The Artemis Rover as an Example for Model Based Engineering in Space Robotics

Jakob Schwendner[1] and Thomas M. Roehr[1] and Stefan Haase[1] and Malte Wirkus[1] and Marc Manz[1] and Sascha Arnold[1] and Janosch Machowinski[1]

***Abstract*— Future application of robotic missions in the space context will require the systems to have both mobility and manipulation capabilities. The limited direct communication with the systems due to visibility, and severe time delays also make it a requirement for the system to perform its actions mainly autonomously. The increasing complexity of the task, as well as the strict requirements for reliability and fault tolerance pose a significant challenge to both engineering and research activities. The SpaceBot Cup was held in November 2013 to probe those capabilities in the context of a competition. In this paper we present the Artemis rover and its software architecture as well as the competition results and lessons learned. Special attention is given to the modular design based on the Robot Construction Kit (Rock) framework – a component based software framework, which uses a component model based on the Orocos Real-Time-Toolkit (RTT).**

I. INTRODUCTION

Since the days crewed space exploration was at its pinnacle with the Apollo program, robotic systems have replaced humans as the agents for the exploration of our solar system. The main advantage of these systems is that they are much more applicable with regards to the type of environment they can sustain. Even though artificial intelligence development has come a long way since the days of the Lunokhod [1], which was purely remote controlled, the latest systems to explore planets [2] or the return to the lunar surface [3] are still limited in what they can do on their own. In general it is preferable to have a human in the loop. It will be a fair while – if ever – until computers will be able to outmatch humans in coping especially with unforeseen situations. A difficult communication environment with delays and only limited connection windows pose limitations to the mission design. Advancing the level of autonomy of space systems improves the options available for difficult missions – usually the ones that are scientifically most interesting [4], [5], [6].

One way of supporting the advance of technologies is to use competitions. These type of events are suitable to foster creative ways of solving current problems, and generating new questions and engineering challenges [7]. One such competition was held by the German Space Agency (DLR) in Rheinbreitbach, Germany in 2013 [8]. The challenge of the SpaceBot Cup was to develop an autonomous mobile manipulation system within 8 months, and then show its capabilities in a 1 h mission. The task was to find and collect two objects in an unknown 21 m by 21.5 m area, and transport them to a third location and then return to the origin. Communication with the robot was severely limited, with a 2 s communication delay and multiple scheduled communication outages.

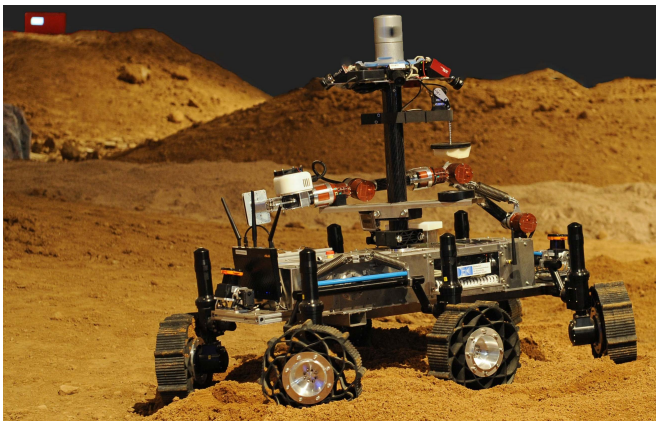


Fig. 1. The Artemis rover – shown at the SpaceBot Cup competition area – is a six wheeled system with a mass of 87 kg (including lander setup) and a size of 830 mm x 1300 mm x 500 mm (width x length x height).

In this paper we describe the design process of Artemis (see Figure 1) to fulfill the functional requirements of hardware and software in order to perform the full mission scenario of the SpaceBot Cup.

In a first step the functional decomposition for each of the tasks was detailed. Subsequently, this decomposition allowed to distribute the work and to assign it to corresponding system experts in the fields of mechanical, electronic and software engineering. A fundamental requirement for the mission was autonomous navigation. In order to achieve this capability a further decomposition was performed into the following functional modules: 1) locomotion, 2) mapping, 3) and navigation comprising path planning and trajectory following. Since a coarse map was provided before the competition an exploration module has been considered optional – forwarding an a-priori list of waypoints for exploration was a valid approach that satisfied the needs of the mission scenario.

A critical functional element was the management of the autonomous activities, i.e. the integration layer for all functional components that were needed to fulfil the mission requirements. This management of Artemis is performed by the supervision [9] – a component dedicated to manage the activities of Artemis based on previously modelled high-level functionality which relies on a set of functional single-

[1]German Research Center for Artificial Intelligence, DFKI Bremen, Robotics Innovation Center (RIC), Germany firstname.lastname@dfki.de

purpose modules.

This paper presents Artemis as one approach of solving the complexity of the SpaceBot Cup scenario both on a hardware and software level. In addition to the description of Artemis and its design process, we describe the competitions results and provide a selection of the lessons learnt throughout the development and the competition.

II. SYSTEM DESCRIPTION

The SpaceBot Cup scenario requests capabilities in multiple fields of robotics, which led to three parallel lines of development: 1) navigation – the ability to get to a specific location 2) manipulation – the ability to manipulate objects 3) exploration and object detection – finding the target locations for navigation and manipulation. Each of these capabilities depends on development of hardware and software, while eventually both need to be managed by the supervision component. The following sections describe the approaches for each of the three development lines:

A. HARDWARE

The first main design driver for Artemis were the expected terrain characteristics with slopes up to 30°, loose surfaces with a variety of sand and stone fields. The second design driver was the required manipulation capability. The SpaceBot Cup participants had to find, identify and manipulate three different objects. The strategy of the Artemis team was to collect both mobile objects and directly transport them to the stationary third object. To cope with the demands the hardware development targeted a highly mobile platform to enable the system to traverse each part of the contest area. Additionally, the platform was equipped with a six degree of freedom manipulator and several sensors (Figure 2). In order to implement these capabilities with only one arm, the system was further equipped with storage devices for the cup and the battery.

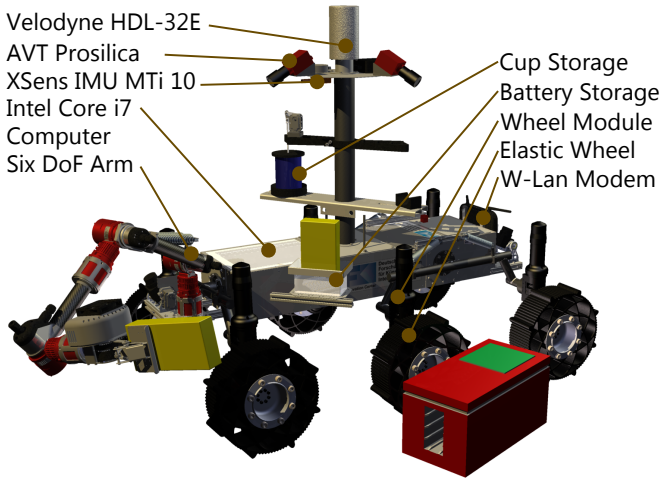


Fig. 2. The Rover with all subsystems (CAD drawing).

Passive suspension: Artemis locomotion platform consists of three single rockers, each is equipped with two fully actuated wheel modules. This enables the rover to drive in every direction from within any orientation and using fully actuated wheels facilitates the manipulation of objects with a six degree of freedom manipulator whilst retaining the possibilities offered by a seven degree of freedom manipulator. The chosen suspension is based on the 3-Bogie design which was proposed for the ExoMars rover [10]. This concept allows for a relatively light weight as well as for higher static stability compared with other concepts like the CRAB, RCL-E, and the Rocker-Bogie [11]. High static stability was a crucial precondition for the placement of a sensor mast at the top of the system. The final system reaches a static stability of $> 45°$ in each direction.

Sensors: The sensor setup for the system is geared towards the individual requirements of the different software modules. The odometry requires the wheel encoder readings as well as an AHRS. Local obstacle avoidance is performed using a tilting laser range finder unit located before the front axis of the system to reduce sensor shadowing. The same unit is used – together with a small camera for color information – for the near range object identification and positioning system. The rotating laser range finder located on the mast of the system is used for the mapping subsystem. The three cameras which are also located on the mast are used for the long range identification of the objects. See Figure 2 for the exact sensor placements and models.

B. SOFTWARE COMPONENTS & SUPERVISION

The software development approach for Artemis has been model-based and component-based. Components in Rock are so-called oroGen components, i.e. Orocos components which have been generated from a specification file which describes the component's interface. The specification mainly defines input and output ports, operations and configuration parameters of a task. Based on this specification a binary - a so-called deployment - can be composed using various task models. This strategy is useful in multiple ways. Firstly, it speeds up the development process since framework specific code is automatically generated and a code skeleton is provided that allows to easily embed functionality that resides in framework independent libraries. Secondly, components are designed modularly and for reuse, e.g., a component for retrieving images from a camera or a path planning component can be easily used in different contexts.

Eventually, since each component comes with an explicit specification further modeling strategies can be applied. The previously mentioned supervision component allows to create high-level functionality using composition of multiple components. This additional modeling involves defining dependencies and data connections between components. Hence, compositions represent subnetworks of components and the supervision can manage these subnetwork during runtime to minimize side-effects of having subnetworks run in parallel and to optimize resource usage. Furthermore, the supervision also allows to perform model based vali-

dation of connections and can automatically compute the size of connection buffers using information given on a components update frequency. The supervision provides an abstract modeling layer and thus does not only apply to oroGen models, but also for ROS Nodes. ROS does not explicitly provide a specification for nodes but since ROS uses well defined interfaces the specifications have been extracted from existing nodes. Having interface specifications for both component types allowed to manage oroGen and ROS Nodes in parallel in the supervision.

C. NAVIGATION

The capability to accurately perform localization and mapping is crucial for a good performance in navigation. To model the environment and localize the robot, a pose-graph based approach for Simultaneous Localization And Mapping (SLAM) is used. As graph SLAM back-end we rely on the g2o graph optimization framework [12].

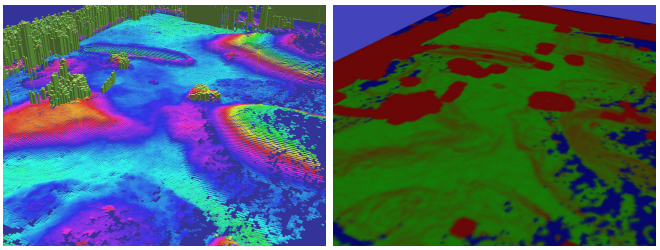
Since we do not model environment features separately, the graph only consists of pose vertices and edges representing constraints between poses. Each pose is associated with a full static 360° laser scan of the environment. New poses will be added depending on the euclidean distance the robot travels from its last known pose in the graph. Possible sensor movements during scan acquisition are corrected using odometry based transformation, so that a static 360° scan can be created for a single robot pose. Due to the high speed of the Velodyne laser scanner (10 Hz) the odometry error is acceptably small.

The transformations between the vertices, represented by edges, are optimized by an Iterative Closest Point (ICP) algorithm, using the known odometry based transformation as a starting point. In particular we use Generalized-ICP (GICP) [13], which has proven to perform quite well on 3D lidar data. After adding a new vertex additional edge candidates to the existing vertices are identified, depending on a maximum euclidean distance (d_{max}). The edge candidates are prioritized and processed continuously from the top of the list. If GICP produces a valid solution for a candidate edge this edge will be added to the graph. To achieve a sufficiently connected graph d_{max} should be at least three times the size of d_{min}, in which d_{min} is the euclidean distance between consecutive poses. The size of d_{max} on the other hand is limited by the ability of GICP to perform valid matches in time.

The global graph optimization is executed every time a predefined number of new edges has been added to the graph. Through the strongly connected graph it is possible to reduce the impact of poor ICP alignments. Adding an outlier detecting approach like [14] or [15] can reduce this impact even more. To limit the memory consumption, older laser scans are deleted on basis of a 2D grid in which the scans are indexed based on the position where they were taken. Deleting older entries in an index cell limits the total number of scans, but assures sufficient number of scans in less covered areas.

As a runtime safety feature it is also possible to continue the SLAM from the latest known location. The current state of the graph is stored at shutdown of the SLAM module and loaded again on startup. If the robot has only moved slightly ($< d_{\text{min}}$), while the SLAM module was stopped, it is possible to continue mapping without losing information or increasing pose uncertainty.

As an additional benefit the SLAM module is real-time capable and can operate nearly independently of the environment type. However, it cannot be applied to strongly ambiguous environments where the ICP algorithm tends to fail. To integrate well with the overall navigation an abstract map out of the aligned point clouds is generated on demand, e.g., when the global planner needs a new map, though this cannot be done in real-time. A result from the mapping module created during the competition run is shown in Figure 3. The generated map uses an extended version of the multi-level surface map (MLS) [16], [17] for representation.



(a) Multi-level surface map with colorized z-height.

(b) Traversability map, increasing costs are visualized by a color changing from green to red.

Fig. 3. Maps of the SpaceBot Cup competition area.

To effectively use the generated maps for navigation, we use a hierarchical approach for path planning, i.e. the developed path planning consists of a global planner using D* Lite [18] and a local planner using VFH* [19].

The global planner creates the shortest trajectory from start to goal avoiding high slopes and obstacles. It is a grid based planner that works on traversability maps, which are generated from MLS maps by dividing the cells into cost-classes according to their slopes. Steeper slopes correnspond to higher costs and slopes with an inclination above 31 ° are regarded as obstacles. In the global content the dimensions of the robot is assumed to be a circle. Therefore obstacle growing can be applied to provide a computational fast way to implement safety areas around non-traversable regions. For performance reasons the movement model of the robot is also simplified. It is assumed that the robot can move directly into all eight neighbor cells, next to cell representing the robot's position, at uniform movement cost. Although the global planner should be triggered at a low frequency, the D* Lite algorithm is used for fast and efficient replanning.

The local planner is designed to loosely follow the trajectory generated by the global planner. In contrast to the global planner it is not grid based, uses a complex motion model and takes the shape of the robot into account. To be able to perform the trajectory generation in real time, the

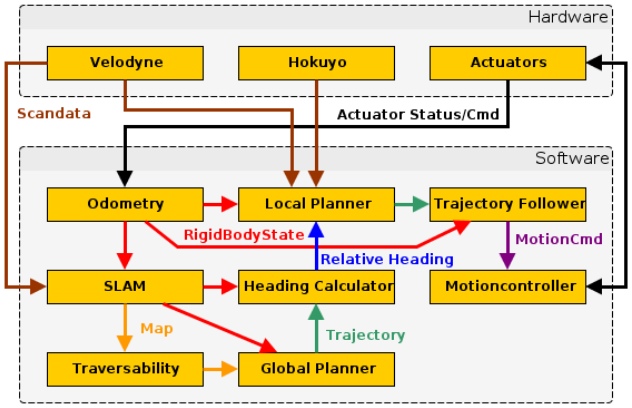


Fig. 4. Modular architecture of the navigation stack.

local planner only computes a trajectory to a horizon that is perpendicular to the vector from the robot to a target location on the global trajectory. The target location is computed by finding the nearest location to the robot on the global trajectory and advancing a certain distance on it. Additional speedup is achieved by only computing the traversability map for the local surrounding of the robot and by using the VFH algorithm to reduce the sample space of the local planner.

To avoid oscillations during the trajectory execution, caused by a 'jumpy' position provider, the local trajectory is transformed into the odometry coordinate system (OCS). An additional benefit this approach allows for a higher update rates of the local trajectory, which results in a smoother movement for the trajectory following.

The global planner is only run if the SLAM map changed, the user changed the goal position, or if the local planner failed. The failure of the local planner is a nominal case, as the global planner may plan through unknown terrain, which might turn out to be non traversable. In this case the global planner is rerun and should change the global trajectory in a way that it can be followed by the local planner. It should be noted, that both local and global planner have to be properly configured to avoid lockouts (see Section IV). The local planner runs almost continuously as it is triggered by position changes of the robot and new sensor input to the local traversability map. If it is triggered, the horizon is determined from the global trajectory and subsequently the local motion planning is performed. In contrast to the global planner the local planner treats unknown areas as obstacles. In the case that repeatedly no motion towards the horizon could be performed the local planner reports a failure to the global planner. Figure 4 shows the integration of all components of the navigation stack.

D. EXPLORATION & OBJECT-DETECTION

The strategy for solving the SpaceBot Cup challenge was to use different modules for the identification of objects from a distance, and an extraction of the objects full pose in the near field. The rational for this is that searching for items is performed more effectively in the visual domain, while extracting a precise pose for manipulation is better solved using 3D localization methods. The object detection method uses the images from the mast camera on which a blob extraction algorithm is applied. Regions with colors similar to the target color are marked as candidates. After simple consistency checking of the blobs using their apparent size, the candidates are projected into a grid map of the environment. Each occurrence of candidate leads to incrementing a candidate related counter in the cell. The cell with the highest count is used as the most likely position of the object.

Once the object is located, the 3D-pointclouds from the front laser scanner are used to find the pose of the object in the scene. This is done by looking for parts of the shape, e.g. planes or curved surfaces and then rate how these elements could be part of the wanted object. The extracted pose is then forwarded to the manipulation subsystem.

E. MANIPULATION

The SpaceBot Cup scenario requires manipulation skills to handle two types of objects: a block-like battery of 1 kg and cylindrical drinking cup holding approx. 0.2 kg of distilled water. The objects have to be grasped and put into their respective stow position on the rover. At the final location the objects need to be retrieved from their storage position respectively and assembled with the so-called base object – a custom-made scale. Figure 2 shows the battery in the manipulator hand and the cup in its stow positions.

We decomposed the different manipulation tasks into five basic abilities: *a*) motion planning to Cartesian and joint space goals, *b*) following joint trajectories, *c*) move the end-effector towards an attractor pose in Cartesian space, and *d*) execution of different grasps suitable for the different objects involved in the task.

For each of those abilities we separately developed components in Rock, mainly by integrating functionalities from existing software libraries, e.g. such as Reflexxes[20] for developing a trajectory controller.

In order to pick or place an object, a motionplanner has to compute a path which is free from self-collisions and collisions with its environment. For the motion planning task MoveIt! [21] software is used, which runs on the ROS framework. The motionplanner uses the current state of the robot and the environment information for generating the collision free path. These informations are given to the motionplanner node from Rock components, since the supervision component can manage ROS nodes and Rock components. Figure 5 shows the software components in ROS and in Rock in use on Artemis.

The gripper component provides the interface to two different grasps types, a flat and a spherical grasp for round objects. A grasp is defined by its type, the hand opening diameter and a reference force to be applied to the object. By observing motor torques and thresholding them, we detect successful grasps.

We implemented a Cartesian control component that uses the weighted damped least squares (WDLS) inverse kinematics solver of the KDL library [22] to generate joint motion

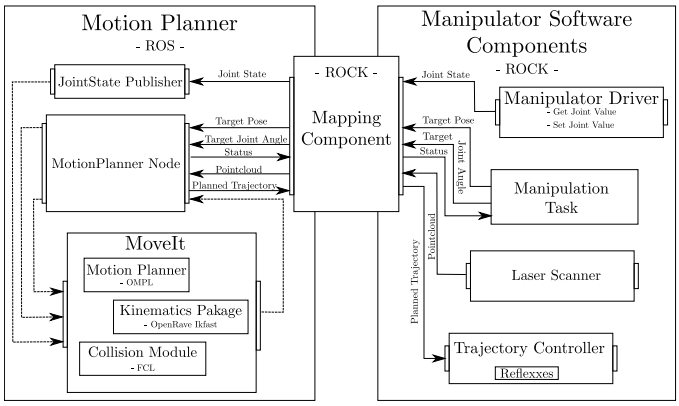


Fig. 5. Motionplanning as complementary activity of ROS nodes and Rock components

commands towards a Cartesian target.

Based on these components, we implemented the core abilities as parametrizable actions. As a bridge between the data-driven component networks and actions we need a mapping of the component network state (defined by the data that is on the network's ports) to discrete events. In addition, we require to influence system behavior at a given moment in time. To achieve this, we used the following synchronization primitives: *a*) *port writers* trigger a behaviour of the component network by writing data to a specific port, *b*) *port readers* read from a port of component network and store data for later reuse in a state-overlapping memory, and *c*) *monitors* trigger an event when a configurable condition, which is a description of data on the ports of a component network, is fulfilled.

The discrete action for Cartesian control "move_arm_cart" serves as example to illustrate the application of the synchronization primitives. This action takes the goal pose, tolerance boundaries, and a validation time as arguments. It instantiates the corresponding component network and triggers the desired behavior by writing the goal pose to the setpoint port of the Cartesian control component. In order to determine whether the desired pose is reached a monitor is attached to the control error port of the same component. It emits a success event when the absolute control error is inside the given tolerance boundaries for the given validation time. The algorithm used for Cartesian control can get stuck in local minima and a monitor is applied to observe the joint positions and detect this situation during execution. A failure event is triggered if joint positions do not change significantly for some time.

Artemis manipulation strategy builds upon fixed, taught-in movements wherever possible. To do so, we defined a home configuration – i.e. a joint configuration that is used as start- and/or end-point for most of our actions. Picking up objects in our approach is a sequence of approaching the object, preparing the grasp and after moving further towards the object closing the grippers. Similarly, the assembly of objects is performed as a sequence of planned movements to two target poses defined relative to the base object. Table I gives an example of how manipulation tasks have been created for handling the battery.

Name of action	Dependencies	Explanation
Primitive actions		
exec_arm_traj	–	Execute a given trajectory
move_arm_cart	–	Move to a given target pose using Cartesian controller
move_arm_cart_p	exec_arm_traj, move_arm_cart	Plan and execute a trajectory to a given goal pose. Optionally use move_arm_cart if planning fails.
Planned movements with pre-defined goals		
move_home	move_arm_jnt_p	Plan and execute movement to home configuration
Tought in movements		
manip_store_arm	exec_arm_traj	Move arm to its store pose
manip_unstore_arm	exec_arm_traj	Move arm from store pose to home
gripper_open	–	Open the gripper
Complex operations		
execute_grasp	move_arm_jnt_p, gripper_open, move_arm_cart_p, gripper_grasp, move_home	Pre-grasp and grasp pose are given as arguments. Move arm to a pre-grasp pose, open gripper, move to grasp pose, grasp and lift object.
move_bat_to_store	move_home, exec_arm_traj, gripper_open	Transports object in hand to battery holder using a pre-defined trajectory
unstore_bat	exec_arm_traj, gripper_grasp	Remove object from battery holder using a predefined trajectory
Top-level operations		
store_bat	move_home, execute_grasp, move_bat_to_store	Grasp and store battery
assemble_bat	move_home, unstore_bat, move_arm_cart_p, gripper_open	Unstore battery and insert into base object.

TABLE I

HIERARCHY OF ACTIONS TO MODEL MANIPULATION TASKS FOR HANDLING THE BATTERY

III. COMPETITION RESULTS

The SpaceBot Cup was held over two consecutive days and the teams where given one day before to prepare at the given competition location and test their communication infrastructure. Teams where not allowed to test their robot in the competition site before the competition, but where bound to test on a small scale test-bed nearby. In the final competition each team got a time-slot of one hour to perform the full mission. Each team had three checkpoints where communication with the system was available for 5 minutes with a 4 s round-trip latency.

No team was able to complete the full mission, and the jury decided to not announce a winner. However, Artemis demonstrated outstanding locomotion capabilities. Though manipulation and object detection had also been prepared, we focus on locomotion and navigation in this result section. Since the exploration site was roughly known through low-resolution maps provided by the organizers, an exploration strategy had been predefined. The main strategy was to explore the site and meanwhile use the robot's and operators' object detection capabilities to locate the target objects. A coarse waypoint sequence was given to the robot in order to perform exploration and after reaching a waypoint the robot tried to advance to the next waypoint. Artemis started off by autonomously traversing large parts of the exploration

Fig. 6. Artemis negotiating an obstacle while navigating autonomously.

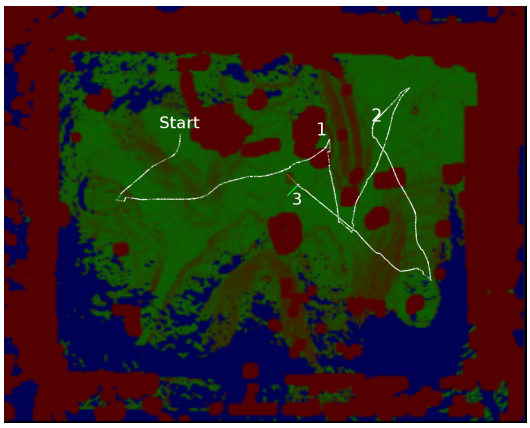


Fig. 7. Traversability map built during the run, with the travelled path during competition: (1) location of getting first time stuck, (2) location of getting a second time stuck and official end, and (3) end of autonomous navigation after exceeding the official competition time.

site (also cf. [23]). The site consisted of sandy terrain with rocky sections. Artemis easily overcame a section of loose soil (cf. Figure 1) where the flexible wheels showed their advantage. Subsequently, Artemis moved over a stone with a size of about two thirds of a wheel's diameter (cf. Figure 6) – compensated by the passive locomotion platform. Finally, Artemis reached a trench. While the global planner computed a plan which led through the trench, the local planner did not allow traversing of the trench and eventually Artemis became stuck since the global planner did repeatedly lead the robot into a situation where the local planner prevented further movement.

Previous to the competition Artemis had been tested to climb up to 35° of inclination. Experiments after the competition using manual operation confirmed the outstanding locomotion capabilities. However, to fully exploit the capabilities the planning parameter required tuning which needed to be done during the run. Issues with the communication infrastructure eventually prevented parameter adaptation and showing off the full capabilities of the system during the competition.

IV. LESSONS LEARNT

There is much to be learnt from a competition and host as well as participants gain experience and can reflect on development strategies, design choices and the implementation with respect to the final performance. Thus, the following discussion will take a critical look at the performance of Artemis in the competition and the decisions made during and prior to the competition.

Incremental goals and the critical path: The decisions made prior to the competition were influenced by an optimistic and ambitious attitude and the goal to complete the full mission – not just parts of it. While resource limitations existed only small time buffers could be accounted for, reducing integration and test time for the fully integrated system to a minimum. The initial focus was put on designing the high-level functionality of Artemis to allow for autonomous operation. However, the competition showed that manual interaction with the operator remains a critical and substantial element for error handling. Eventually, remote operation of the system was a single point of failure of Artemis. While the development targeted a fully autonomous system the risk of not achieving this goal was high. A dedicated approach for risk mitigation would have identified operations as an element of the critical development path leading to a (re)prioritization of the implementation task for operations.

Maintain a robust development procedure: The model-based development approach throughout all development phases proved to be highly beneficial – thanks to a well structured and proven workflow of Rock. Package management facilitated integration of external packages as well as the management of existing ones, and the general encapsulation of functionality in libraries served for good reusability of existing functionality. Furthermore, auto-generation of component's framework code allowed to easily maintain the framework specific (oroGen) components and create interface contracts using well defined input and output types. Finally, creating components with standardized interfaces using Rock allowed to apply a proper system management tool.

Rock facilitates many tasks when developing a robotic system, but additional complexity arose using Rock components and ROS nodes in parallel. The supervision module was capable of handling both component types, yet this functionality was a recent development and as such did not have a perfect integration in the existing development workflow. We did not enforce the workflow early on all developers for testing and smoothing the process and suffered debugging efforts in later stages of the development. Thus, maintaining a robust and reliable workflow when developing complex systems should be given high priority.

The human factor in a component-based development approach: The theoretical benefit of a component-based system is the ease of integration. Artemis development showed that this assumption does directly depend on the maturity of the components, i.e. when interfaces including configuration properties require frequent updates integration becomes much harder. In contrast to the previously mentioned robust workflow for the component design, Artemis' integration workflow to create high-level functionality from these components as part of the supervision showed some

weaknesses with respect to communication between system specialists and system integrators. System specialists used a different set of tools for performing small integration testing than the system integrator which led to some redundant work and a communication gap. The workflow for creating and testing high-level functionality should be homogeneous and allow to communicate and propagate requirements and semantics of components more clearly – ideally in a model-based fashion to allow verification.

Top down versus bottom up: A top down approach seems to be desirable for developments that focus only on a given mission scenario. Developing for Artemis started with a top down approach, but soon turned into a mixture of bottom up and top down approach. The mixed approach originated from reusing existing components with new experimental components where details of the implementation were unforeseen. From our experience, an experimental development approach seems to favour a bottom-up strategy in combination with an agile development approach. However, componentization and modularization is motivated by fostering reusability and a top down view will still be beneficial to identify generic, reusable parts. From our experience as soon as the development turns experimental the impact of a top down approach is severly limited and top down development efforts can stay on an abstract level without detailing interfaces precisely.

Testing: At the time of development we missed advanced offline unit-test facilities in the supervision to evaluate high-level functionality. Since these runtime tests had to be performed either in simulation or on the real system this led to a development slow-down. This stressed that unit-tests should not be missed at any level of hardware and software. Main parts of the initial development for Artemis relied on software simulation and the complementary application of simulation and real world testing led to an increase of the team's efficiency and allowed to cope with the short time frame for integration testing, since faulty behaviour of the real system could be fixed using simulation.

V. CONCLUSIONS & FUTURE WORK

The result of the competition showed that handling a complex navigation and manipulation scenario autonomously is more than putting the parts together. Although the individual parts required for the activities have been extensively researched in the past, the application of the individual skills in an integrated scenario provides additional challenges with many interesting open questions. On the software level the relation between task components, operations and failure management is likely a key element in the advancement towards robust autonomy in complex real world settings. Explicitly setting up the layout of the components of a system does not scale very well in terms of complexity and robustness. As an alternative, functional models of the components and their connections can be used for decomposition, validation and reconfiguration of the component networks. The Rock framework already provides many tools for the support of this approach, however there is much work to be done in order to make it more accessible and robust. Events like the SpaceBot Cup are a great way to probe the abilities and force the evaluation of fully integrated systems and all the problems that come with it.

ACKNOWLEDGMENT

The work presented here is part of the Project "SpaceBot", which is funded by the Federal Ministry for Economics and Technology (BMWI) through the German Space Agency (DLR) grant number 50RA1318.

REFERENCES

[1] B. Harvey, *Soviet and Russian Lunar Exploration*, ser. Springer Praxis Books. New York, NY: Praxis, 2007.
[2] J. P. Grotzinger, "Exploring martian habitability. Habitability, taphonomy, and the search for organic carbon on Mars. Introduction." *Science (New York, N.Y.)*, vol. 343, no. 6169, pp. 386–7, Jan. 2014.
[3] E. Lakdawalla, "China lands on the Moon," *Nature Geoscience*, vol. 7, no. 2, pp. 81–81, Jan. 2014.
[4] C. Kunz, C. Murphy, and H. Singh, "Toward extraplanetary under-ice exploration: Robotic steps in the Arctic," *Journal of Field Robotics*, 2009.
[5] S. Bartsch and T. Birnschein, "Development of the six-legged walking and climbing robot spaceclimber," *Journal of Field Robotics*, vol. 29, no. October 2008, pp. 506–532, 2012.
[6] R. J. Léveillé and S. Datta, "Lava tubes and basaltic caves as astrobiological targets on Earth and Mars: A review," *Planetary and Space Science*, vol. 58, no. 4, pp. 592–598, Mar. 2010.
[7] J. Schwendner and S. Joyeux, "Classifying Autonomy for Mobile Space Exploration Robots," in *i-SAIRAS 2010*, Sapporo, Japan, 2010.
[8] T. Kaupisch and D. Noelke, "DLR SpaceBot Cup 2013 - A Space Robotics Competition," *Künstliche Intelligenz*, 2014.
[9] S. Joyeux, F. Kirchner, and S. Lacroix, "Managing plans: Integrating deliberation and reactive execution schemes," *Robotics and Autonomous Systems*, vol. 58, no. 9, pp. 1057 – 1066, 2010, hybrid Control for Autonomous Systems.
[10] S. Michaud, A. Gibbesch, T. Thueer, A. Krebs, C. Lee, B. Despont, B. Schäfer, and R. Slade, "Development of the exomars chassis and locomotion subsystem," in *i-SAIRAS 2008*, 2008.
[11] N. Patel, R. Slade, and J. Clemmet, "The exomars rover locomotion subsystem," *Journal of Terramechanics*, vol. 47, no. 4, pp. 227 – 242, 2010.
[12] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2011.
[13] A. Segal, D. Haehnel, and S. Thrun, "Generalized-icp," in *Proc. of Robotics: Science and Systems (RSS)*, vol. 25, 2009, pp. 26–27.
[14] N. Sunderhauf and P. Protzel, "Switchable constraints for robust pose graph slam," in *IEEE/RSJ Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 1879–1884.
[15] P. Agarwal, G. D. Tipaldi, L. Spinello, C. Stachniss, and W. Burgard, "Robust map optimization using dynamic covariance scaling," in *IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2013.
[16] R. Triebel, P. Pfaff, and W. Burgard, "Multi-level surface maps for outdoor terrain mapping and loop closing," in *IEEE/RSJ Intelligent Robots and Systems*. IEEE, 2006, pp. 2276–2282.
[17] J. Schwendner, "Embodied Localisation and Mapping," Ph.D. dissertation, Universität Bremen, 2013.
[18] S. Koenig and M. Likhachev, "D*lite." in *AAAI/IAAI*, R. Dechter and R. S. Sutton, Eds. AAAI Press / The MIT Press, 2002, pp. 476–483.
[19] I. Ulrich and J. Borenstein, "Vfh*: Local obstacle avoidance with look-ahead verification." in *ICRA*. IEEE, 2000, pp. 2505–2511.
[20] Reflexxes, "The reflexxes motion libraries," 2013.
[21] A. Sucan and S. Chitta, "Moveit!" 2013. [Online]. Available: http://moveit.ros.org
[22] R. Smits, "Kdl: Kinematics and dynamics library." [Online]. Available: http://www.orocos.org/kdl
[23] MrRheingold, "Spacebot Cup: Robotik Wettbewerb," retrieved March 25 2014, from http://www.youtube.com/watch?v=vaDo4eMk2Go.