Community-Driven Development of Standard Software Modules for Search and Rescue Robots

Stefan Kohlbrecher[1], Karen Petersen[1], Gerald Steinbauer[2], Johannes Maurer[2], Peter Lepej[3], Suzana Uran[3], Rodrigo Ventura[4], Christian Dornhege[5], Andreas Hertle[5], Raymond Sheh[6], and Johannes Pellenz[7]

Technische Universität Darmstadt[1], Technische Universität Graz[2], Maribor University[3], Instituto Superior Técnico[4], Universität Freiburg[5], Robolit LLC[6], Deutsche Bundeswehr[7]

I. Introduction and Motivation

Building a software system for autonomous robots is a difficult and time-consuming task. From the experience at the RoboCup competitions, it takes new teams several years to come up with autonomous robots that are competitive at the RoboCup world championship level. These world class solutions have the tendency to disappear when the teams leave the competition. Code sharing is difficult, because most teams use different, mostly homemade software frameworks. So basic or more advanced modules such as drivers for laser scanners or mapping solutions are re-implemented again and again, which slows down the overall progress of the capabilities of the league and research on autonomous robots in general. Therefore, we proposed to establish an open source standard software solution based on ROS for the RoboCup Rescue Robot League that fosters the development of search and rescue robots. This standard enables new teams to adopt quickly to the world class performance level. The standard software solution makes excellent solutions broadly reusable. This way, each team can focus on their own topics of interest, such as mapping, exploration or victim detection. Overall, it will foster the progress of capabilities developed in the league.

In order to push this idea forward two ROS workshops took place in Koblenz, Germany, in 2010 and 2011, followed by the ROS RoboCup Rescue Summer School in Graz, Austria, in 2012. The outcome was a set of software modules that cover an online mapping solution, laser based localization, and a basic exploration capability. These events also established a group of researchers committed to this initiative. Moreover, these researchers also teach the use of the components, e. g. at the SSRR Summer School 2012 in Alanya, Turkey.

The different events also serve as synchronization points, where new requirements are discussed and prototypes are developed. The main goal of the initiative is to continuously enlarge the set of software building blocks that can be reused in the search and rescue domain.

II. Providing Standard Software Modules

The successful adoption of common software modules by the community has been demonstrated with the *hector_slam* stack for robust SLAM in USAR environments. Originally developed and used by Team Hector Darmstadt, it has been published as open source software and has been used by three of the four most successful teams in the RoboCup Rescue Robot League competition at RoboCup 2012 in Mexico. A detailed description of the SLAM approach is beyond the scope of this paper and available in [1]. Additional packages of the stack allow the creation of GeoTiff maps, storing robot trajectory data and looking up map information. A map created by the *hector_mapping* SLAM node and visualized as a GeoTiff file using the *hector_geotiff* node is shown in Figure 1. The use of the standard mapping package lifted the base line in the league, because providing a map is now a standard for all competitive teams.

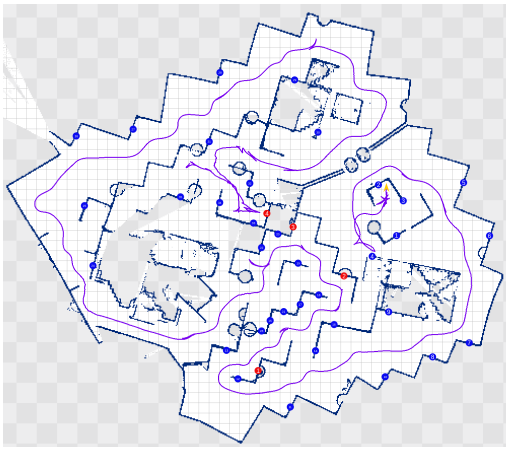

Fig. 1. GeoTiff map created using the *hector_slam* stack. Final competition run RoboCup 2012.

III. Towards Navigation in Unstructured Environments

In order to go one step further the initiative in 2012 focused on the navigation in unstructured environment and developed prototypes for exploiting 3D map information.

For the path planning process, where a path from the robot's current position to the goal is computed, one requires knowledge of the region of the map that is physically accessible by the robot. To address this problem, a grid map specifying

all the locations that are reachable for the robot is built. This map is designated here as *cost-map*. For simplicity sake, a simple binary 2D cost-map was considered. The problem is then how to construct a 2D cost-map, given a 3D octomap of the environment and the robot's current location.

This problem is addressed in two stages. We assume that the octomap is oriented along the world coordinate system. Firstly, the octomap is scanned to identify all map cells that can be considered as ground. We consider a given cell as ground if it is both (1) occupied, and (2) there are enough adjacent free cells above, so that the robot can physically fit into the total free height. These cells are then projected down to a 2D grid. Secondly, once all ground cells are identified, all cells that are 4-neighbor connected with the cell where the robot is currently located are set as reachable. Two cells are considered connected if the height difference between two adjacent ground cells is within a specified steepness threshold. This threshold depends on the robot's physical capability of traversing non-horizontal terrain. Note that these two stages can be run asynchronously, *i.e.,* the first stage only needs to be run every time the octomap is updated, while the second one should be run every time the robot's current location changes.

We have used several octomap datasets, including one built from the NIST arena. The preliminary results match our expectations, including their sensitivity to changes in the steepness threshold. The implementation prototype is currently running in batch mode, but we expect its integration into a ROS node to be straightforward.

Exploration, even in state of the art approaches, is usually guided by a 2D map built using a distance sensor. For USAR environments this is not sufficient, as the focus should be to observe 3D space with a victim sensor (e. g. a thermal camera), that usually has a significantly smaller field of view than, for example, a laser range sensor. To be able to reason about exploration towards finding points of interest such as victims, it is necessary to have knowledge about what area has been covered by the victim detection sensor. This area is likely to be different from the area covered by the mapping sensor.

Thus, we implemented a 3D coverage map, based on octomaps [2]. Besides the occupancy information in each known cell, we additionally store, if this cell has been seen by each individual sensor. We model a sensor observation for the i-th sensor as the 6-dof pose $(\mathbf{x_i}, \mathbf{q_i})$ and field of view $(\phi_i, \theta_i, r_{min_i}, r_{max_i})$, where ϕ_i, θ_i are the horizontal and vertical opening angle and r_{min_i}, r_{max_i} are the minimum and maximum range of the sensor. Given the pose and field of view, we compute a view frustum at the sensor pose and perform raytraces towards all end points at the maximum range. The origin of each raytrace is the sensor pose. We raytrace only through free space modelled in the octomap, and stop when an occupied cell is encountered. All free cells between the minimum and maximum range (or a blocked cell) are then marked as seen for sensor i.

In most cases it is only necessary to reason about covered or uncovered cells near geometry, as it can be assumed that free space cells do not contain anything. Thus we optionally only retrieve cells that have an occupied neighbor cell. To prevent computationally expensive neighbor lookups, we store this property in the cells as well. This property is only computed when a new distance sensor observation was integrated. We use octomap's change detection to only recompute the neighbor property on cells that are semantically changed, e.g. from free to occupied, and their neighbors. This efficient implementation bounds the runtime even in larger maps. An example of the coverage octomap can be seen in Figure 2. In the future, we plan to not only return neighbor cells, but all cells within a distance threshold. This can be done efficiently using the dynamicEDT3D library available in the recent octomap release.

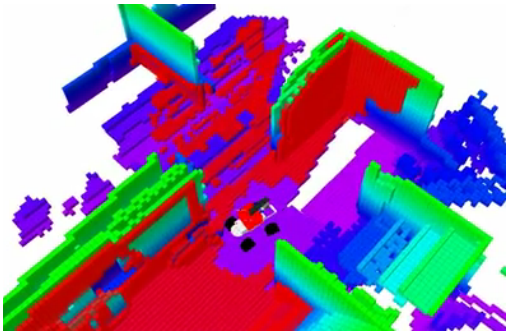

Fig. 2. This figure shows the coverage octomap within a rescue arena. A sideways mounted camera is used as a sensor. (Covered cells in red.)

IV. Conclusion and Future Activities

We presented a successful community-driven initiative for the development of standard software for robots in USAR environments. The adoption of a first set of mapping tools by the RoboCup Rescue Robot League proofs the concept.

The current status of the common framework for RoboCup Rescue initiative is documented in the ROS wiki at `http://www.ros.org/wiki/robocup_rescue`. The page gives also instructions on how to get and how to install the software components, and how to get involved in the initiative.

We plan to continue the initiative with a summer school on *Perception and Action in 3D Environments*, co-located with the SSRR symposium 2013.

V. Acknowledgment

We like to thank all individual researchers and research groups that have contributed to the standard software initiative. The initiative has been partly funded by the RoboCup Federation, the European Fund for Regional Development (EFRE), the Land Steiermark, and the Republic of Slovenia.

References

[1] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, November 2011.

[2] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, Anchorage, AK, USA, May 2010.