Proceedings of the 12th International Symposium on Artificial Intelligence, Robotics and Automation in Space - i-SAIRAS'14, Montréal, Canada, 17-19 June 2014.

Phobos and Deimos on Mars – Two Autonomous Robots for the DLR SpaceBot Cup

Niko Sünderhauf*, Peer Neubert, Martina Truschzinski,
Daniel Wunschel, Johannes Pöschmann, Sven Lange, and Peter Protzel

Dept. of Electrical Engineering and Information Technology, TU Chemnitz, Germany
e-mail: firstname.lastname@etit.tu-chemnitz.de

* School of Electrical Engineering and Computer Science, Queensland University of Technology, Brisbane
email: niko.suenderhauf@qut.edu.au

Abstract

In 2013, ten teams from German universities and research institutes participated in a national robot competition called SpaceBot Cup organized by the DLR Space Administration. The robots had one hour to autonomously explore and map a challenging Mars-like environment, find, transport, and manipulate two objects, and navigate back to the landing site. Localization without GPS in an unstructured environment was a major issue as was mobile manipulation and very restricted communication. This paper describes our system of two rovers operating on the ground plus a quadrotor UAV simulating an observing orbiting satellite. We relied on ROS (robot operating system) as the software infrastructure and describe the main ROS components utilized in performing the tasks. Despite (or because of) faults, communication loss and breakdowns, it was a valuable experience with many 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 robots for planetary explorations and also for terrestrial applications. A call for proposals was launched in October 2012 and ten German universities and research institutes were selected as participating teams. After the kick-off meeting in March 2013, the ten teams funded with €50,000 each had 8 months to prepare a robotic system for the SpaceBot Cup held in November 2013. This paper describes the required tasks, the technical choices made to build a team of two ground robots and one flying robot, challenges and performance issues, and the lessons learned from the competition.

2 The SpaceBot Cup

To simulate a typical exploration scenario, the DLR used an indoor motocross arena to model a rugged Mars-like planetary surface, including gravel, fine sand, boulders of different size, trenches and small hills of up to 2 m height with slopes up to 30 degrees. Figure 2 gives an impression of the area with a size of 36 × 28 meter. After one day of on-site preparation, there were two days of competition. Each team had a single run of one hour to perform the following tasks: The robots started from a landing site and had to autonomously navigate the terrain (obviously without GPS) on a suitable path. Two objects, a mug filled with water (600g) and a battery pack (800g) were randomly placed on the surface, one hidden under an overhang, thus not visible from above. The robot had to find the two objects, pick them up and transport them to a base object placed on a steep hill. There, the mug had to be placed on top of the base and the battery pack had to be inserted into a recess of the base object. Then the robot had to return to its landing site with possible new obstacles

Figure 1. : Phobos and Deimos on the rugged Mars-like terrain during the SpaceBot Cup competition.

Figure 2. : Space Bot Cup Arena. During the competition the blue ambient light was replaced by more neutral colored spotlight illumination. The red base object is visible in the background on top of the hill.

on its path placed by the jury. The performance measure was the time to complete the mission with penalty/bonus times for certain mission tasks. Beside those measures, the jury had some leeway to subjectively assess the overall performance of the system.

Outside the arena was a container simulating a mission control center from which team members got sensor data from the robot, but could talk to the robot only during three five minute checkpoints. The communication had a simulated delay of 2 seconds in each direction as well as packet losses and performance degradation.

The SpaceBot Cup focused mainly on the autonomy of the systems operating in a realistic terrain. Other conditions like temperature, atmospheric pressure etc. were not considered. Also RGB-D sensors like the Kinect were allowed and usable due to the indoor setting. The use of UAVs simulating observing satellites was also not prohibited.

3 System Overview and Mission Plan

Systems operating autonomously in rough terrain without global localization fail with high probability, to put it mildly. Thus, our plan always was to use two ground robots, if not for cooperation then for redundancy (see section 4.5 for a discussion on this). Additionally, we used an autonomously operating quadrotor UAV for providing aerial images to aid in finding the objects (see section 5 for details). A cubic box at our landing site hosted a WiFi access point for the communication channel between the robots and mission control and served as a start and landing platform for the UAV. AR-Tags on the box aided our ground robots in a localization relative to the landing zone if necessary. We designed the whole system to fulfill the mission completely autonomously. We ran individual ROS[1] cores at each robot and the mission control. No teleoperation or other user control was planned at any time. However, the robots should provide the mission control operators with sensory data to enable them to document the mission state and to interact with the robots in case of emergency. The artificial delay on the network communication rendered standard TCP protocols unusable due to the required acknowledge after each package. To enforce autonomy and prevent tempering, only one port was allowed for communication which made standard ROS communication difficult. Obvious workarounds like VPN tunnels were not allowed. To comply with the contest rules, all communication between the robots and mission control was serialized, tunneled through a single network port and parsed at the opposite end. This also enabled the communication between the different roscores.

The mission plan was implemented in a hierarchical state machine (section 4.4) and uses a navigation system based on a danger map (section 4.2) and modules for fully automated object detection and manipulation (section 4.3).

4 Phobos and Deimos: The Ground Robots

4.1 The Hardware Platform

The mission scenario described above 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[2] as a platform for our own extensive hardware modifications and systems integration. See Fig. 1 for illustration and Table 1 for technical specifications. The skid-steered platforms are capable of carrying up to 20 kg payload. With their independent wheel suspension, the large contoured tires, and the powerful 4×250 W wheel hub drives, these robots proved to be a good choice for the type of terrain encountered during the competition. We were the only team that deployed a team of two ground robots in the competition.

[1] Robot Operating System, http://www.ros.org

[2] http://www.robotnik.es/en/products/mobile-robots/summit-xl

Table 1. : Technical specifications of the two ground robots

Attribute	Value
Dimensions	693 x 626 x 985 mm
Weight	47 Kg
Speed	3 m/s
Batteries	8x3.3V LiFePO4
Traction motors	4x 250 W brushless servomotors
Max. climbing angle	about 45°
CPUs	Intel Core i7-3770 and i5-3330
Price	about €15,000

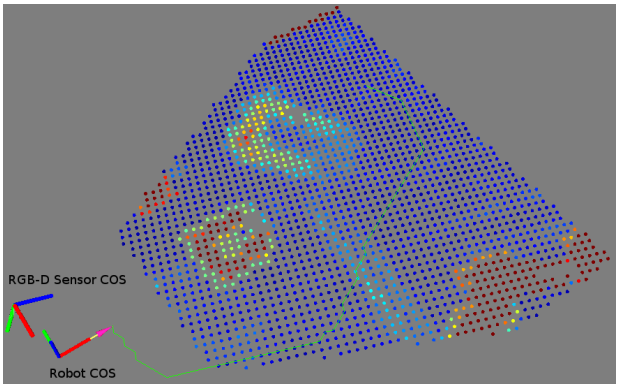

Figure 3. : Danger map with color-coded traversability costs (high costs are shown in red) and the paths planned by the global (visualized in green) and local planners.

4.2 Danger Map based Navigation

The traversability of the terrain was modeled following an approach similar to [3]. From RGB-D data collected by an Asus Xtion sensor mounted on a pan tilt unit, we create a digital elevation model with a grid resolution of 5 cm. The traversability of each cell is estimated based on a *danger value* that is calculated from the terrain slope in the vicinity of each cell and the maximum step height (i.e. the maximum height difference) in this cell. The maximum traversable slope angle and step size is determined according to the capabilities of the robot's locomotion system. The resulting *danger map* could be used directly as a *cost map* for the path planning subsystem.

To plan and follow a path to a goal location, we applied a hierarchical approach using a *global* and a *local* planner. The danger map served as a cost map for both of these planners, which were implemented as simple A* planners. In future work we want to replace these by a more sophisticated Field D* approach. The global planner first created a path from a start pose (typically the current robot pose) to the desired goal pose (e.g. the next exploration waypoint) so that the accumulated costs from the danger map are kept small. The local planner on his part tried to follow this global path and created the necessary motor commands, (forward velocity and turn rates). The local planner could also re-plan the trajectory on a local scale, e.g. when obstacles occurred that were not known to the global planner. This situation typically occurs when the robot enters a previously unknown area during the exploration. In particular, the local planner was called with 2 Hz and selected a *local goal* point on the global path within a maximal distance of 2 meters from the current robot position. It then generated a *local path* to that intermediate goal point.

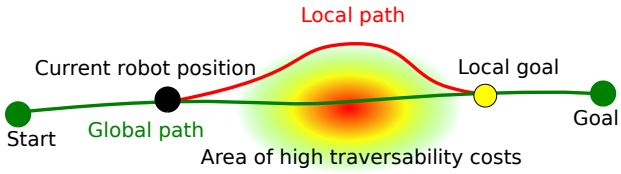

Figure 4. : Two path planners work hand in hand to move the robot towards its goal: First the global planner creates a path based on the current knowledge of the terrain in the danger map. It then commands the local planner to follow this path. While creating motion commands for the motors, the local planner constantly assesses the updated danger map and reacts to obstacles or areas of high traversibility costs. A local path is generated that can locally diverge from the global path but attempts to follow it as closely as possible. A variety of recovery behaviors (see section 4.4) ensures that the robot reaches its goal.

Motion commands v and ω were continuously generated using a simple proportional control strategy that first turned the robot in place towards the next local waypoint and generated a forward velocity only if that waypoint was within a 45 degrees bearing:

$$\begin{aligned} v &= v_{\max} \cdot \cos(2\alpha) && : \cos(\alpha) > \tfrac{\pi}{4} \\ v &= 0 && : \text{otherwise} \\ \omega &= \omega_{\max} && : \cos(\alpha) > 0.5 \\ \omega &= \sin(\alpha) * \omega_{\max} && : \text{otherwise} \end{aligned}$$

where α is the relative bearing from the robot to the next waypoint on the local path.

4.3 Mobile Manipulation

One of the mission goals was to find, grasp, and transport two objects (a battery pack of 800 g and an open mu gfilled with 400 ml water, see Fig. 6) to a third one, and assemble all objects there. Since none of the commercially available arms fulfilled the combined constraints of required payload, maximum size, and price, we decided to design an 6 DOF manipulator arm specifically for our needs. The arm is powered by six Dynamixel servo motors (two MX-106 and four MX-64). Dynamixel servos

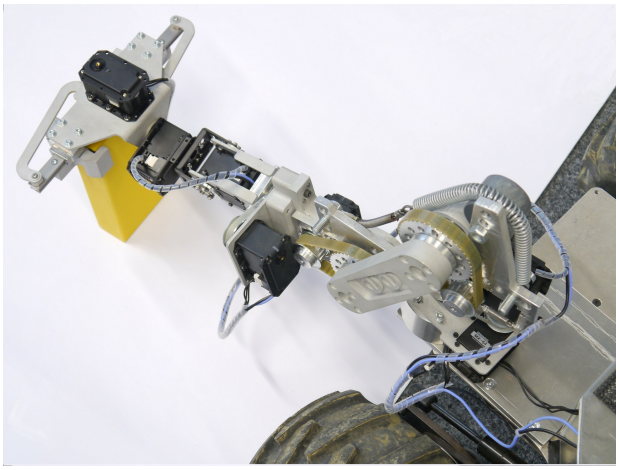

Figure 5. : The custom made 6 DOF manipulator arm with its task-oriented gripper. For transportation of the objects the robots had additional racks to avoid holding the objects during large robot movements and to free the gripper for transportation of multiple objects.

are lightweight, strong and easy to use motors that are fully ROS compatible. They are controlled through the Dynamixel packages[3]. ROS MoveIt![4] is used to calculate the kinematics based on an URDF model of the manipulator. A custom ROS node implements the interface between planned manipulator trajectory (given as individual joint state sequences) and the Dynamixel servo interface. See Fig. 7 for an overview.

The mobile manipulation task is based on a fully automatic detection of the objects and their 6D pose. 3D Geometry and color of the objects were known in advance. The objects do not provide significant texture that could be used for detection. To simplify the object detection task, striking object colors were chosen by the competition organizers. We used an Asus Xtion camera as visual sensor. This sensor had serious problems with color saturation of the non Lambertian object materials in combination with spotlight illumination (as it was the case during the contest). This limitation of the sensory input also restricted the benefits of robust colorspace models like normalized color rgb or $l_1l_2l_3$ [4]: the overexposed yellow battery pack shows to large parts the same white color as the overexposed sand. To overcome this limitation, we quantized the colorspace into classes with different probabilities for belonging to the object to search. An initial (offline) calibration step assigns high probabilities to the spectrum of object colors and additionally lower (but non zero) probabilities to possibly overexposed image regions. Of course, this introduces a lot of false positive indicators for detections when relying only on the object color. To deal

[3]e.g. http://wiki.ros.org/dynamixel_controllers
[4]http://wiki.ros.org/moveit

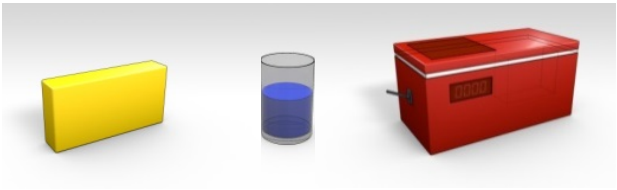

Figure 6. : The three mission related objects. The yellow battery pack and the blue mug had to be searched, picked up and transported to the red base object. The battery pack had to be placed into a slot. The blue mug was filled with water and had to be placed on a scale on top of the base object. After placing both objects, releasing a switch on the base object should indicate the completed manipulation task.

with high false positive rates object detection pipelines typically create a discrete set of hypotheses followed by a hypotheses verification step, e.g. [1]. Such pipelines perform well for detection of the 3D pose of textured or complexly shaped objects in arbitrary poses.

While the objects in the SpaceBot scenario do not provide texture or a sufficient complex shape to detect salient 3D keypoints, their pose in the world is limited to the 2.5 dimensional ground plane. In fact, the objects were known to stand on flat areas with sufficient space to place the robot at the same level (at least from one side) but with possible overhangs over the objects. We formulate the given object detection problem in terms of a template matching in a projective space: Exploiting the knowledge about possible object poses we can reduce the search space for objects to a projective 2D plane given by the robots footprint coordinate systems XY plane. We deduce several projective models for each object from its known 3D geometry. Using an orthogonal projection to the 2D ground plane, we reduce the search space for each projective model to the three dimensional (x, y, Θ) space (2D position and orientation of the model). The number of models for an object depends on the number of projective views that are sufficiently different (keeping in mind that distinction between different orientations and shift are handled by the search space). E.g. for the box shaped battery object, these are typically the three different rectangular sides. Reducing the search space to the three (x, y, Θ) dimensions and the few models, we can apply an exhaustive search over a sufficiently fine grid (e.g. 0.5 cm and 5 degree in the projective plane) of all object poses. We use a normalized correlation measure on the quantized object colors of the current projected view and the projected object model. To reduce false matchings in large overexposed image areas, we surround the projected object model with negative values for the correlation computations.

We use a set of predefined grasping points for han-

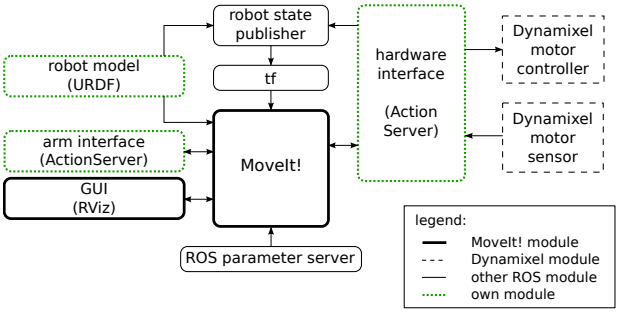

Figure 7. : Overview of MoveIt! integration. MoveIt! computes kinematics based on a URDF model of our custom made arm. Further, we had to provide two ROS ActionServer. The first implements actions to interface the arm (e.g. move the arm to a position). The second ActionServer contains the interface to the hardware (in our case the Dynamixel servomotors) and executes the planned trajectories.

dling the objects. After successful detection of an object, the robot tries to approach the object to reach a position from where it can grasp it. Therefor we use a sequence of predefined approach positions that are input to the GoToPoint behavior presented in the following section. At each point of the sequence we try to verify our observation with the current camera view and start recovery behaviors if the detection gets lost.

4.4 Hierarchical Control Architecture

The control architecture of the robot was implemented as a hierarchical task-level state machine using the SMACH[5] package of ROS. SMACH allows the easy creation of relatively complex hierarchical state machines that support concurrence and provides full access to the underlying ROS message passing concepts such as topics, services and actions. A special focus of our control architecture design lay on the creation of contingency modes and recovery behaviors on all relevant levels. To accomplish this, we had to extend the original SMACH container modules to support the concepts of *timeouts* or *adaptive transitions*.

We used timeouts for all blocking tasks to prevent the robot from getting stuck forever in a state while waiting for the arrival of a message or the occurrence of an event. Timeouts also kept the robots from trying to accomplish a goal that could not be reached despite all recovery behaviors. Such recovery behaviors were activated whenever the robot failed to fulfill a given task (e.g. reach a waypoint, grasp an object).

Adaptive transition nodes ensured that the system could escalate its recovery efforts from a simple retry (e.g. try to grasp again) over a new approach (e.g. move to another position and try to grasp from there) to a complete abort or even future prohibition of a given task (e.g. grasping failed definitively, remember and never attempt to grasp this object again).

Fig. 8 illustrates the general layout of a behavior, using the simple GotoGoal behavior as an example. It is implemented as a SMACH concurrence container and comprises three SMACH states executed in parallel: Two of these states simply wait for an event (timeout or an operator-issued emergency stop) to occur and would then terminate the whole behavior. The core functionality calls the global planner and uses the local planner to follow the returned path to the goal point. If any error occurs, the recovery behavior is triggered. This contains a decision node that remembers how often the recovery was triggered while trying to reach a specific goal point. Depending on this count, different recovery behaviors are then activated, escalating from a simple retry over clearing and rebuilding the danger map to trying to reach the goal in a purely reactive way without using a pre-planned path. If all these fail, the recovery behavior declares the goal unreachable and terminates the GotoGoal behavior with a failure. Higher behaviors in the hierarchy outside this module can then decide how to proceed by triggering their respective recovery behaviors and so on.

4.5 Cooperation

In our initial concept we envisioned two heterogeneous ground robots with very distinct capabilities and tasks. One platform was intended to be small and agile and responsible for the rapid exploration of the environment, including mapping and object identification. The second, bigger platform was planned to carry the custom-made arm and perform the required manipulation and transportation tasks. This concept, although elegant and appealing, relies heavily on the availability of a stable communication channel between both robots. In addition, a dedicated cooperation layer must be introduced in the (now distributed) control structure that coordinates data sharing and task allocation between both robots. Such concepts have been successfully demonstrated in complex robot tasks, e.g. [7], [5] and the cooperative localization and sharing of pose graphs over unreliable and low-bandwidth channels has been researched in [8].

However, after careful considerations we decided to abandon the initial concept of strongly cooperating heterogeneous robots for the following reasons:

1. We assessed the risk of one robot being disabled during the competition (due to sensor failures, algorithmic malfunction or operator error) to be high. For the sake of full redundancy we therefore decided to build two identical robots.
2. We found the support for multi-robot teams is currently underdeveloped in ROS. Even modules for ba-

[5] http://wiki.ros.org/smach

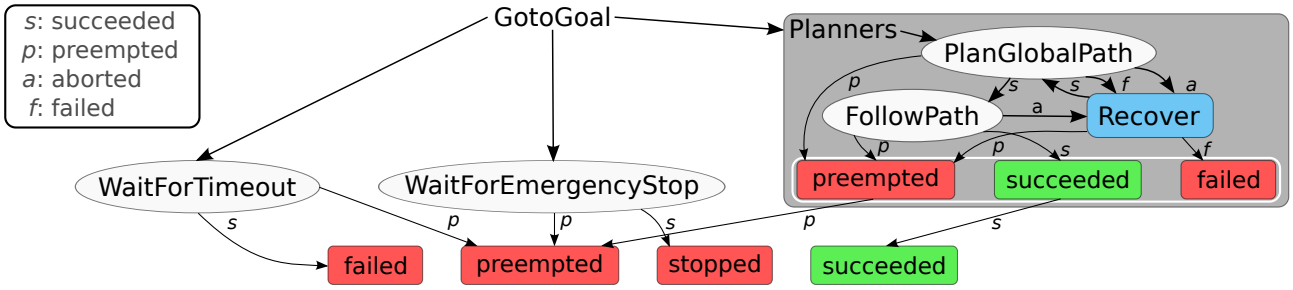

Figure 8. : Simplified view of the goto behavior implemented in SMACH. We use three concurrent containers (WaitForTimeout, WaitForEmergencyStop, and Planners). The first prevents the behavior to get stuck forever, trying to reach an impossible goal. The second captures the event that the operator issues an emergency stop. The third behavior contains the actual functionality and comprises the global and local path planners (PlanGlobalPath, FollowPath) and a complex recovery behavior that is triggered whenever a problem or error occurs in one of the two other states.

sic tasks such as data sharing between ROS cores running on multiple robots were not standardized or assessed to be not reliable and stable enough.

3. The default communication channel between robots (2.4 GHz Wifi) was assessed to be not reliable and stable enough during the competition. This is partly due to the characteristics of the reproduced Marsian surface with hills, ditches and the resulting signal shading, but also due to the high risk of interference with the ubiquitous personal devices such as smart phones and tablets of the spectators in the arena or simply the robots of the other teams. A second communication channel at 866 MHz (e.g. using serial 866 MHz XBee modules) could be a partial solution for future events, although allowing only low bandwidths.
4. The communication channel between the robots and the mission control station was designed to be delayed and expected to be unreliable. Since we wanted to limit operator interactions to emergency situations, an additional layer in the control hierarchy would have been necessary to coordinate both robots. This was beyond our scope for the SpaceBot Cup 2013 but surely is an important direction for future work.

5 The Quadrotor UAV

We also deployed an autonomous UAV (an *AR.Drone 2*, as can be seen in Fig. 9a) to support the robots on the ground with the task of finding the mission critical objects. The *AR.Drone 2* is a commercially available quadrotor system produced for the mass market. It is designed to be used as a toy, but has many features which makes the system suitable for research applications. For example in [6] the system is extended by another on-board embedded PC for achieving autonomous navigation. The system is about 63 cm in diameter with its protecting hull and has an overall weight of 456 g.

Compared to the usage of other quadrotor systems, the AR.Drone benefits from sophisticated flight control algorithms, executed on the onboard PC, which achieve stable hovering with position stabilization in absence of a global measurement system like GPS. An in-depth description regarding these algorithms and other hardware details can be found in [2].

As with the ground robots, the AR.Drone control architecture is based on a SMACH state machine. Due to limited processing power of the on-board PC, the state machine including the ROS framework runs on a separate PC within the box at the landing side and communicates with the quadcopter via Wi-Fi. For interfacing the AR.Drone communication protocol to the ROS framework, we used the ardrone_autonomy[6] package. Basically there are two main states implemented: a manual mode for testing purposes and an automatic mode.

The automatic mode implements two full 360° camera scans on two different altitude levels above the deployment site. While the position hold functionality of the AR.Drone is sophisticated, it is not sufficient for an accurate position hold over the deployment site, as needed for the 360° camera sweep, so we used a special pattern called *oriented roundel* already recognized by the AR.Drone's on-board PC. As soon as recognized, the internal sensor readings respond with information, where the special tag was found within the bottom camera image. These coordinates are given in a range of 0 to 1000 for both image dimensions, regardless of the camera resolution. We used this information for approximating the metric position of the pattern underneath the quadcopter through (1), which is exemplarily given for the x-direction:

$$x = \left(\left(\frac{2 \cdot x_{AR}}{1000} - 1\right) \cdot \tan\left(\frac{FoV}{2}\right) - \arctan\phi\right) \cdot h \qquad (1)$$

[6] http://wiki.ros.org/ardrone_autonomy

(a) (b) (c)

Figure 9. : (a) The AR.Drone 2 as we used it during the competition. (b) Live image from the quadcopter while hovering over the deployment site. (c) Magnified image part which shows one special target – the blue spot.

where x_{AR} is the tag's coordinate and h is the quadrotor's height above ground in metre, detected by its sonar sensor. Additionally, FoV stands for the bottom camera's field of view which is about 49°×35° and ϕ is the absolute roll angle of the quadrotor. Based on the calculated metric position information, a simple PD-Controller is used for position hold. Due to the latency induced by the Wi-Fi connection, the performance is not the best, but is sufficient for our needs.

By using the described flight functionality, we could get an initial clue where to search for the mission critical objects. In order to achieve this, we had to tilt the front camera downwards about 30° to cover more of the competition area and modify the AR.Drone's ROS interface to get the maximum image resolution. An example of the aerial view is shown in image 9b. As can be seen, the image quality is badly reduced. This happens automatically if the Wi-Fi connection is not optimal.

After marking an object within the images, several information and assumptions could be used to guess the final position of this object:

- The yaw orientation of the quadrotor $\mathcal{B}$ relative to the special tag at the deployment site results in the rotation matrix $\mathbf{R}^{\mathcal{WB}}$, where $\mathcal{W}$ marks a world frame originated in the tag.
- The current altitude h of the quadrotor results in the translation $\mathbf{t}_{\mathcal{B}}^{\mathcal{W}}$ between quadrotor and world frame.
- We make the assumption that the area is a plane.
- We make use of the intrinsic camera parameters $\mathbf{K}$ and the extrinsic calibration between the quadcopter and the camera $\mathbf{R}^{\mathcal{BC}}$ – composed of the downward tilt angle.

By using the assumptions made, we can calculate the direction vector for the homogeneous image coordinate $\tilde{\mathbf{u}}$ and rotate it into the world frame, as shown in (2). Afterwards we can calculate the intersection point $\mathbf{p}_{\text{obj}}^{\mathcal{W}}$ of the line-plane intersection between the direction vector $\mathbf{p}^{\mathcal{W}}$ and the x-y-plane as shown in (3).

$$\mathbf{p}^{\mathcal{W}} = \mathbf{R}^{\mathcal{WB}} \cdot \mathbf{R}^{\mathcal{BC}} \cdot \mathbf{K}^{-1} \cdot \tilde{\mathbf{u}} \tag{2}$$

$$\mathbf{p}_{\text{obj}}^{\mathcal{W}} = \mathbf{t}_{\mathcal{B}}^{\mathcal{W}} - \frac{h}{\mathbf{p}_z^{\mathcal{W}}} \cdot \mathbf{p}^{\mathcal{W}} \tag{3}$$

Notice that this is an approximation, but good enough for an initial guess of the objects' positions.

6 Lessons Learned

6.1 A rugged robot is good, but worthless when blind or paralyzed.

Sometimes it is that simple: Rugged mechanics facilitate autonomy. This became evident for autonomous navigation of our ground robots in comparison to the much more fragile constructions of other contestants.

For existing active 3D sensors there are known (and possibly unknown) conditions under which they do not work properly, e.g. transparent materials, light absorbing materials, or intense back light. We relied on a single type of sensor (Asus Xtion) for obstacle avoidance. This probably caused a collision of one robot with an missed obstacle followed by an accidental release of the emergency stop and an interruption of the motor power.

6.2 Do not rely on a working communication.

As it turned out, our initial apprehensions about the unreliable communication between the robots and the operators in the mission control station proved to be true during the competition. All teams experienced massive problems when trying to send commands to their robot or receive data from it. This lead to the very unpleasant situation that some the participating teams could not even send a *start* command to their robot.

In anticipation of this situation we applied our strategy of contingency behaviors even to the seemingly simple task of starting the robot: The robot would start executing its mission plan after a) it received a start command

from the operators or b) two minutes after an external button on the robot was pressed by the field crew that carried the robot into the arena, or c) 10 minutes after booting the robot's main computer. This threefold safety ensured that all of our three robots did successfully begin to execute their mission plans, in contrast to some other robots that remained motionless at the deployment site. Since we did not have any data connection to the ground robots until approximately 10 minutes into the mission, the robots were started by the 2 minutes watchdog timeout.

Unfortunately, the faulty communication prevented us from sending a crucial command to the second ground robot Deimos after Phobos was disabled due to a collision with an obstacle later in the mission run. Deimos was meant to remain in a waiting position to not interfere with Phobos while Phobos executed the mission plan of exploring, searching for objects etc. We planned to activate Deimos whenever we felt that Phobos experienced difficulties or had eventually moved far enough away from the deployment site. This decision proved to be unfortunate in hindsight. The activation of Deimos relied completely on the communication with the mission control crew. Because this introduced a single point of failure, Deimos could not contribute to the success of the mission.

We feel our assessment of keeping the human operators out of the loop as much as possible was confirmed by the experiences made during the SpaceBot Cup. At no point in the mission should a robot be solely dependent on a human operator to take control or even only send a simple start command.

6.3 Anticipate the ROS effect.

We coined the informal term *ROS effect* to describe our mixed feelings about using and being dependent on ROS for a long-term project on a non-PR2 robot. ROS is in our eyes still the best middleware, ecosystem and algorithm collection for mobile robotics available. It is, however, not a production-ready and stable framework. The diffuse feeling that "there must already be a working module in ROS" that implements a certain task the robot has to accomplish, often gives way to disappointment as the module is not maintained anymore (the PhD student graduated a year ago), does therefore not compile anymore, is not, badly, or incorrectly documented, or breaks after dependent modules are updated over night. Unfortunately this was a re-occurring pattern during the design and implementation phase for the SpaceBot Cup.

Unfortunately we cannot claim to have contributed to improving this situation since we did not release our code to the community as well-maintained modules for ROS. We acknowledge the effort it takes to create usable open source software modules and the difficulties to maintain them over a long time.

7 Conclusions

We have seen our systems working well in principle. They worked (mostly) in the lab and (sometimes) in the competition. The problem is robustness. The hardware of the rovers was fine for this purpose, sensors can be improved with current technology (better cameras, laser scanners), ROS has some major software engineering issues, but is not the bottleneck. One of the main hurdles in achieving more robust autonomy is poor exception handling. Handling exceptions on all levels of the control architecture via recovery behaviors and contingency modes is the key to a robust and truly autonomous long-term operation of any robot. While the actual mission plan is often rather straight forward, designing and implementing these exception handling routines in the control architecture is very tedious. SMACH in its current form (even with our extensions) is not well suited to control the behavior of a robot in a complex environment with many possible error sources. Another way to increase overall robustness is to use multiple robots that cooperate in good times but are able to work independently if other robots or communications fail. Here too, the key is exception handling and improved situational awareness, and we will concentrate on these issues in future work. The SpaceBot Cup -as all competitions with a firm deadline- boosted our productivity and we hope to present some new ideas at a possible SpaceBot Cup 2015.

References

[1] Aitor Aldoma, Federico Tombari, Johann Prankl, Andreas Richtsfeld, Luigi di Stefano, and Markus Vincze. Multimodal cue integration through hypotheses verification for rgb-d object recognition and 6dof pose estimation. In *ICRA*, pages 2104–2111. IEEE, 2013.

[2] Pierre-Jean Bristeau, Françis Callou, David Vissiève, and Nicolas Petit. The Navigation and Control Technology Inside the AR.Drone Micro UAV. In *18th IFAC World Congress*, pages 1477–1484, Milano, Italy, 2011.

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

[4] T. Gevers and A. Smeulders. Color based object recognition. *Pattern Recognition*, 32:453–464, 1997.

[5] John G. Rogers III, Carlos Nieto-Granda, and Henrik I. Christensen. Coordination strategies for multi-robot exploration and mapping. In *ISER*, pages 231 – 243, 2012.

[6] Jacobo Jiménez Lugo and Andreas Zell. Framework for Autonomous On-board Navigation with the AR.Drone. *Journal of Intelligent and Robotic Systems*, 73(1 – 4):401 – 412, 2014.

[7] Edwin Olson, Johannes Strom, Ryan Morton, Andrew Richardson, Pradeep Ranganathan, Robert Goeddel, Mihai Bulic, Jacob Crossman, and Bob Marinier. Progress towards multi-robot reconnaissance and the MAGIC 2010 competition. *Journal of Field Robotics*, 29(5):762–792, September 2012.

[8] Jeffrey M. Walls and Ryan M. Eustice. An exact decentralized cooperative navigation algorithm for acoustically networked underwater vehicles with robustness to faulty communication: Theory and experiment. In *Proceedings of the Robotics: Science & Systems Conference*, Berlin, Germany, June 2013.