TrimBot2020 Deliverable D1.2

Platform 2: Supports online operation for demonstrator 1

Principal Author: BOSCH
Contributors: BOSCH, UEDIN
Dissemination: CO

Abstract: This document presents the extensions necessary to implement online operation for platform 2. Therefore, the previously used hardware of platform 1 has to be changed regarding the network configuration and the camera setup. Additionally to these changes a wheel based odometry capability is added to the vehicle. The software architecture needed for the online operation is also presented in this document. In this regard, the software packages for the cameras and the Semantic SLAM are only briefly introduced, while the user interface (GardenUI) and the navigation components, in particular the state machines, are described in more detail.

Deliverable due: Month 18

1 Introduction

Platform 2 extends the previous platform 1 with online operation and navigation features. This is needed for the first Trimbot demo, which is about vehicle navigation. Platform 1, presented in D6.1 (First integrated platform), is therefore extended by a Wifi network router to establish a connection between the vehicle and a remote laptop. The remote laptop is used to run the graphical user interface (GUI) which controls the online operation. Figure 1 shows the remote laptop and the vehicle of platform 2 during an online operation in the test garden in Renningen. A detailed description of the hardware setup for platform 2 is given in Section 2. An overview of the software architecture used for the online operation is presented in Section 3. As previously defined the software is based on the Robot Operating System (ROS) and therefore all interfaces are implemented as ROS messages.

Figure 1: Online operation of the platform 2 vehicle with remote Laptop in the test garden in Renningen. The GUI on the remote laptop is used to set destinations for the vehicle navigation.

2 Hardware Setup

The hardware setup for the platform 2 vehicle is nearly the same as for the platform 1. It is based on the same modified BOSCH Indego lawn mower with the same aluminium frame. In contrast to platform 1 the aluminium frame of platform 2 is able to carry an additional onboard laptop for SLAM and a Wifi router for the network connection. The sensor setup is, apart from the cameras, the same as for platform 1. For this reason, only the new camera setup is presented in the following, and all other sensors and their integration are described in D6.1 (First Integrated Platform). Figure 2 shows the platform 2 vehicle with the new camera setup, embedded PC, Wifi router and the space for an onboard Laptop. For the online operation the vehicle is also extended by odometry. However, the sensors for the odometry are inside the lawn mower and can therefore not been seen in Figure 2. The network configuration with the Wifi router allows to control the vehicle from a remote laptop. Both the vehicle odometry and the network configuration are presented in more detail in the following.

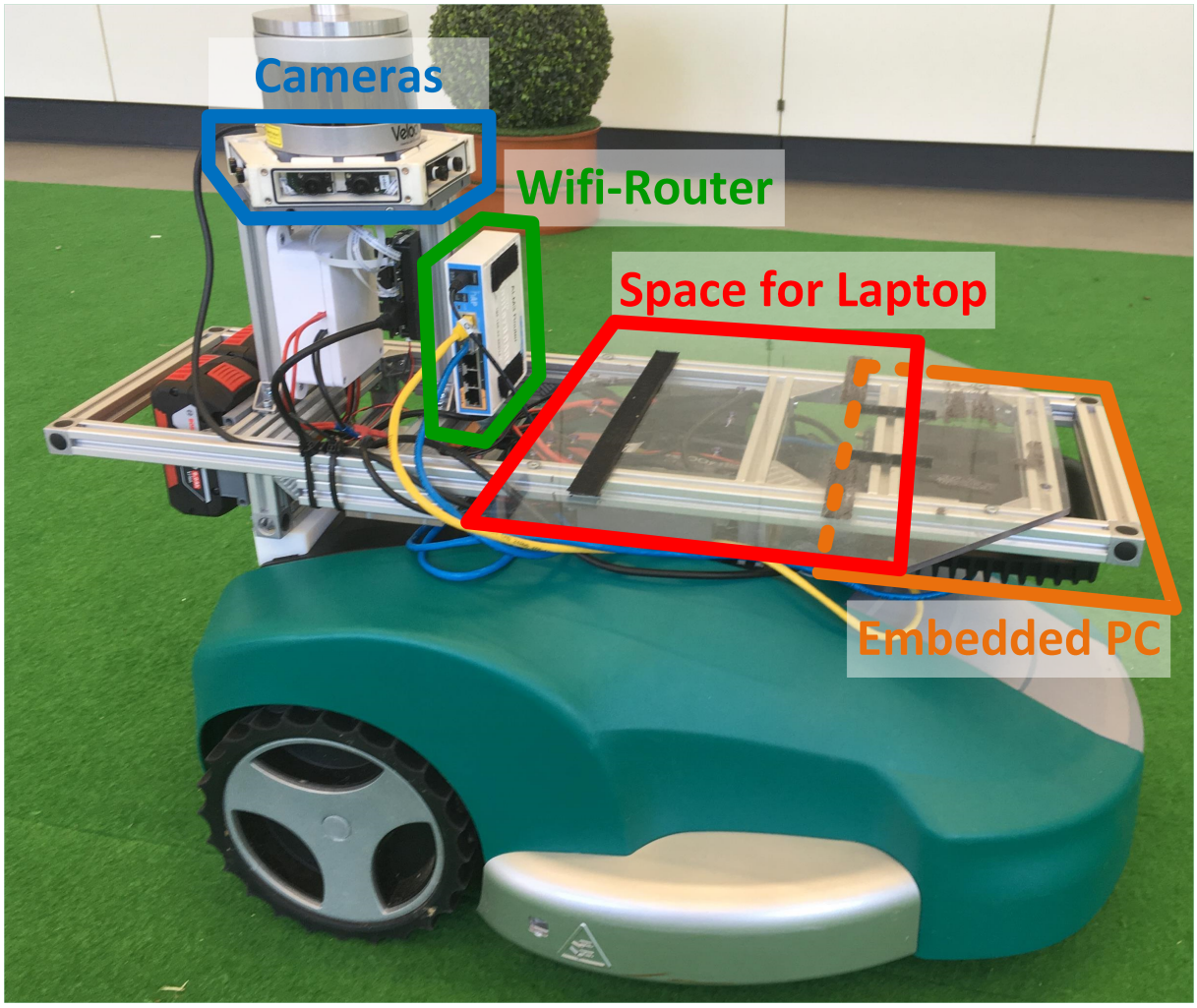

Figure 2: The vehicle for platform 2 with the new camera setup (blue), the Wifi router (green), space for an onboard Laptop (red) and the embedded PC (orange).

2.1 Vehicle Odometry

Information about the vehicle's position and orientation is indispensable for navigation tasks. For platform 2 the position of the vehicle w.r.t. the garden map will be provided by the visual SLAM from workpackage three and is therefore only briefly introduced in Section 3.2. To support the SLAM algorithm a wheel based odomentry is implemented on the vehicle itself. This odometry is based on the differential drive setup of the Indego lawn mower with integrated wheel encoders for each wheel. With these encoders the distances travelled by the two wheels d_r, d_l are measured at fixed time intervals $(t-1, t)$. Based on these measurements the travelled distance for the vehicle center d_c is calculated by:

$$d_c = \frac{d_r + d_l}{2} \tag{1}$$

With the assumption that for small values of x, the result of $sin(x)$ is approximately the same as x, the orientation change Φ of the vehicle within small time interval can be calculated by

$$\Phi = \frac{d_r - d_f}{b} \tag{2}$$

where b is the baseline between the two wheels. For dead reckoning the orientation change and the travelled distance are integrated over time by:

$$\Theta(t) = \Theta(t-1) + \Phi \tag{3}$$

$$x(t) = x(t-1) + d_c * cos(\Theta(t)) \tag{4}$$

$$y(t) = y(t-1) + d_c * sin(\Theta(t)) \tag{5}$$

where $x(t)$ and $y(t)$ are the position and $\Theta(t)$ is the orientation of the vehicle at time t w.r.t. the vehicle's start location. The dead reckoning position will be refined by the visual position derived from the SLAM system.

2.2 Vehicle Cameras

At the project meeting in Freiburg (16. - 17.1.2017), the decision was made that the previously used camera ring consisting of 8 cameras should be replaced by a pentagon with 5 stereo camera pairs. This new setup will now be used for the platform 2 and also for the final demonstrator platform 3. Figure 3 shows the new camera setup with five stereo camera pairs. A camera housing for this pentagonal setup was built by the project partner from WR. Figure 4 shows this new housing.

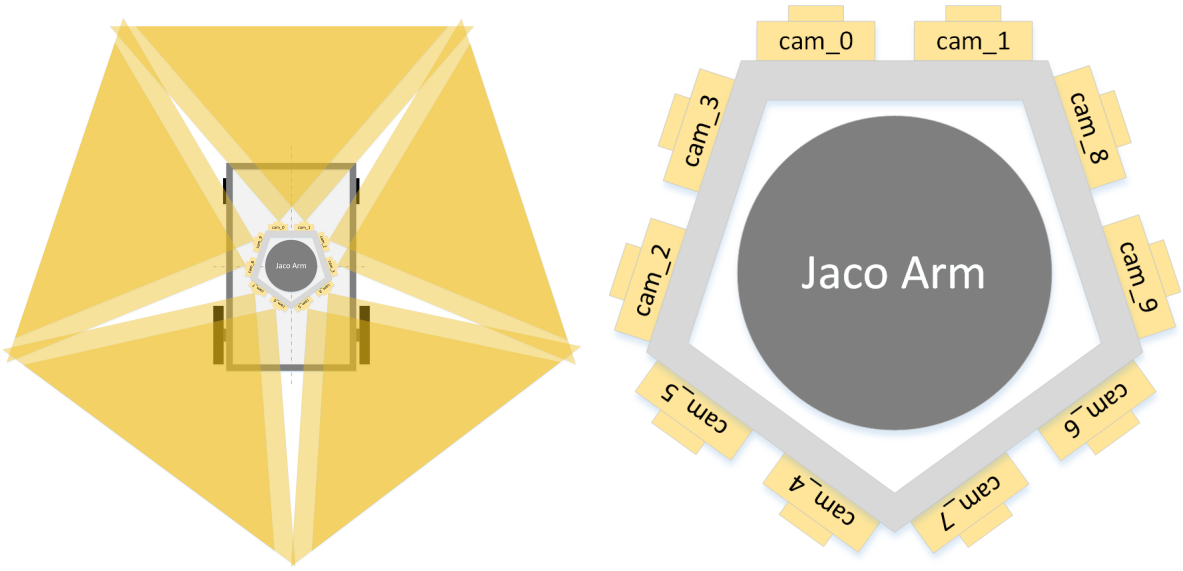

Figure 3: Schematic of the camera setup with five stereo camera pairs. Left: Schematic of the stereo visibility. Right: Detailed camera arrangement.

2.3 Network for Online Operation

For the online operation a continuous connection between the remote laptop and the vehicle with embedded PC and onboard laptop is necessary to send user commands to the vehicle. Therefore, a Wifi network is established using a Wifi router on the vehicle. This configuration with the Wifi router on the vehicle allows to have the same network configuration anywhere.

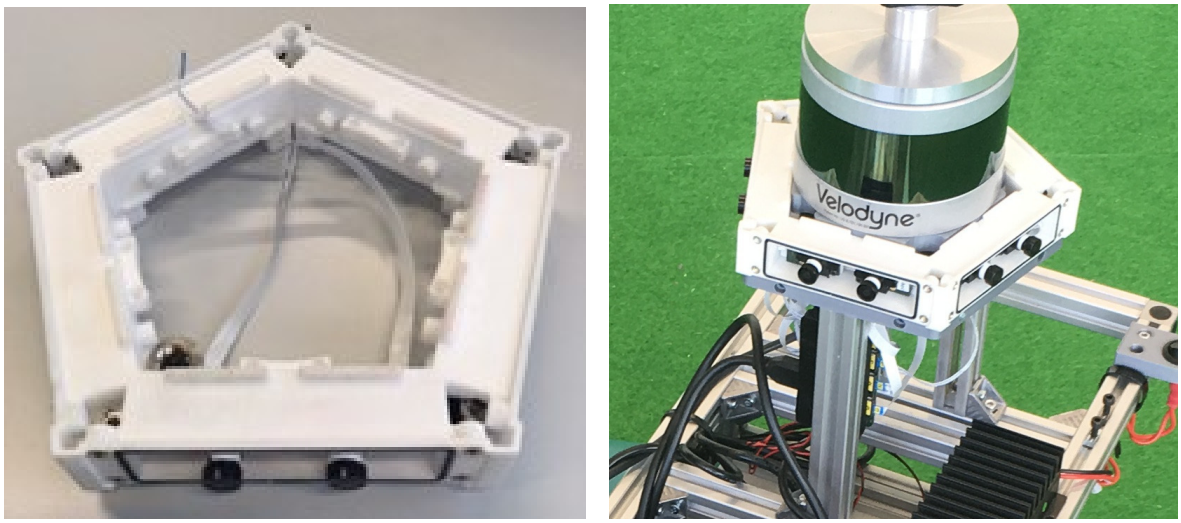

Figure 4: Housing for the pentagon camera setup.

Thus, platform 2 can either be used in the test garden in Wageningen or in the test garden in Renningen without changing the network configuration. The embedded PC and the onboard laptop are connected via Ethernet to the router and the remote Laptop is connected via Wifi. The embedded PC will be the ROS master and has therefore a static IP of 192.168.33.10. To establish a ROS communication between several machines the IP address of the ROS master has as to be set as environment variable `ROS_MASTER_URI`. Therefore, the IP and the ROS port from the embedded PC has to be set as `ROS_MASTER_URI` on the onboard laptop and the remote laptop. Further the own IP address has to be set as `ROS_IP` on all machines.

```
export␣ROS_MASTER_URI=http://191.168.33.10:11311
export␣ROS_IP=http://191.168.33.XX
```

3 Software Architecture

The software architecture for online operation of platform 2 includes four software packages: Camera, SLAM, SketchMap and Navigation. The interfaces between these packages are defined as ROS messages. An overview of the software architecture including all the packages and the messages used for the communication between the packages is shown in Figure 5. Each package encapsulates the software parts from one partner and can therefore contain several components. The messages used within a package are not included in the overview, since they are only used by one partner and are therefore handled by that partner themselves.

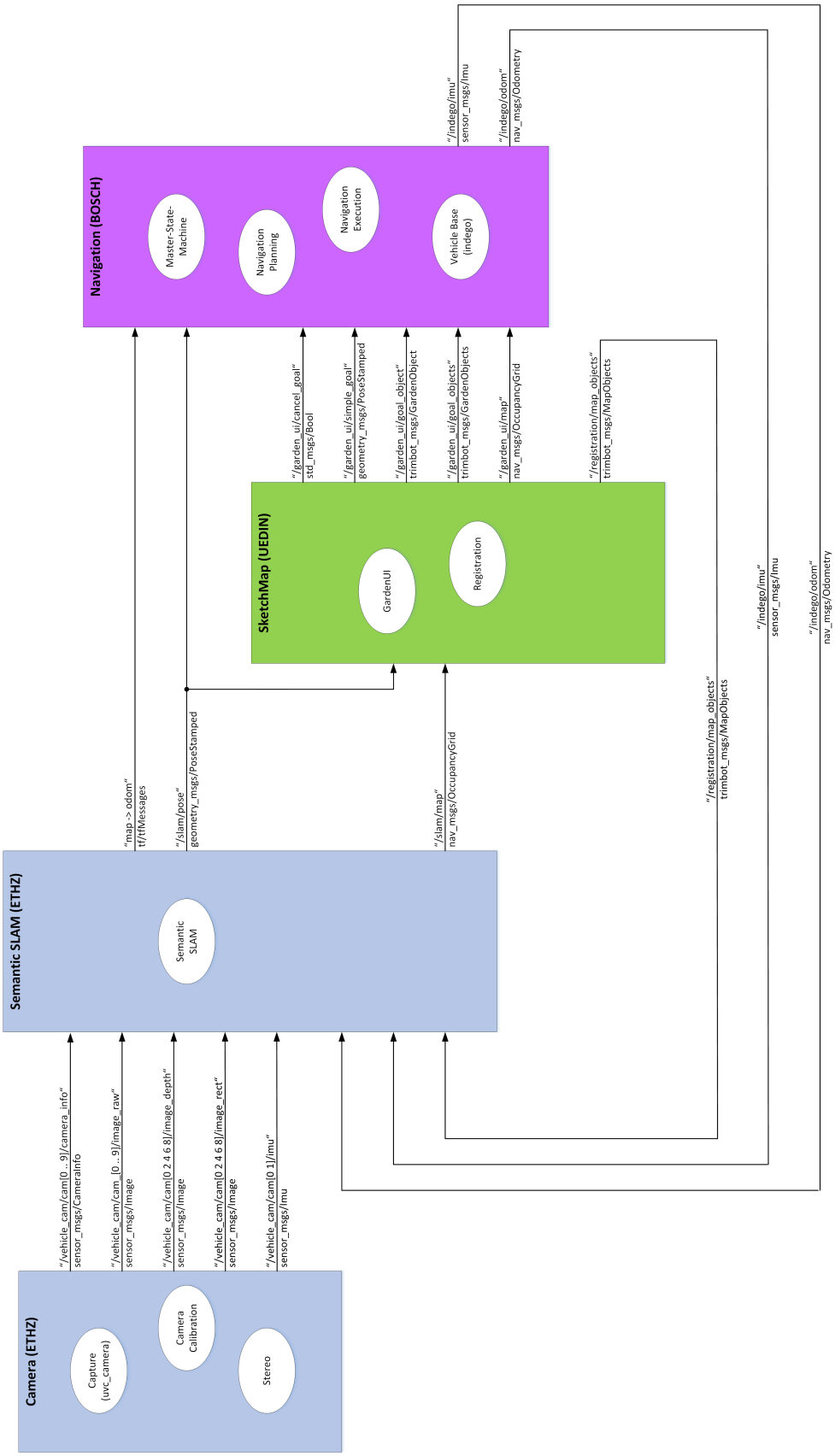

Figure 5: Overview of the software architecture for platform 2.

3.1 Camera

The camera package includes the camera calibration and the camera driver (`uvc_camera`). The stereo image processing is done on the camera itself. For this stereo processing a calibration file with all the camera parameters is needed. This file is generated by the camera calibration component and loaded to the cameras by the `uvc_camera` driver. The `uvc_camera` driver also provides the raw image stream and the camera info for the each camera. On each camera pair the left camera provides a colour image and the right camera provides a grey scale image. For the colour images the rectified image is also provided by the camera driver. Additionally, to these images the depth image for each camera pair is provided.

3.2 Semantic SLAM

Semantic SLAM is part of workpackage three and therefore documented in detail in the deliverables of workpackage three. In this document only the function of Semantic SLAM with regard to online operation and navigation is briefly described. The Sematic SLAM package is used to process input from cameras and build a map of the environment and localise the robot w.r.t. this map. The main product of the SLAM algorithm is a 3D point cloud where points are associated with image descriptors. However, to support the navigation task it also generates an occupancy grid to communicate the positions of static obstacles in addition. The SLAM algorithm also provides the position of the robot w.r.t. the map as a 3D pose. The global coordinate system used for the robot localisation is given by the map and the derived 2D occupancy grid. The scale is given by camera calibration and results in approximately metrical (world) units.

3.3 Sketch Map

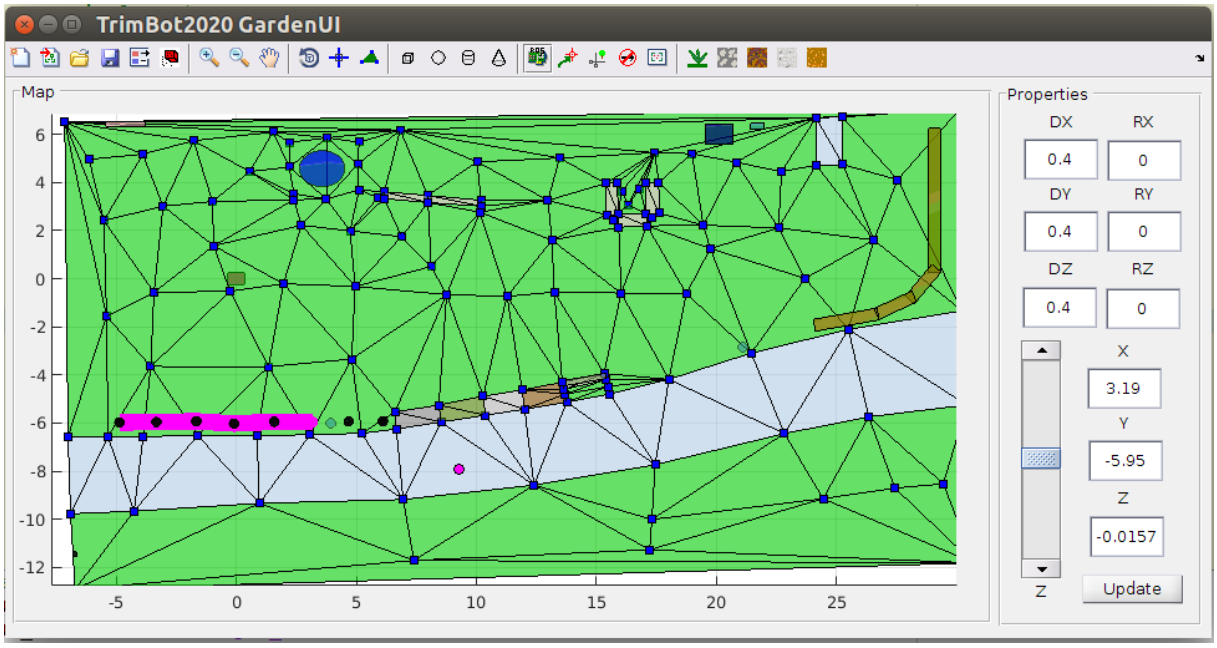

Figure 6: Garden User Interface. Magenta line: sequence of objects selected for trimming. Blue markers: control points of the surface mesh.

The Garden User Interface (`GardenUI`) allows an user to specify a 2.5D semantic sketch map of a garden (Figure 6). The sketch map has two main parts, the terrain represented by a mesh surface and a set of objects represented with primitive shapes (cube, sphere, etc.). Every segment of the terrain and every object has attached a semantic label (grass, hedge, topiary etc.). See *D7.4 - Ground-truth data definitions and acquisition* for complete list of semantic labels and their color coding.

The sketch map must be initially registered to the SLAM map in order to issue correct coordinates for navigation. Currently the registration transform is rigid because the sketch map is drawn with the help of metric measurements from the garden plan. A later version will allow less accurate initial sketch maps that will be subsequently refined by deformable registration with the sketch map.

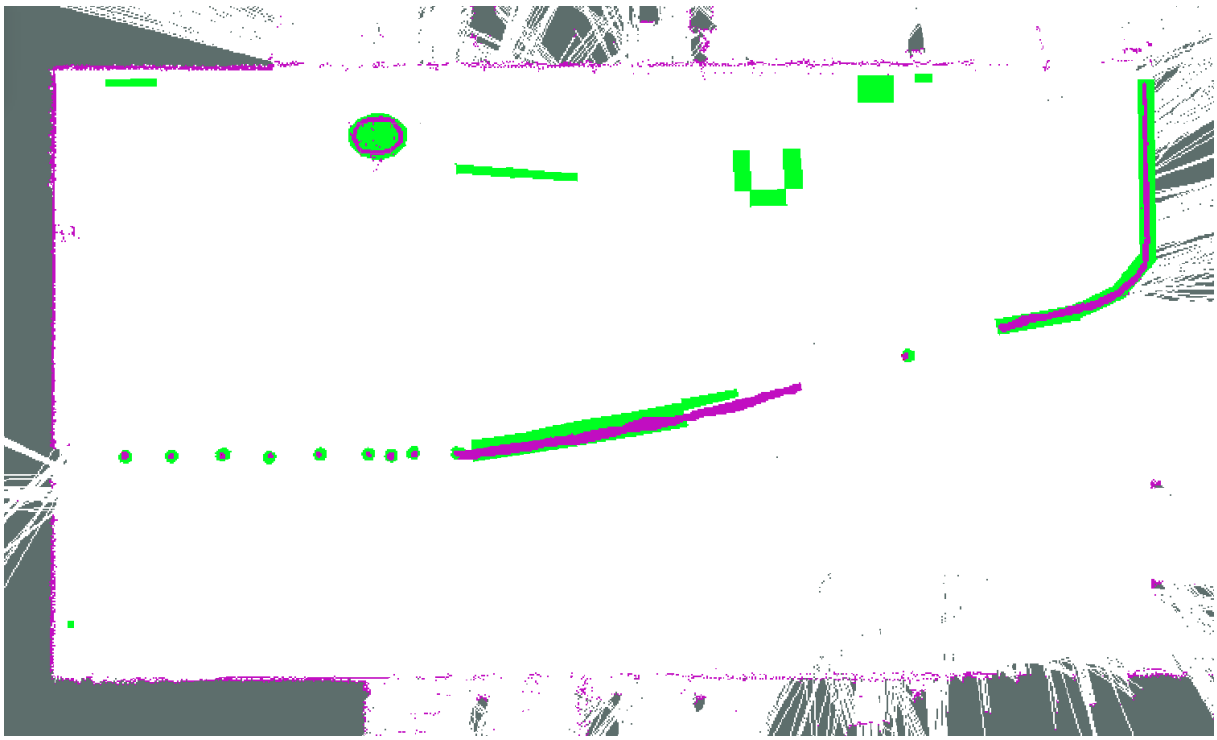

Figure 7: Occupancy grid for navigation. White: free space. Magenta: obstacles from SLAM. Green: obstacles from sketch map. Grey: unseen space.

The `Register map` button retrieves the current 2D occupancy grid from SLAM and computes a rigid 3D transform by matching map objects to objects localised in the occupancy grid. Objects are segmented in the image of the occupancy grid as connected components when height exceeds a given threshold. The registration transform is then obtained using the Iterative Closest Point (ICP) algorithm run on the two sets of object center points.

The interface with a loaded and registered map can be used to issue commands to the robot. First a ROS node must be started using `ROS init` button and entering the ROS master URI, as described in Section 2.3. Then the robot, whose pose is also displayed in the map, can be sent to a specific location using `Navigate to location` button and clicking in the map,

which results in a message with the corresponding XY coordinates (`PoseStamped` message). Alternatively the robot can be sent to a currently selected object using `Trim object` button, then the `GardenObject` message is published[1]. When multiple objects are selected, they are sent in `GardenObjects` sequence instead. The current action can be cancelled with a dedicated button. GardenUI also updates the occupancy grid from SLAM with semantic information, i.e. indicates terrain which is not drivable (Figure. 7). Only surface tiles labeled as grass or pavement are considered safe to drive over.

3.4 Navigation

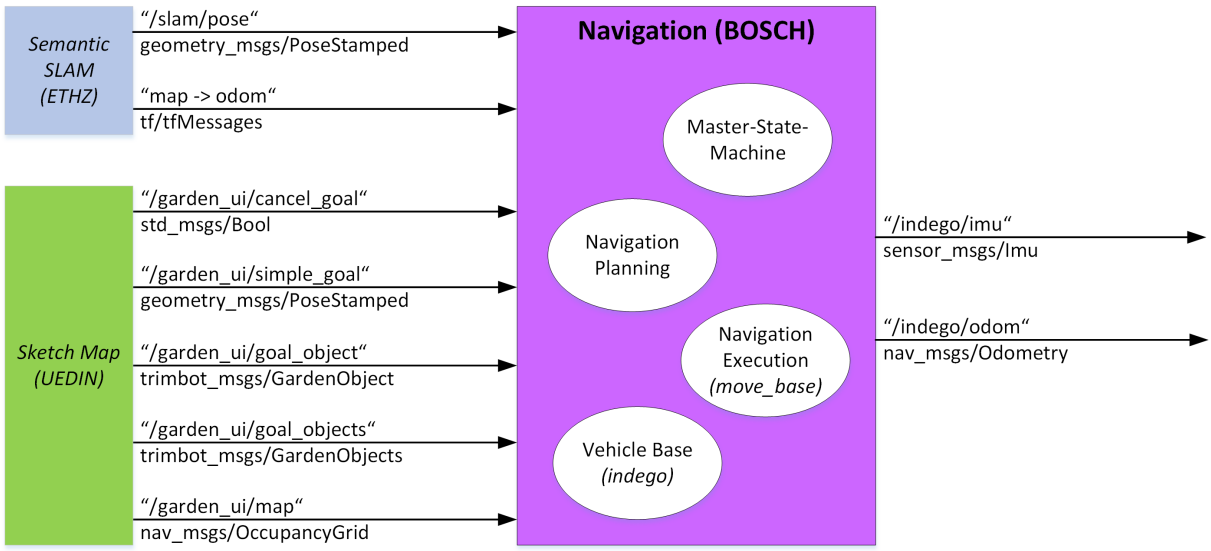

Figure 8: Interfaces with ROS messages used by the navigation package.

The navigation package includes the master state machine, the navigation planning, the navigation execution (`move_base`) and the vehicle driver (`indego`). Figure 8 gives an overview of the navigation package with all inputs to these package and the provided outputs. The inputs to the navigation package come from the Semantic SLAM package and the Sketch Map packages. The Semantic SLAM has to provide for the navigation package the position of the vehicle w.r.t. the map as a `PoseStamped` message and a `tf` message for the *map-to-odometry* transformation. More information about the map-to-odometry transformation[2][3] strategy can be found at the ROS Wiki. The navigation package gets also inputs from the `GardenUI` component of the Sketch Map package. These inputs are the registered map as an `OccupancyGrid` message, the goals where the vehicle has to naviagte to as a simple goal (`PoseStamped`), a goal object (`GardenObject`) or a list of goal objects (`GardenObjects`) and the message to cancel the current goal action. The difference between a simple goal and a goal object is that the

[1] For the purpose of the first demo the nested `MapObject` message contains the identical description as the parent `GardenObject`. It will be later changed to the trimming shape description.

[2] ROS Enhancement Proposals (REP) for naming conventions and semantic meaning of mobile platform coordinate frames: `http://www.ros.org/reps/rep-0105.html`

[3] Documentation of the ROS standard adaptive Monte Carlo localisation (AMCL) including an explanation of the map to odom transformation: `http://wiki.ros.org/amcl#Transforms`

simple goal is just a coordinate in the garden to which the vehicle has to drive and a goal object is an object which has to be trimmed e.g. a bush, a hedge or a rose. If the goal for the navigation is a goal object the navigation planning has to calculate a position in front of this object and an approaching manoeuvre to reach a good trimming position for this object. Therefore, the navigation planning component consist of several nodes for finding the best trimming position and an approaching manoeuvre.

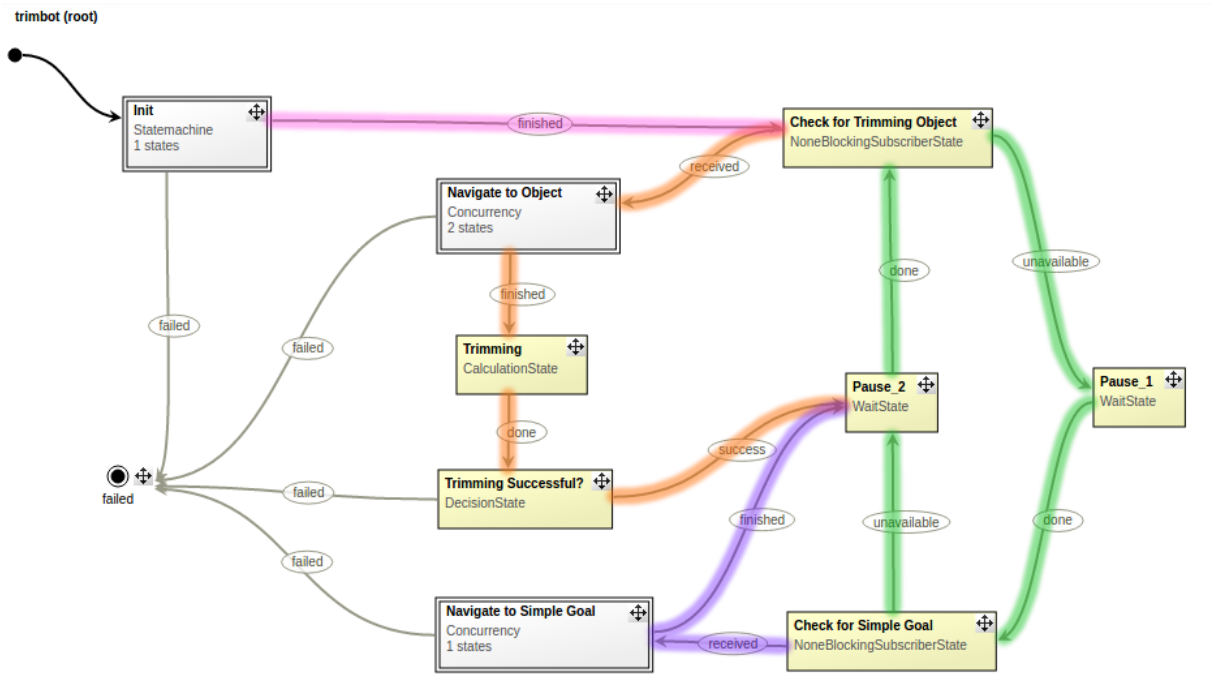

Figure 9: Master State Machine: The pink transition is activated after a successful initialisation. The green transitions build a loop in which the state machine waits for new user goals. The orange transitions show the procedure if a message for a trimming object is received and the purple transitions show the procedure if a message for a simple goal is received.

The master state machine controls the behaviour of the robot. It is implemented with FlexBE[4] which allows an easy modelling and implementation of state machines for ROS. FlexBE has some pre-implemented states e.g. ROS message subscriber, logging states or calculation states. FlexBE also allows to interleave state machines. In Trimbot, this concept is used to encapsulate the state machines for navigation and trimming to keep the master state machine simple. The master state machine used for the online operation of platform 2 is shown in Figure 9. The grey boxes are interleaved state machines and the yellow boxes are states. The arrows between these boxes are the transitions with their conditions. The master state machine consist of an `Init` state machine where the vehicle and all sensors are initialised. After a successful initialisation (pink transition) the master state machine circulates between two ROS message subscriber states (green transitions). Between the two subscriber states two additional `Wait` states are included. The first subscriber states is the `Check for Trimming Object` which checks if a `gardenObject` message is received. If such a message is received the

[4] `http://wiki.ros.org/flexbe`

orange transition to the `Navigate to Object` is activated and the navigation state machine will be entered. After a successful navigation the state machine will go to the `Trimming` state. In this version of the state machine the `Trimming` and `Trimming Successful` states are empty and only passed to go back to the *wait for user goal circle* (green transitions). The second subscriber state is the `Check for Simple Goal` which checks if a `simpleGoal` message is reveived. If this message is received the purple transition to the `Navigate to Simple Goal` is activated and this state machine will be entered. After a successful navigation the state machine will return to the *wait for next user goal loop* (green transitions). For the final demonstrator 3 the `Trimming` and `Trimming Successful` states will be replaced by more complexed state machines and a recovery behavior will also be integrated.

The `Navigate to Object` state machine is shown in Figure 10. First, in this state machine the best trimming position in front of the object is calculated. After logging and validating this position the `Drive to Destination` state triggers the `move_base` component to navigate the vehicle to this position. If the vehicle has reached this position a final approaching manoeuvre is performed by the `Approach Object` state.

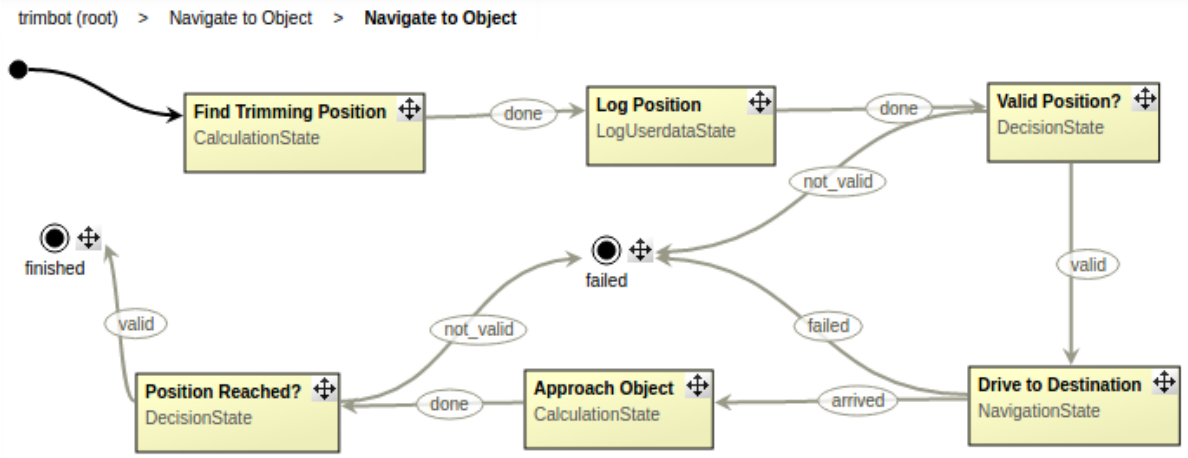

Figure 10: The state machine for the navigation to a trimming object.

The `Navigate to Simple Goal` state machine is shown in Figure 11. Based on the user defined goal position, this state machine first calculates in `Find Simple Goal Position` a position which the vehicle can safely reach. After logging and validating this position the `Drive to Destination` state triggers the `move_base` component to navigate the vehicle to this position. For this purpose, the `move_base` component plans a path taking into account all the obstacles listed in the map, thus avoiding all these obstacles.

The ROS `move_base` is used for the navigation execution to navigate the vehicle to a given destination. The goal position for the `move_base` is provided by the navigation planning which calculates for each user defined goal a position to which the vehicle can navigate taking into account the traversable areas in the map. This is needed to avoid goal positions which are inside an obstacle or to close to an obstacle and therefore not reachable by the vehicle. The navigation planning is triggered either by the `Find Trimming Position` or `Find Simple`

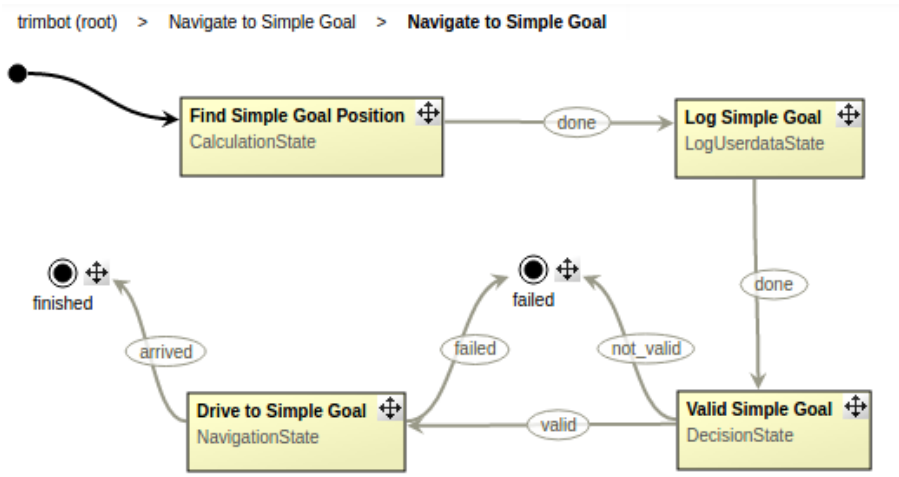

Figure 11: The state machine for the navigation to a simple goal.

`Goal Position` states of the state machines. Based on the goal position calculated by the navigation planning, `move_base` calculates a collision free path and navigates the vehicle along this path. The `indego` driver is used to send the velocity commands from `move_base` to the vehicle itself. This node also provides odometry and IMU data from the vehicle which can then be used by other packages like the Semantic SLAM package.

