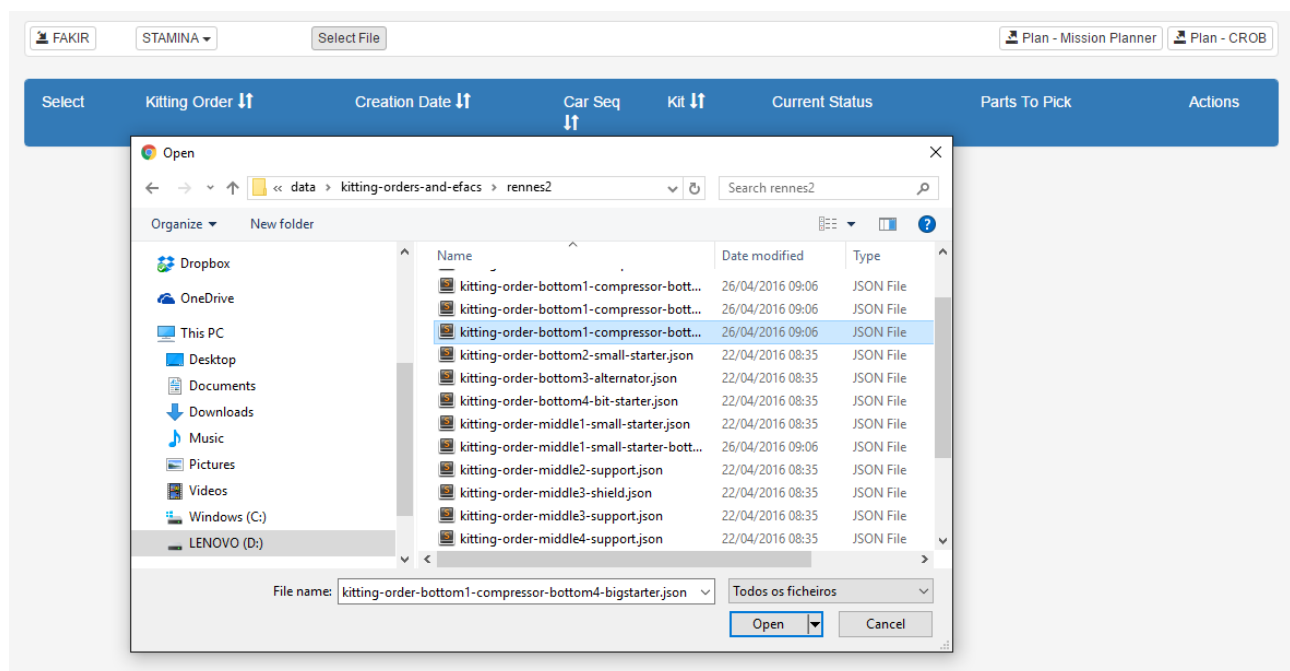


Figure. 30: Importing a kitting order into the Logistic Planner.

Additionally, the Logistic Planner allows the User to import eFAC from files or from PSA MES. The first option is done in the same way as described above but while selecting kitting order files one must select an eFAC file. For the online retrieval of eFAC from PSA MES, a separate button allows the user to call a web service implemented on the MES and to automatically import eFAC documents ready to be planned by the Logistic Planner.

6 Mission Planner Integration

This Section describes the integration of the Mission Planner (WP4) in the third test sprint. In the second test sprint, the Mission Planner was used to generate the full sequences of skills for the robots in place of the Task Planner. The full skill sequences were then sent with the mission to be performed by the robot. For the third test sprint the Task Planner was fully implemented and successfully integrated in the STAMINA system. This means that the Mission Planner was back to its primary role of generating missions as high level goal sets instead of skill sequences.

The mission planner communicates with the Logistics Planner using the PlanKittingOrders and ParkRobots services. The former is called by the Logistics Planner when a new set of kitting orders is received from the MES and includes the kitting orders, world model and robot skills information. For the third test sprint this was updated so that the returned missions contained only the goals for the robot. The latter, the ParkRobots service, was newly added for the third test sprint so that the Logistic Planner could include a way for the operator to send the robots back to their charging locations at the end of a shift, or as required. This was tested and shown to be working with the goals being of the form (RobotAtLocation robot_name robot_park_location).

As the test sprints are run with only a single robot, no multiagent planning is required by the Mission Planner for the purpose of testing with the STAMINA robot in the factory. As the Mission Planner's job in the test sprints is limited (compared to the "blue sky" research part of WP4 which considers a fleet of robots in simulation), the majority of the contributions of UEDIN for this test sprint are for the Task Planner component of SkiROS that uses automated planning to find the correct sequence of skills to perform the current task.

7 Complete system test

7.1 Introduction

For the experiments the following test-sprint narrative (TSN) was defined:

The robot received a kitting order. It then starts its round to collect the requested parts. The entire process runs smoothly: The picking and placing works without failures, the navigation works smoothly and the robot does not get stuck. At the end of its round the robot delivered its kitting box to a pre-defined location. If the system discovers a failure, it is able to call for assistance.

The following Performance Indicators (PIs) were identified:

1. complete system functional
 - a. able to collect parts: *The system is able to perform a kitting order*

- b. doesn't break: *it is able to detect and handle failures through re-planning and human assistance.*
 - c. kit correctness / number of errors per kit: *The generated kits are error-free.*
- 2. reliability (with respect to different scenarios)
 - a. error recovery capability: autonomous operation, operator-assisted, crash
- 3. effectiveness of the robot and the logistic planner (incl. mission planner)
 - a. startup-time
 - b. setup-time of the system and of the relevant parts of the world model
 - c. execution time
 - d. re-configuration time
- 4. testing normal functional mode, systematic changing of test scenario
 - a. A container (LV) gets empty
 - b. A box (SB) gets empty
 - c. Wrong part in container
 - d. Human populated area
 - e. slightly moved boxes, test localization + docking behaviour
 - f. packaging material on the floor (material + box on the floor)

In the following sections we will first discuss the testing of the complete system (Sect 7.2). Then, we will discuss the sources of errors by investigating the skills separately and identify which skills (Sect. 7.2.1) and which skill-primitives (Sect. 7.2.2) generated errors. Finally, we will investigate to what extent the SkiROS system was able to recover from detected errors (Sect. 7.3).

7.2 Complete System Functional Testing

The complete system was tested in the realistic environment at PSA based on real kitting orders, see the supplied videos.

For systematic testing the kitting orders consisted of 3, 4 or 5 different parts. The parts were

3 parts	compressor, alternator, small starter
4 parts	Same as above, plus the large starter
5 parts	Same as above, plus the heat-shield

The test results for 39 test runs were recorded. The results are summarized in Table 1.

Mission type	Kitting Tasks	Success according to TSN	Detected (good) fails	Undetected (bad) fails	System Crashes	Unrecoverable Failures
--------------	---------------	--------------------------	-----------------------	------------------------	----------------	------------------------

5 parts	2	0	2	0	0	0 %
4 parts	22	7	11	4	0	18 %
3 parts	15	10	2	3	0	20 %

Table 1 shows the success rate of a set of experiments for different number of parts. Success is defined as the absence of any failure during the execution according to the TSN. In case of detected failures, the system called for human assistance. No system crashes were identified.

During the tests, no system crashes were detected.

“Undetected fails” are *bad* errors that cannot be detected by the system. These errors are bad because the robot assumes a success and is thus not able to notify the system of the error. Undetected errors can lead to undefined states in the system. An example for an undetected error is an object slipping out of the gripper. As the stamina system presently lacks a force torque sensor on the wrist, this incident cannot be detected (see Sect. 7.2.1). As “Unrecoverable Failures” Table 1 gives the percentage of these undetected failures.

“Detected fails” are *good* errors because the system is able to identify these as soon as they happen. Generally speaking, these *good* errors are skill pre- or post-condition check failures. A detected fail usually still allows the system to complete its mission. In most cases, the system will be able to recover from them by simple re-planning (see Sect. 7.3). In some cases, the system cannot re-plan and needs to call for help. These are the cases mentioned in Table 1: The system was not able to complete its mission without calling for human assistance.

In principle, the big challenge is to detect and identify errors. Only identified errors can be treated so that the system can recover from them to complete the mission. An error recovery can be achieved, e.g., through suitable error handling procedure. The present STAMINA system does not yet have suitable error handling procedures other than direct re-planning. Therefore, we treated in this section the need for human intervention as an error as they make the system ineffective.

The following should be pointed out:

- In Table 1 one can see that for 3 and 4 part kits, the amount of unrecoverable failures was 18-20%. The 0% of unrecoverable failures for the 5 kit parts is somewhat artificial because the detectable error was persistent: it was not possible with the present grasping method to pick the heat shield.
- In the rows of “4 parts” and “3 parts” of Table 1, the 18%, and respectively 20%, are averages over the entire set sequence. However, during these tests, as we became more and more familiar with the peculiarities of the system it became evident that most of the errors were predictable. Consequently, the last five test runs were, except for one undetected failure (object slipped out of the gripper), successfully completed in accordance with the TSN. As we will discuss below, the observed undetected failure can be handled using additional hardware.

Mission type	Kitting Tasks	Success according to TSN	Detected (<i>good</i>) fails	Undetected (<i>bad</i>) fails	System Crashes	Unrecoverable Failures
--------------	---------------	--------------------------	--------------------------------	---------------------------------	----------------	------------------------

4 parts	5	4	0	1	0	20 %
---------	---	---	---	---	---	------

In the following, we will have a closer look at the different failures that happened during execution.

7.2.1 Evaluations of the Skills within the complete system

In this section we will have a closer look at the various failures, i.e., the reasons for the system failures listed above.

Skill	Attempts	Success	Detected Fails	Undetected fails	System Crashes	Average time	Success
drive_freiburg	84	70	14	0	0	38,89	83 %
pick_aau	119	84	26	9	0	55,38	71 %
place_aau	81	72	9	0	0	34,61	89 %

Table 2 summarizes the success rate of the various skill executions.

Table 2 summarizes the success rates of the various skills. Again, we differentiate between success, detected fails and undetected fails. There were no crashes of the skill plugins detected.

Skill drive_freiburg: The drive skill has been greatly improved since the last test sprint, see earlier discussion. However, the robot has shown some strange driving manoeuvres when driving down the kitting zone. According to the partners from Freiburg this “drunk driving” navigation was due to a parameter setting that were fixed at the end of the test sprint and which will have to be tested at the next test sprint. The “drunk driving” was not a problem as such. However, when the robot approached a goal pallet with a “drunk driving” path, the robot easily ended up in a non-parallel pose to the pallet and the drive_freiburg skill was not able to execute corrective manoeuvres and filed a “detected fail” by calling for help. Using manual control of the platform, the robot was able to recover from this error.

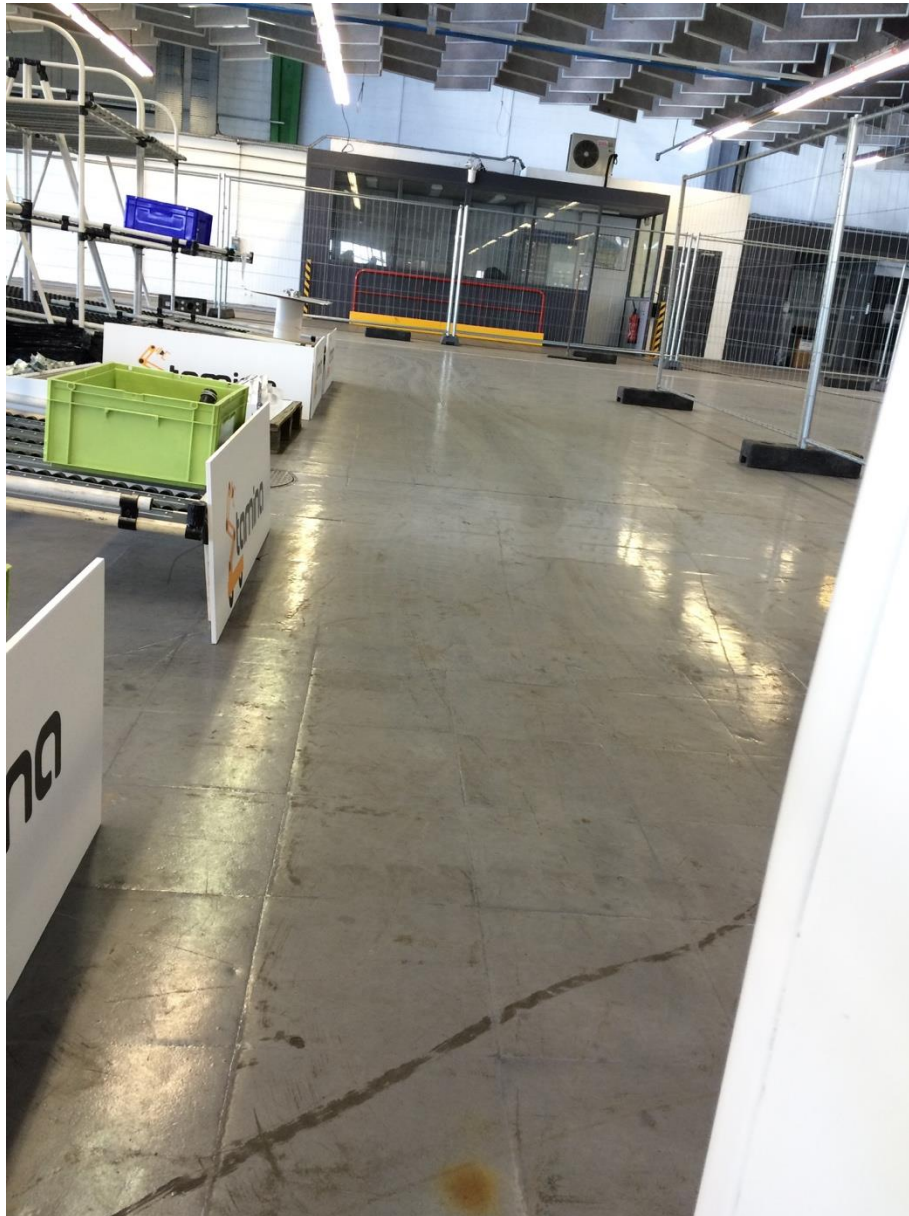


Figure 27 shows the tracks on the floor made by the drunk-driving movement of the robot. The tracks can be seen very faintly.

Skill pick_aau: The used pick skill was the pick_aau skill. This skill consisted of various skill primitives provided by UBO, incl. locate_bonn, registration, object_train, grasping_pose_learn (see Table 3). However, the skill pick_bonn suffered had continuous failures so that it was replaced by the alternative pick skill pick_aau. pick_aau is defined as a finite-state-machine-like processing pipeline built by AAU that controls the execution of the various relevant skill_primitives (see Table 3).

The success rate of the skill pick_aau was 71%. However, including the detected (*good*) failures, the rate of un-recoverable errors was only 7.6 %. Unrecoverable errors were due to, e.g., objects fallout out of the gripper (see Figure 28).

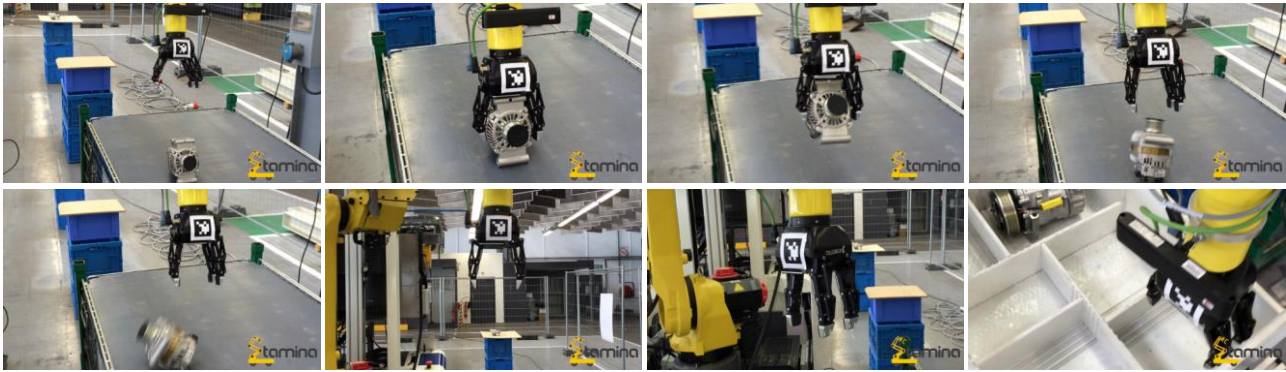


Figure 28 shows a typical example when an object falls out of the gripper. The robot is not able to detect this incident. It completes the pick&place cycle and assumes a successful completion.

Another often undetected failure is due to a bad *registration*. *Registration* is the process of computing the pose of an object based on the visual information from a 3D camera. The object pose is needed, e.g., for correctly picking an object.

The registration is realized as a low-level function (skill-primitive) of the SkiROS-system. An undetected fail of the registration-primitive can in some cases lead to an undetected fail of the picking skill. An example for this case can be seen in Figure 29: The object pose gets registered wrongly, and the robot arm attempts to pick the object from a wrong direction.



Figure 29 shows the result of a bad registration: The pose of the object is computed wrongly, and the robot arm approaches the object from the wrong direction.

7.2.2 Evaluation of the Skill-Primitives within the complete system

The section briefly summarizes the failure rates of the various skill primitives. The table shows a high success rate for most of the skill primitives except for the primitive `placing_pose_learn`. An error analysis revealed the following reason: During the training of a placing pose, the object in the gripper of the robot arm needs to be moved manually into the right compartment. After that, the module `placing_pose_learn` is executed. The module now attempts to plan an arm movement from a pre-placing position to the current pose by simulating the movement. This is the movement the system will have to perform later during the execution of the placing skill. If the planning fails, the

primitive reports an error and the procedure of manually moving the object in the gripper into the right compartment needs to be repeated. The reasons for possible failures are:

- The gripper does not provide any information about the state of its fingers which means it is not known to the robot how wide open the fingers of the gripper are. Therefore, for safety reasons, the model of the gripper used for planning is always in "completely open" state. An open gripper requires more space when entering the kitting box which means that planning might fail even though the gripper holding an object (and thus with fingers closed) does fit into the kitting box compartment.

Therefore, this problem is due to the limits of the gripper. A solution is to place the object in the gripper or specify which distance to the kitting box bottom and the sides of the compartment and to possibly increase the size of the compartment.

- When approaching some kitting box compartments, especially the ones close to the robot base, the robot arm can come close to a singularity. This is a general problem for certain locations. A solution to avoid a singularity is, e.g., to test different placing poses with a tilted forearm. However, due to the wrist camera of the robot arm potentially colliding with the kitting box, the number of possibilities are limited so that a trial and error approach might be required.

Initially, the reasons for errors were not clear, and trial and error was used to identify the error causes. This is the reason for the high error rate of the `placing_pose_learn` in Table 3.

A way to reduce the error likelihood is by *not* try to find a very good fit for the object deep inside the compartment but by letting the object drop from a slightly higher distance (≈ 3 to 4 cm).

Primitive	Attempts	Success	Fails	Undetected fails	Crashes	Average time	
<code>locate_bonn</code>	121	120	1	0	0	2,71	99 %
<code>registration</code>	155	139	7	9	0	6,17	90 %
<code>arm_motion</code>	528	495	33	0	0	15,23	94 %
<code>kittingbox_registration</code>	82	82	0	0	0	2,65	100 %
<code>grasping_pose_learn</code>	8	8	0	0	0	3,31	100 %
<code>placing_pose_learn</code>	12	7	5	0	0	10,87	58 %
<code>object_train</code>	7	7	0	0	0	50,83	100 %

Table 3 shows the success rates, failure rates and execution times of the employed skill primitives.

7.3 Error Recovery Capability

In this section we discuss the error recovery capability of the STAMINA. The error recoverability of the system is due to the

1. pre- and post-condition checks of the skills that detect deviations between the world model and the real world
2. the task planner on the robot that is able to re-plan if some pre- and post-conditions are not met.

For each kitting order that is sent from FAKIR to the logistic planner a mission planner plans the sequence of skills that will allow completing the kitting order. This works under the assumption that the world model, based on which the planning happens, indeed reflects the true state of the physical world. Deviations between the world model and the physical world are detected by the pre-conditions and post-conditions of the skills. If deviations are detected, the world model is updated and the task-planner on the robot generates a new plan.

This is summarized in Table 4: In case for 4 parts, the original plan worked out only 4 times, but it had to re-plan 18 times. The re-planning was successful 17 times, but it failed once (second column). Of these 17 new plans the system had to plan again 11 times, out of which only one re-planning was successful and 10 re-plannings failed. All in all, the table says we have 3 kitting orders successfully executed with the original plan, 7 kitting orders successfully executed with one new plan and 1 kitting order successfully executed with two new plans. This visualizes the power of the task planner: The system is able to naturally deal with deviations between the world model and the physical world (a human might call these deviations *unexpected surpris*es).

It must be noted, however, that these numbers (11 successful kitting orders) count the cases where failures remained un-detected. If compared to Table 1, one can see that there were 4 undetected errors.

	0 replans	replan failed	1 replan	replan failed	2 replan	replan failed
3 parts	7	1	3	0	4	1
4 parts	4	1	17	10	1	
5 parts	1	1	1	1		

Table 4 shows how many plans and re-plannings it took to either complete the kitting orders or to request assistance: One can see that in most cases, more than one plan was necessary. E.g. for 4 part orders, the system had to re-plan once, in one case even twice. In some cases, the planning failed and the system needed to call for help because it could not handle the problem on its own.

8 Conclusions

In terms of the requirements in the DOW, this test sprint was a great success. The complete system is fully integrated and running with a reliability that can be considered high at this stage. The planner is able to handle a large number of errors autonomously, and some of the undetected

failures can be dealt with. E.g., the loosing of a part from the gripper can be detected using a force-torque sense on the wrist.

In terms of our own ambitions this test sprint was OK. The major disappointment stems from the difficulties with the pick_bonn skill. On the other hand the goal of the test sprint is to test and to identify errors that would have not been detected without the real-world experiments. The pick_bonn skill is exactly an example for this.

A number of suggestions from the partners were identified as a result of the test sprint:

- UBO has identified two sources of errors affecting the performance of the picking pipeline: motion planning and object pose estimation. This is mainly the result of the limitations of the platform present in the UBO lab: the platform at BAS is different hardware-wise and features additional hardware, such as the mobile base. In order to make the picking system stable UBO will address the following:
 1. extend the perception of parts by introducing new viewpoints of the object in the training phase.
 2. improve the arm motion by replacing planning with interpolation in the joint space for short trajectories.
 3. specify a set of requirements for the allowed poses of the mobile platform with respect to the pallets, such as the distance to the pallet and the relative angle. These tolerances will have to be determined experimentally.
- The 3rd test sprint has seen the complete system running successfully. The following steps must focus on increasing the success rate and cycle time, from the optimization of all the functional parts. In particular the following improvements are required for SkiROS:
 1. Remove completely the undetected fails, by increasing the pre and post condition checks
 2. Reduce the cycle time, by increasing the number of concurrent operations
 3. Improve the task re-planning

9 Appendix

The appendix contains recent relevant publications:

Action Selection for Interaction Management: Opportunities and Lessons for Automated Planning

R. Petrick, M.E. Foster.

Proceedings of the Workshop of the UK Planning and Scheduling Specialist Interest Group (UK PlanSIG), Glasgow, Scotland, UK.

Fast Edge-Based Detection and Localization of Transport Boxes and Pallets in RGB-D Images for Mobile Robot Bin Packing

D. Holz and S. Behnke.

Proceedings of the International Symposium on Robotics, VDE, 2016.

I Can Help! Cooperative Task Behaviour through Plan Recognition and Planning

C. Geib, B. Craenen, R. Petrick.

Proceedings of the Workshop of the UK Planning and Scheduling Special Interest Group (UK PlanSIG), 2016, Glasgow, Scotland, UK.

On the enhancement of Industrial Logistic Systems with Semantic 3D Spatial Representations

C. Toscano, M. Oliveiro, R. Arrais, G. Veiga.

IEEE Conference on Industrial Informatics INDIN2016.

Submitted January 2016.

Separating Representation, Reasoning, and Implementation for Interaction Management

M.E. Foster, R. Petrick.

Proceedings of the International Workshop on Spoken Dialogue Systems, 2016, Saariselkä, Finland.

A vertical and cyber-physical integration of cognitive robots in manufacturing

V. Krueger, A. Chazoule, M. Crosby, A. Lasnier, M.R. Pedersen, F. Roviada, L. Nalpantidis, R. Petrick, C. Toscano, and G. Veiga.

Proceedings of the IEEE, Special Issue on Industrial Cyber-Physical Systems. 2016

Vision-based robotic system for object agnostic placing operations

N. Rofalis, L. Nalpantidis, N. A. Andersen, and V. Krüger

International Conference on Computer Vision Theory and Applications (VISAPP). 2016