Team Description Paper

Jon Martin, Daniel Ammon, Helmut Engelhardt, Tobias Fink, Florian Gramß, Alexander Gsell, Dominik Heigl, Philipp Koch, and Marco Masannek

Institute of Technology Georg-Simon-Ohm,
Kesslerplatz 12, 90489 Nuremberg, Germany
http://www.th-nuernberg.de

Abstract. This is an amazing abstract. It will be more substantial when more text ist added to the main part.

@phil: abstract

1 Introduction

The Autonohm@work team at the Nuernberg Georg Simon Ohm University of Applied Sciences was founded in September 2014. The team consists of Bachelor and Master students leaded by a research assistant who guides and supervises them. To develop a functional mobile-robot- manipulator the different groups in the team had put much effort and knowledge into research to develop the packages to manage the robot. Our main focus is attended to mobile manipulation, object perception and navigation in a unconstrained environment. Only two members of the team that took part last year in Magdeburg continue. To compensate that, new members have joined us and the cooperation between the rescue and the atwork teams has been intensified.

2 Hardware Description

We use the KUKA youBot omni-directional mobile platform, which is equipped with a 5 DOF manipulator (Figure 1). At the end effector of the manipulator an Intel RealSense camera with a motion sensor has been mounted. Next to the camera we replaced the standard gripper from youBot with an also youBots soft two-finger gripper. Thanks to it we are able to grasp bigger and more complex objects more precisely.

A Hokuyo URG-04LX-UG01 laser scanner at the front of the youBot platform is used for localization and navigation. We are planning to add a second laser scanner of the same type on the back of the robot. This improves localization quality and ensures better obstacle avoidance, mainly when driving backwards with the robot.

Last year we used the internal computer, together with an external ASUS Mini PC (4 GB RAM, Intel Core i3). We used the internal computer in the youBot to start-up the motors and also for the SLAM. This was a huge error because we added an enormous data transfer between the two computers, what slowed down the complete system. To avoid the communication problems and

latency between them both, we decided to run everything, except the motor drivers, on the external PC. We also replace the slow i3 for a more powerful CPU Intel Core i7-4790K, 4x 4.00GHz. Table 1 shows our new hardware specifications.

fill in specs and format table

For connecting the two PCs we use a router mounted on the back of youBot. With external machines we can connect to the routers network and communicate with both PCs on the youBot.

We also added an Inertial Measurement Unit (IMU) to gather information about the heading of the platform. On the IMU runs a fusion algorithm which provides the orientation of the platform as quaternion or Euler angles.

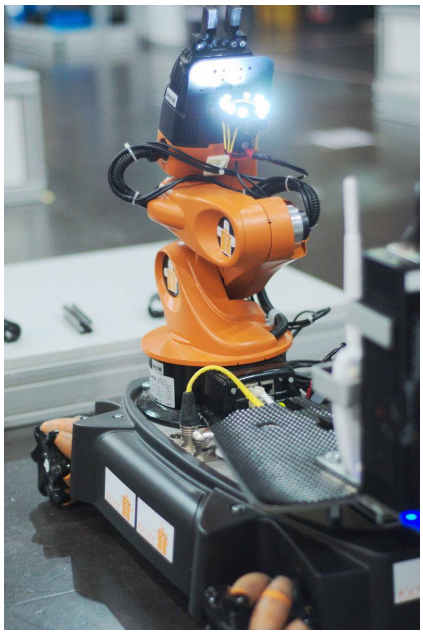

Fig. 1. KUKA youBot Plattform

Tab. 1. Hardware Specifications

PC 1	
CPU	XXXXXXXX
RAM	XXXXXXXX
HDD	XXXXXXXX
OS	Ubuntu 14.04
PC 2	
CPU	XXXXXXXX
RAM	XXXXXXXX
HDD	XXXXXXXX
OS	Lubuntu 14.04
Grasp	
a	XXXXXXXX
b	XXXXXXXX
c	XXXXXXXX
Lidar	
a	XXXXXXXX
b	XXXXXXXX
c	XXXXXXXX
Router	
a	XXXXXXXX
b	XXXXXXXX
c	XXXXXXXX
IMU	
a	XXXXXXXX
b	XXXXXXXX
c	XXXXXXXX

3 Software

The software architecture is based on the Robot Operating System ROS. The system runs with Ubuntu 14.04 and ROS Indigo installed. The ROS communication infrastructure is used to communicate and pass information between the

nodes, as for example camera data or execution orders for the 5 DOF manipulator.

Several software tools are needed for image processing and controlling the system. In the following we generally describe the software we use when developing and in the contest.

Unless otherwise stated, the following software is available from the official ubuntu software repositories.

cgdb For online-debugging we use cgdb. It is a command line interface to the GNU debugger software.

dd We create images from the hard discs to recover from a hard disc failure at the contest.

eclipse For all C++ development we use eclipse IDE.

chrony Using ROS on multible machines requires time synchronisation between them. We use chrony for synchronizing the time on PC2 with the time on PC1.

git Our software is maintained via git version control system. Git enables us working simultaneously on our software tree.

htop To watch active processes and CPU-Load we use htop.

openssh-server It provides access to the robot via ssh protocol.

QtSixA We steer our youBot with a Playstation 3 joystick. We need this software to connect the joystick to the PC via bluetooth.[1]

screen Each time we accessing the platform through ssh we enter a screen session. With this software we are able to open multiple command line sessions, set a split screen and reconnect to a session if network connection drops.

stress We used stress to test our thermal management. It immediately can switch all CPU load to 100%, so it simulates worst case CPU load.

vim For developing over ssh connection on the robots PCs, or editing ROS-Launchfiles we use vim text editor with several plugins.

OpenCV OpenCv has been used to build an 2D image processing node. Some useful functions and algorithms, have been included into our object recognition.[2]

Additionally we use the following ROS[3] packages:

amcl For localization problem we eventually use amcl package. It implements a particle filter algorithm. (see chapter 6)

map-server Storing and loading occupancy-grid-maps is done by the map-server package.

navigation For global/local path planning, path regulation and obstacle detection/avoidance we plan to use the navigation stack from ROS.

[1] http://qtsixa.sourceforge.net/
[2] http://opencv.org/
[3] http://wiki.ros.org/

robot-pose-ekf For fusing the data from the IMU and the wheel encoders we use the robot-pose-ekf. It is an implementation of an extended Kalman filter algorithm. It enables us to mix the two systems and improves the odometry data from the wheel encoders.

stdr-simulator We use it for simulating everything except the actuator. Every team member can test software through simulation in STDR, before testing with the robot in real.

youbot-driver Our youBot platform is controlled with this package. So we are able to send movement commands from PC1 to PC2 over ROS. The Drivers are running on PS2 , which is connected to the motor drivers and the actuator via EtherCAT protocol.

Finally our own software packages:

ohm-tsd-slam For recording a map from the environment we use out own SLAM algorithm. (see chapter 5)

particle-filter For localization problem we eventually use our own particle filter instead of amcl package. (see chapter 6)

statemachine To control the robot we have implemented a statemachine algorithm with the statemachine framework from our laboratory. (see chapter 7)

4 Image Processing

To grab an object reliable, we needed a stable image processing. Our current system uses a 2D camera. The used Intel RealSense camera is also able to handle future tasks like 3D object recognition. To accomplish 3D tasks, the structured light method will be used.

Important criteria for the 2D image processing:

1. speed
2. stability
3. low use of cpu resources

To fulfill these criteria, the OpenCV[4] library has been used. The incoming image gets converted into a gray-scale image. Some times it's necessary to apply an Median or Gaussian filter afterwards, to get better results. Finally the edges will be extracted through the canny edge detector. The received binary edges image is now ready to be processed.

To recognize a certain object, a complete contour needs to be defined. Complete contours get identified through the findContours function from OpenCV. The saved contours get sorted by given attributes like number of corners, area size or diagonal length.

If a object has been identified, the position of the object gets calculated. To get accurate results, the distance between camera and object needs to be fixed. Through a fixed distance, pixel data can be converted into meters. For a prices griping X,Y,Z coordinates and the Z rotation of the object will be calculated. After calculation the data gets published and accessible to the manipulation.

[4] http://opencv.org/

5 Mapping

The arena at RoboCup@work is virtually unchanged during the competition. Therefore, a SLAM approach is unnecessary. Since industrial production sites have in general a static layout, this assumption is justified. However, a SLAM algorithm is required to generate the initial map, or update the information to add smaller changes.

The Autonohm RoboCup Rescue team deploys successfully a self developed SLAM approach, ohm-tsd-slam. The referring ROS package is based on a 3D/2D reconstruction and localization framework which has been subject of May et. al. [8]. Additional work by Koch et. al. [7], aimed at multi-source-SLAM and loop closing capabilities.

In order to build maps of the RoboCup@work arena, only data of a 2D LIDAR is required. In an iteration, the SLAM framework first reconstructs an artificial laser scan out of the map from the last known robot pose, using a raycasting implementation.

In the second step, a scan matching algorithm based on ICP (Chen and Medioni [3], Besl et. al. [2], Zhang [10] and RANSAC (Fischler et. al. [6])) algorithm estimates the transformation between the reconstructed laser data and and the current scan. This pose change is applied to the last known pose and results in the new localization of the robot.

The map is being updated from the new acquired robot pose and converted in a ROS compatible data format (occupancy grid) in the last step. This step is necessary as the ohm-tsd-slam package uses an abstract representation based on Signed Distance Functions (SDF) (Curless et. al. [4]), which is incompatible to standard ROS packages. Figure 2 depicts a map generated by ohm-tsd-slam.

More information and a git repository regarding the 2D single or multi-SLAM ROS package ohm-tsd-slam can be found on the referring ROS Wiki page [1].

6 Localization

As a first attempt on the RoboCup atWork competition, we decided not to try the Precision Placement and Conveyor Belt test. It was a good idea because during this attempt we had big problems with the localization and navigation, which are basic functions. This limited us to try further and more difficult tests. We gain experience and improve our software so we will try this time more difficult tests.

consistent style of RaW

We have three different strategies concerning the localization problem. The first one is to use our own particle filter algorithm. The second is to use the amcl package from ROS. And the third strategy is to use our SLAM algorithm mentioned in chapter 5 for localization. Because we have not yet decided which solution to use, we will mention all three in short.

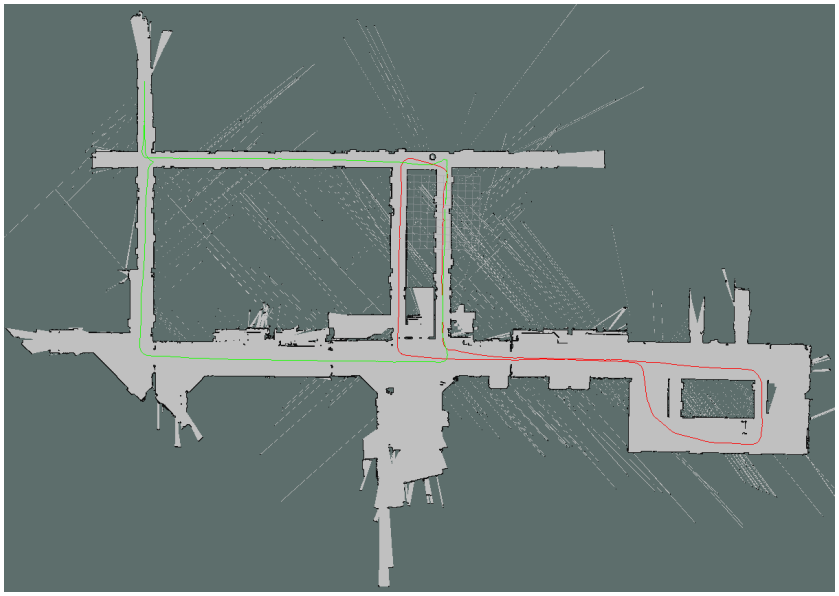

Fig. 2. Mapping Example. Image depicts the resulting map of two cooperating robots. The red and green lines illustrate the estimated trajectory of both units. This map shows an additional feature of ohm-tsd-slam, the approach is able to map comparably large areas as the reconstructed map had an edge length of 122 m. *Source: Koch et. al.[7].*

6.1 Particle-Filter (TH-Nuernberg)

We are working on an own particle filter algorithm at our laboratory. Its functionality is close to amcl localization, [5] and [9]. If we get it working in time, we would like to use our own software at the contest.

6.2 Particle-Filter (ROS-AMCL)

The navigation stack from ROS-System includes a package called amcl (adaptive Monte Carlo localization). It provides a particle filter algorithm for robot localization. We already checked the compatibility of amcl algorithm and our SLAM approach. So we are able to record a map with our SLAM algorithm and afterwards locate and navigate in that map via amcl and navigation stack from ROS.

6.3 SLAM for Localization

SLAM means Simultaneous Localization and Mapping. This is because while recording a map, the robot needs to know its position in the map - so it must

locate itself in the map while building the map. If we disable the mapping part from our SLAM algorithm, we are able to load a previously recorded map and locate the robot. The problem with this approach is that if the algorithm fails to process one measurement, the localization is lost in most cases and we have to quit the run. Particle filters are able to recover from such failures.

7 Mission Planning

For the main control of the system, a State Machine with singleton pattern design is used. Every state is designed to be as small as possible. For the German Open 2015, we implemented three main states divided on smaller sub-states: move, grasp and deliver (see figure 3).

On the initialization state, the robot receives the map and localizes itself on it. Then we go to state Idle and wait until the tasks are received from the referee box. We divide the complete task into smaller subtask and manage them on the stateNext.

The first step is always to drive to a specific position with a specific orientation. In this case we would first plan a trajectory on the map, drive to the desired position by following the path waypoints and finally wait there for some seconds. Once the moving task is finished we return to StateNext to check which is the next desired state.

If we want to grasp an object, we would first smoothly approach to the service area and go to stateFindDesiredObject. After localizing the position of the desired object, we grasp it, lay it in our back and return back from the service area.

In case of delivering an object, we also approach smoothly to the service area but this time we will pick up the object from our back and deliver it on the service area. Finally we return back from the service area and go again to StateNext.

The State Machine framework could be found on GitHub under our Laboratorys repository: autonohm/obviously.

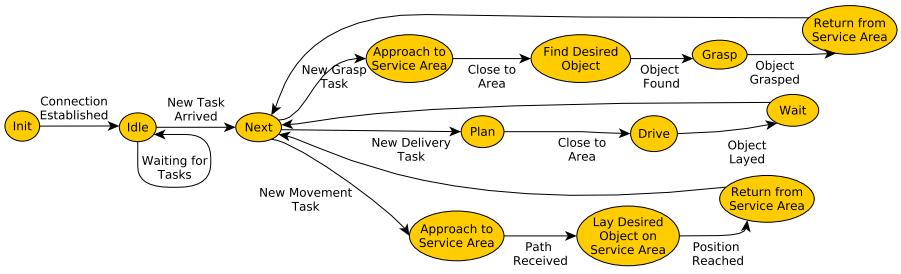

Fig. 3. Structure of the Statemachine

8 Object Manipulation

To grasp objects reliably an exact position from the object perception is needed. The position of objects will be calculated based on information, received from optical/infrared sensors (2D and 3D). After the calculation is finished the robot will navigate to a pregrasp position. Once the base has reached the final position, kinematics will lead the arm near the object. For precise gripping a 2D/3D optical/infrared sensor has been attached to the end effector. In gripping stance the arm- 2camera will be activated to measure the final gripping pose. Because manipulation is an upcoming issue in our robotic institute, we decided to build our own inverse kinematic.

9 Reusability and Applicability

Every time we develop software, we try to link the ROS system as late as possible. So we can ensure good reusability and applicability on other systems? Additionally most of our software is managed in standalone packages, which makes integrating them in other systems very easy. With our laboratory's library[5], which includes many standard solutions to robotic problems, we prevent rewriting or copying code and we simplify software maintenance.

10 Conclusion

consistent writing

In this paper we first briefly introduce our Autonohm team and team members. We then present the HW we use and give a brief explanation about the SW packages we are using. We also describe more in detail the software developed by us such as the 2D image processing software, mapping, localization and state machine.

we also 2x

[5] https://github.com/autonohm/obviously

References

1. Ohm tsd slam. `http://wiki.ros.org/ohm_tsd_slam`. Online; accessed 29-March-2016.
2. P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14(2):239–256, Feb 1992.
3. Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 2724–2729 vol.3, Apr 1991.
4. Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA, 1996. ACM.
5. F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo localization for mobile robots. *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, 2(May):1322–1328, 1999.
6. Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.
7. Philipp Koch, Stefan May, Michael Schmidpeter, Markus Kuhn, Christian Pfitzner, Christian Merkl, Rainer Koch, Martin Fees, Jon Martin, and Andreas Nuchter. Multi-robot Localization and Mapping Based on Signed Distance Functions. *2015 IEEE International Conference on Autonomous Robot Systems and Competitions*, pages 77–82, 2015.
8. Stefan May, Philipp Koch, Rainer Koch, Christian Merkl, Christian Pfitzner, and Andreas Nüchter. A Generalized 2D and 3D Multi-Sensor Data Integration Approach based on Signed Distance Functions for Multi-Modal Robotic Mapping. *19th International Workshop on Vision, Modeling and Visualization*, 2014.
9. S. Thrun, W. Burgard, and D. Fox. *Probablistic Robotics*. Massachusetts Institute of Technology, 2006.
10. Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *Int. J. Comput. Vision*, 13(2):119–152, October 1994.