Published in Proceedings of ISR 2016: 47st International Symposium on Robotics, 2016. Publisher: VDE, ISBN: 978-3-8007-4231-8

Two Autonomous Robots for the DLR SpaceBot Cup – Lessons Learned from 60 Minutes on the Moon

Sven Lange, Daniel Wunschel, Stefan Schubert, Tim Pfeifer, Peter Weissig, Andreas Uhlig, Martina Truschzinski and Peter Protzel, TU Chemnitz, Dept. of Electrical Engeneering and Information Technology, Germany*

Abstract

The SpaceBot Cup 2015 was a national robotics contest among 10 German universities and research institutes. It was organized by the German Aerospace Agency DLR and required 60 minutes of autonomous operation in a challenging environment. The robots had to explore and map the environment, find, transport and manipulate two objects, and navigate back to the landing site without global position information like GPS under very restricted communication. This paper describes our concept, the used systems and algorithms, and our experiences in terms of lessons learned.

1 Introduction

The DLR Space Administration decided in 2012 to host a national competition called SpaceBot Cup to foster new ideas and to assess the current state of the art of autonomous robotics for planetary explorations as well as for terrestrial applications. Ten German universities and research institutes were selected as participating teams. After eight month of preparation, the SpaceBot Cup was held in November 2013, finishing with unexpected results. In contrast to the promising robotics communities' research papers, no participant could solve the given task. Mostly the teams got stuck right in the beginning, due to the restricted communication and the accompanying problems, which will be explained later on. As a consequence, the DLR Space Administration started a new call in 2014 with similar terms. Again, ten teams were selected and funded with €50,000. In the following, we will describe the required tasks, the technical choices made to build a team of two ground robots, challenges and performance issues, and the lessons learned from the competition.

With this kind of field report about our experiences with a complex system applied to a very challenging mission, we want to provide some insights and suggestions applicable to a variety of similar tasks.

2 The SpaceBot Cup

The SpaceBot Cup 2015 was organized starting with a Kick-Off-Meeting in December 2014, followed by several individual team status meetings, leading to a qualification stage, and the final competition in November 2015.

The mission of the SpaceBot Cup 2015 resembled the tasks of the first SpaceBot Cup challenge in 2013, in which we also participated [1]. On a simulated moon-like planetary surface built inside a television studio, the task was to explore the environment autonomously as well as finding, assembling and transporting different objects.

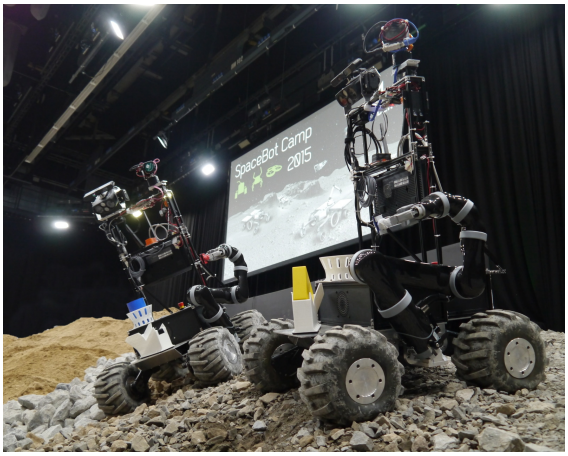

Figure 1 Our two identical robots named Phobos and Deimos on the rugged terrain during the SpaceBot Camp 2015.

The environment had about 30 m × 20 m of rugged realistically modeled surface, which was composed of different materials like gravel and fine sand as well as boulders of different size, trenches and small hills of up to 2 m height and 30° slopes. A landing site marked the starting point for the robots where they had to start their autonomous exploration (obviously without GNSS) navigating around dangerous soil or slopes. At unknown locations, a battery pack (a yellow, rectangular cuboid) of 800 g weight and an open mug filled with about 200 g of a soil sample[1] had to be discovered. The robot had to pick up and transport the

*e-mail: {firstname.lastname}@etit.tu-chemnitz.de

[1] As an optional task, the soil sample could be gathered by the robots using an empty mug.

Figure 2 A panoramic view of the final stage of the competition. The complete area is divided by a black barrier into two identical fields to allow parallel runs.

detected objects to a third object called base station. After reaching the base station, the objects had to be assembled. By activating a switch, the sub-task was fulfilled. Within the remaining time, all robotic systems had to return to the landing site.

Besides completing the described task within one hour, other constraints had to be considered. Similar to real missions, a communication delay of 2 s with possible packet loss was implemented by the organizers. To further tackle the reality of the mission, a spatial separation of the planetary environment and the robots' control station was ensured. The operators could monitor the planetary robots only by the delayed sensor data. Within three time slots of 5 min each, it was possible to send commands back to the planet's surface.

Although a realistic terrain causes specific conditions like special temperature, radiation or atmospheric pressure, they were not considered as the competitors focused on the autonomy of the systems. Therefore, RGB-D sensors like the Kinect were allowed and usable due to the indoor setting.

In contrast to the last SpaceBot Cup, a qualification stage was held about two month before the finals to foster communication between the teams and to show their ability to solve the final mission. Surprisingly, only three out of ten teams were able to solve all stages of this qualification. Luckily, we were one of them. According to the rules, at least four teams had to pass the qualification for the SpaceBot Cup finals to take place. The organizers decided to have a final showcase with a simplified mission rulebook instead of a ranked competition and called it SpaceBot Camp.

Due to a tight schedule, two teams performed their runs simultaneously on the left and right side of the arena (see Fig. 2). In contrast to the original task, the robots started on a plane but slightly rough area. They had to find and grab the battery and mug, which were placed in about 5 m and 10 m distance. Booth objects had to be assembled at the base station, which was positioned on a 2 m high plateau with a steep slope at its edge.

3 System Overview

In retrospect, we gained considerable expertise from the past SpaceBot competition. Hence, we give a short review of the missing or flawed components in 2013 followed by a detailed overview of our new hardware design.

3.1 Pitfalls in 2013

The most relevant component which was obviously neglected by many teams, including ours, was the *communication between the ground station and the robot*. It is just an engineering task, but time-consuming and not very research relevant. In retrospect, we could not determine a specific source of error. There appeared to be flaws in our solution as well as in the organizers' communication scripts.

Clearly, to *rely solely on one RGBD sensor* for collecting environment information is insufficient, for instance in proximity of light absorbing materials or intense back light. In 2013 this probably caused the collision of one robot with a missed obstacle and lead to the termination of our run after about 15 min.

Also, *self made hardware solutions* lead to maintenance overhead and require a high amount of manpower, so we banished our custom made manipulator and replaced the needed actuator by a commercially available one, as described later on.

Our solution to use a *UAS (Unmanned Aerial System) with offline processing* during the SpaceBot Cup event with multiple WLANs being active, showed to be not reliable. Communication dropouts and increasing time delay led to instability and an emergency landing outside the mission's arena. Therefore, we focused on another UAS with adequate payload to carry sensors and enough processing power for on-board processing.

Despite these problems, several components proved to be well working and robust enough for the task as described in the following sections.

3.2 The Hardware Platform

The described mission scenario of the SpaceBot Cup with its rough terrain, called for an extremely capable robot platform. After a careful consideration of the commercially available platforms, we decided to use two *Summit XL* rovers from the Spanish manufacturer Robotnik as a platform. The skid-steered robots are capable of carrying up to 20 kg payload. With their independent wheel suspension, the large contoured tires, and the powerful 4×250 W drive, these robots proved to be a good choice for the type of terrain encountered during the contest. For mastering the given tasks autonomously, we had to do extensive hardware modifications and systems integration (see Fig. 1), resulting in a robotic system based on commercially available hardware

and Open Source Software (ROS – Robot Operating System) for mobile manipulation in rough terrain.

Our mission concept includes the deployment of a team of two ground robots. Despite the relatively lightweight robots (35 kg without additional sensors and casing), we were limited by the competition rules of deploying a maximum of 100 kg. This leads to a maximum of about 14 kg of additional payload for each robot, leaving 2 kg for communication infrastructure on the landing site.

Since each robot should be able to complete the mission on its own, we have to equip each with a manipulator (in our case the weight is 5.3 kg), which results in a remaining payload of 8.7 kg for sensors, PC, communication devices and the enclosing structure.

Figure 3 Left: The aluminium structure for mounting the manipulator as well as the layout of our aluminium profiles as base for sensors and additional devices. Right: Close-up of the custom made PTU with the MultiSense S7 and the Asus Xtion mounted on it.

To be prepared for future enhancements, we use building kit like aluminium profiles of the type *MakerBeam* for the basic structure. With its 1 cm × 1 cm profile dimension, they are lightweight and still strong enough for our needs. The structure without devices, is shown in Fig. 3. A sensor tower with a height of 69 cm (base to top) is placed approximately in the middle part of the robot, with a removable top for easy transportation.

Our overall sensor concept includes low-level sensors, namely the *wheel odometry* provided by the ROS interface of the Summit XL platform as well as an *Xsens MTi-G IMU* for 3D orientation estimation. Additionally, we use a *MultiSense S7* stereo camera from Carnegie Robotics together with the *Asus Xtion* for visual odometry as well as map generation. A rotating *Hokuyo UTM-30LX* 2D laser scanner completes our sensor concept in terms of global localization and mapping.

Both of our cameras are mounted together on a Pan-Tilt-Unit (PTU) (see Fig. 3), which is constructed with *Dynamixel MX-64* servos. This allows different camera angles for driving and grasping and furthermore enables us to inspect a wide area without the need of moving the robot itself. Our Pan-Tilt-Unit is a custom-made construction, where the heavy MultiSense S7 (1.5 kg) rotates around its center of mass to avoid constant load of the servo motor. To keep it simple, we used Teflon discs as a kind of bearing between the stiff and moving aluminium parts to reduce the friction.

In the central part our robot casing, a standard Mini-ITX board with an Intel Core i7-3770 CPU is mounted onto a vibration damped aluminium plate. All sensors and actors are connected via USB – except for the MultiSense S7, which has an Ethernet interface. The robot's motor controllers are interfaced by an USB to CAN bus adapter.

For on-field communication, we used three Netgear AC 1900 WiFi routers. One on board of each rover, configured in bridge mode, and one acting as stationary access point connected to the organizer's network route towards our ground control station. A management device within this route enforced communication restrictions like delay and port limitations. We have chosen these routers to provide a 5 GHz wireless link, without having to think about network drivers or configurations. Additionally, we connected an ARM based *Odroid-XU4* mini computer to the access point router. This mini computer served as relay station between the rover network and the link towards the ground station.

4 Localization

Solving the mission task autonomously requires a precise localization within the previously unknown environment. This is a typical SLAM problem, which is well known in the robotics literature. We decided to divide our localization and mapping problem into a local and a global component. This is necessary as one part uses sensors with a fast data rate whereas the other part depends on a well suited but slow sensor. The first part is called local localization - the robot estimates its relative motion continuously without any global consistency (see subsection 4.1). However, a global consistency is necessary to know the actual pose in the world. Therefore, the second part of localization is termed global localization - the task to build a map of the environment and estimate the own position and orientation inside this map (see subsection 4.2). Both the local and the global localization are shown in the first two layers in Fig. 4.

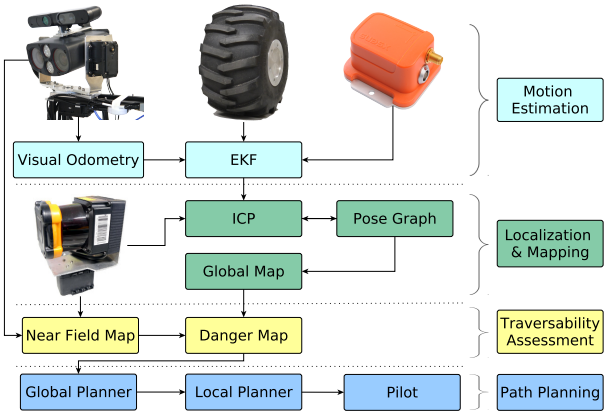

Figure 4 The localization, mapping, and navigation concept with its corresponding data flow.

4.1 Local

The local localization estimates the relative motion from the robot between two time steps - i.e. it determines how the

robot is moving without global consistency. Accordingly, a motion estimation is needed which returns position changes, velocities or accelerations for translation as well as rotation. In order to combine advantages of different sensors and algorithms a data fusion is needed as depicted in the upper layer in Fig. 4.

In our system, we employ three kinds of sensor information: wheel odometry, visual odometry, and orientation of an IMU. The wheel odometry is returned by the SummitXL driver and contains 2D position changes. In contrast, the visual odometry computes a 3D pose by comparing subsequent frames from a camera. As our robot is equipped with two types of cameras, we employ two different algorithms for visual odometry: The RGB-D camera Asus Xtion is used in combination with the algorithm *FOVIS* [2] whereas the stereo camera Multisense S7 images are the input for *LIBVISO2* [3]. The third source of information is the IMU Xsens MTi-G returning raw information of its internal gyroscope, accelerometer, and magnetometer. Furthermore, it fuses this information internally to return an orientation estimate, which is essential for our robot's pose estimation. Unfortunately, after each driver restart, the internal filter needs a settling time of about 15 min to get usable bias values (drift compensation). As we may restart our robot or parts of the software components during the mission, instantaneous orientation information is essential. So, we decided to use the ROS package *imu_filter_madgwick*[2] [4], which gives in our case nearly the same estimation quality like the IMU's internal filter, after bias settling, but in a nearly instantaneous manner.

These three sensor sources compose the input of an EKF (Extended Kalman Filter), implemented within the *robot_pose_ekf*[3] package. Since it accepts merely one visual odometry input, we always have to choose between the both mentioned visual odometry algorithms. However, in case of sunlight the stereo camera should be preferred whereas the RGB-D camera is advantageous in indoor environments with less textured surfaces. An alternative ROS package for odometry fusion is the *robot_localization* package[4] [5] which takes more than one visual odometry estimation.

As mentioned earlier in this paper, the cameras for visual odometry are attached to a PTU. While the visual odometry is running, the PTU must not rotate in order to avoid motion estimation errors. Accordingly, the robot pauses the visual odometry during zero motion to enable the PTU to rotate. In addition, this reduces the CPU workload while the robot is not moving which is needed by other tasks like object recognition, robot arm trajectory planning, and global position estimation.

4.2 Global

The second layer in Fig. 4 shows the concept of our global localization. The central component of the system is a map consisting of all consecutively acquired 3D laserscans. A special custom-made rotating base comprising a slip-ring and a servo motor is used for generating a 3D scan with the help of a 2D laserscanner (Hokuyo UTM-30LX). More details about the custom-made 3D laserscanner can be found on our website[5] including a mechanical drawing as well as images and videos. In summary, the complete approach for the global localization is

1. The robot stops to record a 3D laserscan.
2. The new 3D scan is matched to nearby scans inside the global map.
3. The transformations between them are estimated by ICP and added into a pose graph for optimization.
4. All poses of the sub-scans within the global map are updated with the new optimization result.
5. The new scan together with its optimized pose is added into the global map.
6. Finally, the local localization estimation is adjusted by the global localization.

Once a global localization run is initiated through our state machine, see Sec. 7, the robot is already stopped and begins recording a new 3D laserscan at its current position. Subsequently, the acquired 3D scan has to be matched to the global map. Its current pose as a combination of the last known global pose and the current local pose is used to determine the closest 3D scans inside the map. All these close scans are matched with the new scan. This scanmatching is done with the ICP (Iterative Closest Point) implementation from the *3D Toolkit*[6] which is designed to cope with large 3D point clouds [6].

These relative transformations between the new 3D scan and corresponding past scans as well as all past scan-matching information is transferred into a pose graph for optimization. As an optimization back-end, we use GTSAM – see e.g. [7] for a general introduction. As a result, the optimized pose graph contains a global pose for every scan of the map. This approach allows the robot to improve its map especially after loop closures, i.e. after re-visiting an already known location.

Finally, all past 3D scans inside the map are updated by the optimized poses and the new scan is used for both extending and enhancing the map. Usually, this leads to a correction of the robot's pose estimation. As an example of our global mapping capabilities, a final map of the competition arena is shown in Fig. 5.

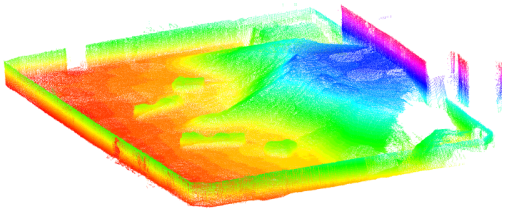

Figure 5 The SpaceBot Camp arena without the dividing wall – mapped with 15 3D laserscans.

[2] imu_filter_madgwick - http://wiki.ros.org/imu_filter_madgwick
[3] robot_pose_ekf - http://wiki.ros.org/robot_pose_ekf
[4] robot_localization - http://wiki.ros.org/robot_localization
[5] 3D laserscanner - https://tu-chemnitz.de/etit/proaut/3dls.html
[6] 3D Toolkit - http://www.threedtk.de

5 Navigation and Traversability

Following our approach from 2013, we split our navigation system into several small and compact modules (see Fig. 4). Like before we use a simple *pilot* to drive the robot from one way point to the next. These points are created by our *path planner*, which uses a terrain evaluation called *danger map* (based on [8]). In 2013 this map was directly created from one sensor, but this time we added a new feature - our *near field map*, which acts as an omnidirectional sensor fusing data from different sensors.

The essence of our *near field map* is a *OctoMap* [9], which is not only a discrete representation (using voxel) of the known space but it also stores its occupancy. For each new measurement the data is not inserted directly, but the occupancy of all related voxel will be updated. Especially made for ranging sensors, this takes both into account: The occupied voxel at the end of each ray and all unoccupied voxels along. Since the occupancy of not updated voxel will stay unchanged, the *OctoMap* represents always the latest possible information for each voxel, e.g. a rotating 2D laser scanner will implicitly create a 3D representation after a while. Another advantage is granted to the updating process: single erroneous measurements can be suppressed and different sensors can be weighted according to their reliability.

Our *near field map* also consists of several filters, which will mutate the sensor data: On the one hand the update rate is reduced, e.g. the frequency of the RGB-D sensor is limited from 30 Hz to about 3 Hz for run-time reasons. And on the other hand the robot with all its parts is removed from the data, therefore it will not recognize itself as an obstacle. A similar filter is set up for the output, mainly to limit the update rate and to limit the data to the region of interest. The output itself is a simple point cloud of all occupied voxels, i.e. the inner occupancy is reduced to a binary state.

This point cloud is directly fed to the second module within the navigation system - our *danger map*, based on [8]. The resulting map is a discrete 2D representation of the traversability, calculated from the terrain slope and the maximum step height in the vicinity of each cell. The maximum traversable slope angle and step size are determined according to the capabilities of the robot's locomotion system.

The next module is our two-level *path planner*. On the top-level is a global planner, which generates just a few meta points for each new target pose, referred to as coarse path. On the bottom-level is the local planner, which steadily creates close-knit way points from the robot's pose to the next meta point(s). The separation is motivated by the idea to have a rough direction while reacting to new or unknown obstacles within a reasonable amount of time.

Finally, the last module is the pilot, which continuously creates driving commands based on the robot's current pose and the next way point.

6 Mobile Manipulation

The mission of the SpaceBot Cup containing the tasks: find, grasp, and transport two objects (battery and mug) to a third one, and assemble all objects there like shown in Fig. 6. Additionally, one of the small items should not be visible from the aerial view. Thus, it must be assumed it is located inside a small cave or beneath a ledge. In respect to mounting the battery pack into the base object or handling the filled mug without loosing its content the robot has to execute tasks with distinct geometrical constraints which are the most challenging ones for an autonomous robot.

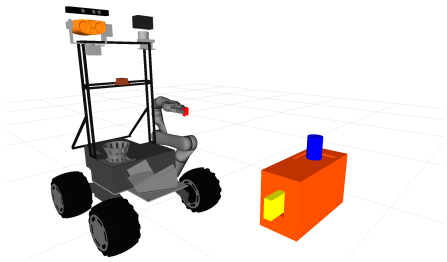

Figure 6 Our rover with the mission objects in RViz.

6.1 Mechanical Approach

Aside from an optional gathering of a soil sample, the task is basically the same as in 2013, when we used a custom-made manipulator. Due to the high amount of maintenance and the lack of reliability we had to replace it with a more suitable model. Fortunately after the competition in 2013, the robotic arm Mico, manufactured by Kinova, became commercially available. Thus, we chose Kinova's Mico as it met our requirements of a lightweight 6-DoF (Degrees of Freedom) arm with sufficient payload. Due to the fact that the scope of Mico is limited we had to find the most efficient mounting position. Therefore, we analyzed and verified the workspace experimentally with a virtual representation of our complete rover. A coarse approximation for the workspace of such a 6-DoF arm is a sphere around its shoulder joint, see [10, p. 72]. In order to move this sphere closer to the ground, the manipulator was tilted forwards. Finally, the off-centered position we chose, allowed a lower mounting point and a better protection against frontal collisions.

6.2 Algorithms and Software

A collision free manipulation in an unknown environment with a moving platform is still a task where no robust out-of-the-box solutions exists and a high amount of custom engineering is needed. However, with *MoveIt!*[7] there is a mighty framework that offers the tools to build such a solution. We used this framework as it provides a huge API around the so called *Move Group*, which acts as an interface between low-level control and high-level planning, and brings access to the *OMPL* (Open Motion Planning Library, [11]). Due to the fact that manipulation is a planning and control problem, good models of the environment, the robot's arm and possible interactions are essential. Therefore, we used the MoveIt! representation of the environment, the robot's arm

[7] MoveIt! – http://moveit.ros.org/

itself and the interaction of both to control the manipulator and plan our tasks. A overview about MoveIt!'s capabilities we used is given in the table 1.

Type of object	Data source	Internal representation
Robot itself	Joint controller	Joint states and URDF
Known objects	Object recognition	Primitive collision objects
Unknown environment	Depth image	Octomap

Table 1 Different ways MoveIt! represents collision objects.

The development of autonomous manipulation software is difficult due to the high risks that real experiments come with. Manipulators like the Kinova Mico are able to damage itself or other parts of the robot like expensive sensors quite easily. So it is crucial to have the possibility of realistic simulations to test the behaviour of planners and state machines. With its good virtual geometric representations and fake joint controllers, MoveIt! supplies an infrastructure to simulate almost everything you can do with the real hardware on a geometrical level. Although the missing Gazebo support of Mico's software stack restrains physical level simulations.

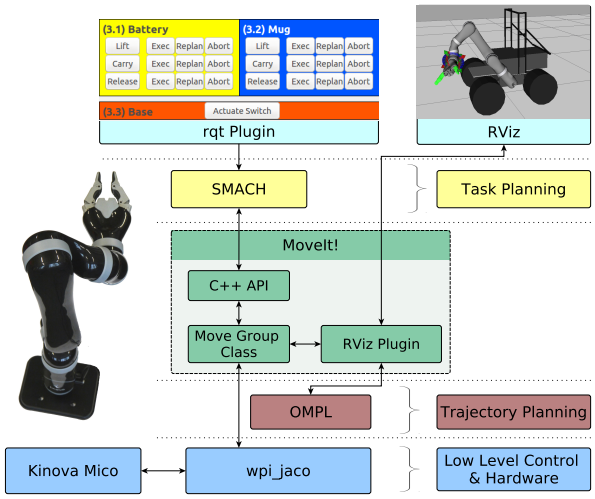

Figure 7 Software structure used to solve the manipulation task.

As Fig. 7 shows, we utilize MoveIt! to calculate paths with the OMPL between the actual position and goal poses results from our higher planning in SMACH or manual commands from our HMI. Goals are generated depending on the position of recognized objects and multiple pre-defined grasping poses for each object. After each planning process, the shortest path calculated between start and goal position is selected. Accordingly, this path will be executed by the *wpi_jaco package* [8], which is a third party ROS driver/controller for the Kinova Jaco/Mico manipulators.

All we have mentioned at this point, is a straight forward implementation of a MoveIt! based autonomous manipulation system which looked very promising but almost twice the time was necessary to make it successfully go through practical tests.

[8] https://github.com/RIVeR-Lab/wpi_jaco

6.3 MoveIt! – a love-hate relationship

Although the MoveIt! framework contains a huge set of functions, like all complex pieces of software, it contains the same amount of annoying bugs. In this section we want to introduce some of the problems we encountered and how we built solutions around them.

Inconsistent robot state – The robot state that is used for planning does not match the current state. Attached objects disappear or trajectories start at the wrong place.[9][10] A possible workaround is to use a *Planning Scene Monitor* instead of the `MoveGroup::getCurrentState()` function of MoveIt! to request the current state.

Collision checking on attached objects – Attached objects are ignored if the corresponding link has no geometry. [11] Therefore, if possible attach it to a link with geometry.

Collada mesh parsing – Some Collada meshes (.dae files) are parsed wrong causing some problems with missing or phantom collisions.[12][13] It is recommended using STL files instead of the Collada ones.

Planning in narrow conditions – Actions like deploying the battery pack out of its mount requires path planning in very tight spaces which seems to be a general problem with MoveIt! and the OMPL. None of the planning algorithms that were available in 2015 were able to find a valid plan in a repeatable manner. Even if a valid plan was found by the planner, the following error message occured: *Motion plan was found but it seems to be invalid (possibly due to postprocessing)*. This happens if the trajectory smoother of MoveIt! smooths the trajectory too much and pushes it into a collision element. To prevent this behaviour you can try to set the `longest_valid_segment_fraction` parameter in the ompl_planning.yaml to a small value e.g. 0.05 but there is no guarantee for success. A more robust solution could be achieved with MoveIt!'s Cartesian paths which move the objects into free space and start the normal planning process afterwards. This simple function generates straight linear paths as long as the manipulator doesn't reach a joint limit.

7 Control Architecture

The control architecture of the robot was implemented as a hierarchical task-level state machine using the SMACH[14] package of ROS. SMACH allows the easy creation of complex hierarchical state machines that support concurrence and provides full access to the underlying ROS message passing concepts such as topics, services and actions.

Programming errors happen all the time, especially if many people are working on the same code and the deadline for the competition is coming closer. Some syntactic checks are done by SMACH on startup, but that merely catches unconnected outcomes of SMACH itself. As SMACH is written in python, a script language, a good think to do is

[9] https://github.com/ros-planning/moveit_ros/issues/583
[10] https://github.com/ros-planning/moveit_ros/issues/442
[11] https://github.com/ros-planning/moveit_core/issues/158
[12] https://github.com/ros-planning/moveit_core/issues/202
[13] http://jonweisz.com/collada-fools-me-twice/
[14] http://wiki.ros.org/smach

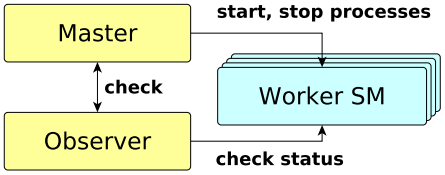

Figure 8 Example of the MultiSMACH concept. The master process starts and stops the worker state machine processes. An Observer instance checks if the worker state machines are behaving as expected. This can mean running watchdog timers or do some sanity check of the behaviour. The observer and master may also check each other to have a failsafe if there is any malfunction within these components.

to run a code checker like *pylint*[15] to catch some anomalies before runtime. Another thing to do: run practical test with your system as soon as you changed anything and try to provoke difficult situations. One challenge we encountered was how to limit the effects of programming errors in our state machine, if they only surface in rare conditions. To increase the overall stability of the control architecture we separated complex state machines into smaller, independent tasks, called *worker state machines*. These state machines run as separate processes and communicate by simple ROS topics, referred to as *trigger topics*. We use this concept to create a more robust software architecture. It can deal with unforeseen exceptions and even programming errors to some degree. If a *worker state machine* should crash the rest of the system can still work. The problematic state machine itself might also work again after an automatic restart, if the problem was caused by very rare conditions. This might not always seem to be a good idea, but for a robot competition a simple «Carry on!» might do the trick.

The design also allows us to monitor each subtask and decide if it is working properly. An observer task might even try to validate the logic and detect anomalies. On a basic level that could mean something like an assertion checking. Consequently, this information can also be used to restart or permanently shutdown problematic or currently unnecessary parts of the control architecture. However during the competition the monitoring part was not implemented. We only had an automatic restart mechanism for individual *worker state machines* implemented.

Another benefit of this design is the easy access to inner connections (*trigger topics*) of the control architecture. During the robot competition, we were able to solve a deadlock situation by sending a, ping-like, trigger command to one of the *worker state machines*.

The communication between our robots and our ground station was realized with two separate networks, for field and ground side and a connection software with the ability to handle the restrictions of the competition (delay, only one separate port for up link and one for down link). The synchronization between field and ground control network was realized with our *serializer* software. For each direction there is a sender and receiver process. The sender subscribes to specified ROS topic of a ROS master on the first network. The messages of these topics are then hashed, compressed, tagged and serialized. The transmission is done with UDP on the specified up link and down link port. The received packages are deserialized, ordered, verified, and announced as ROS messages to a ROS master within the second network. The endpoints of this communication link is an *odroid* mini PC on the field side and one of our ground station PCs on the other side.

For distributing the ROS messages between the ROS masters within the two networks, where port restrictions and high delays are not an issue, we used the *multimaster_fkie*[16] package. This package contains tools to synchronize ROS topics and services between multiple ROS master instances. It is described in [12]. On the field side the ROS master instances of our rovers and the *odroid* are synchronized. Hence all messages send by our ground station are distributed from the *odroid* to our rovers. The same applies for sensor data topics coming from our rovers for the ground station PCs in the other direction.

On the ground control side we used the same setup. All topics meant for our secondary rover were synchronized to our secondary workstation. During normal operation mode each workstation was configured to only process the topics of one of the rovers.

8 Conclusion

Despite our second participation in the SpaceBot competition it was still a challenging task. We made the decision to design our rovers completely new by adding new sensors and replacing the manipulator with a commercial model. In the following, we present some lessons we have learned during the competition and list future steps for the improvement of our complete system.

8.1 Lessons Learned

A good HMI is essential! We improved our HMI concept, as we believed the old one to be to basic for the qualification run. We wanted to be prepared if anything went sideways. Since the time limit for the qualification tasks was quite tough, we wanted quick access to any major capability of the system. We used the *rqt* plugin structure for creating a GUI with all major interaction possibilities. Hence, we implemented direct access to the driving mode, several trigger buttons for our state machines, the camera feeds, and of course log messages from the system.

As driving is a very essential capability of a rover, we wanted to have different modes for that. Starting with a fully autonomous mode, we could step down the autonomy level to manual control, with some semi-autonomous modes in between. We made use of that feature several times during the competition to save time when high precision was not necessary due to the absence of close obstacles. For data visualization we had two *RViz* windows in addition to our command GUI – one for driving and the other for

[15] https://www.pylint.org/

[16] multimaster_fkie - http://wiki.ros.org/multimaster_fkie

manipulation. Each window showed only the necessary sensor data relevant for the task.

Don't ignore the boring low-level basics. Working on low-level tasks like robust communication under the competition's limitations is painful and time consuming but necessary! Everyone talks about the new sensor or improved algorithm that was implemented and probably made a difference. But you can also spend a lot of time to make communication robust if there are constraints in place, like limited bandwidth or high delay. Most importantly make sure you have a test setup to check the limit of your own system and also the system provided by others. Check what happens if you exceed specified limits. Make your test system as similar as possible to the setup you will find during the competition. And do not assume something should behave as specified, always test and verify. The devil is in the details.

Don't underestimate the necessity of continuous maintenance of the software system! ROS packages change over time. Do not expect a working robot after two years of development pause. Do not try to make everything by your own! Use ROS packages wherever possible. This may result in pain, because some packages are stopped to be maintained or have a changed API interface, but on the other hand your own packages are likely to be outdated and incompatible with the current ROS version.

8.2 Future Work

Even if our systems could manage the given task of the SpaceBot Camp 2015, there is much work to be done, especially regarding ROS package integration. Due to historical reasons, we implemented our own path planner and cost map node. Meanwhile the ROS *navigation stack*[17] provides more advanced components like the *costmap_2d* [13] with its plugin structure, the *dwa_local_planner* based on [14], and the *global_planner* with different planning strategies. We have already begun to utilize these packages within our system and got promising results regarding a smoother driving performance.

To enhance robustness of our localization method, we plan to add robust methods to our pose graph solution like *Switchable Constraints* [15] or *Dynamic Covariance Scaling* [16]. Furthermore, it may be reasonable to replace the EKF, used for sensor fusion, with an incremental factor graph based solution like *iSAM2* [17]. Especially in the absence of global measurements, this may be beneficial (see e.g. [18]). In addition, robust methods – like already stated above – can be used.

Due to resources and time constraints, we could not use the possibility of our two robots exchanging environment information as shown in other work like [19]. This would reduce the exploration time to find the mission objects considerably.

9 Literature

[1] N. Sünderhauf, P. Neubert, M. Truschzinski, D. Wunschel, J. Pöschmann, S. Lange, and P. Protzel. Phobos and Deimos on Mars – Two Autonomous Robots for the DLR SpaceBot Cup. In *Proc. of Intl. Symp. on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)*, 2014.

[2] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Intl. Symp. on Robotics Research (ISRR)*, 2011.

[3] A. Geiger, J. Ziegler, and C. Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *Intelligent Vehicles Symp. (IV)*, 2011.

[4] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan. Estimation of IMU and MARG orientation using a gradient descent algorithm. In *Proc. of Intl. Conf. on Rehabilitation Robotics (ICORR)*, 2011.

[5] T. Moore and D. Stouch. A generalized extended kalman filter implementation for the robot operating system. In *Proc. of Intl. Conf. on Intelligent Autonomous Systems (IAS)*, 2014.

[6] J. Elseberg, D. Borrmann, and A. Nüchter. Efficient processing of large 3d point clouds. In *Intl. Symp. on Information, Communication and Automation Technologies (ICAT)*, 2011.

[7] F. Dellaert. Factor graphs and gtsam: A hands-on introduction. 2012.

[8] A. Chilian and H. Hirschmüller. Stereo camera based navigation of mobile robots on rough terrain. In *Proc. of Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[9] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013.

[10] B. Siciliano and O. Khatib, editors. *Springer Handbook of Robotics*. Springer-Verlag Berlin Heidelberg, 2008.

[11] I. A. Şucan, M. Moll, and L. E. Kavraki. *IEEE Robotics & Automation Magazine*.

[12] S. H. Juan and F. H. Cotarelo. Multi-master ros systems. Technical report, Institut de Robòtica i Informàtica Industrial, 2015.

[13] D. V. Lu, D. Hershberger, and W. D. Smart. Layered Costmaps for Context-Sensitive Navigation. In *Proc. of Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2014.

[14] D. Fox, W. Burgard, and S. Thrun. The Dynamic Window Approach to Collision Avoidance. *IEEE Robotics Automation Magazine*, 1997.

[15] N. Sünderhauf and P. Protzel. Switchable Constraints for Robust Pose Graph SLAM. In *Proc. of Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

[16] P. Agarwal, G.D. Tipaldi, L. Spinello, C. Stachniss, and W. Burgard. Robust Map Optimization using Dynamic Covariance Scaling. In *Proc. of Intl. Conf. on Robotics and Automation (ICRA)*, 2013.

[17] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree. *Intl. Journal of Robotics Research*, 2012.

[18] S. Lange, N. Sünderhauf, and P. Protzel. Incremental Smoothing vs. Filtering for Sensor Fusion on an Indoor UAV. In *Proc. of Intl. Conf. on Robotics and Automation (ICRA)*, 2013.

[19] J. Dong, E. Nelson, V. Indelman, N. Michael, and F. Dellaert. Distributed Real-time Cooperative Localization and Mapping using an Uncertainty-Aware Expectation Maximization Approach. In *Proc. of Intl. Conf. on Robotics and Automation (ICRA)*, 2015.

[17] http://wiki.ros.org/navigation