Team Description Paper – Team AutonOHM

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
{jon.martingarechana,ammonda48113,engelhardthe57850,
finkto51573,gramssfl60109,gsellal39762,heigldo49073,
philipp.koch,masannekma61828}@th-nuernberg.de
http://www.autonohm.de

Abstract. This team description paper describes the participation of Team AutonOHM in the RobuCup@work league. It gives a detailed description over the team and the previous achievements. Furthermore, improvements to hard- and software for the participation in future competitions are discussed.

1 Introduction

The AutonOHM@work team at the Nuremberg Institute of Technology Georg-Simon-Ohm was founded in September 2014. The team consists of Bachelor and Master students, supervised by research assistants and a professor.

The team is divided into groups, each developing specific parts of the robot control software, such as manipulation, object detection, localization and navigation. AutonOHM participated for the first time in the German Open 2015, achieving the 5th place out of seven. Since this was the first competition ever, the team was satisfied with the result.

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 4.00 GHz. Table 1 shows our new hardware specifications.

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.

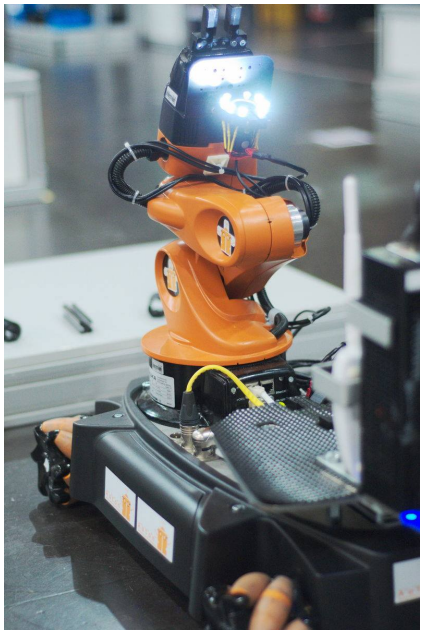

Fig. 1. KUKA youBot Plattform

Tab. 1. Hardware Specifications

PC 1	
CPU	Intel i7-4790
RAM	16 GB DDR3
Storage	SSD 128 GB
OS	Ubuntu 14.04
PC 2	
CPU	Intel Atom D510
RAM	2 GB DDR2
Storage	SSD 32 GB
OS	Lubuntu 14.04
Grasp	
Type	soft, two-finger
Stroke	20 mm
Lidar	
Type	Hokuyo URG-04LX
Range	5.6 m
Resolution	0.352 deg
Router	
Type	Edimax 2.4/5 GHz
IMU	
Type	BNO055 Bosch
DOF	9
Rate	30 Hz (ROS)

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, software used for developing or to control the robot in the competition is listed.

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 both PCs.

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 access 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.

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

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

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 the reality.

youbot-driver Our youBot platform is controlled with this package.

Finally our own software packages:

ohm_tsd_slam For recording a map from the environment we use our 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 reliably, we needed a stable image processing. Our current system uses a 2D camera. 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 is necessary to apply a 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 an object has been identified, the position of the object is calculated. However, as only a 2D sensor is used, the distance between object and camera has to be known. With this information, the 3D coordinates of the object are estimated and can be used for instance to manipulate.

[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 deployed 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. [7]. Additional work by Koch et. al. [6], aimed at multi-source-SLAM and loop closing capabilities.

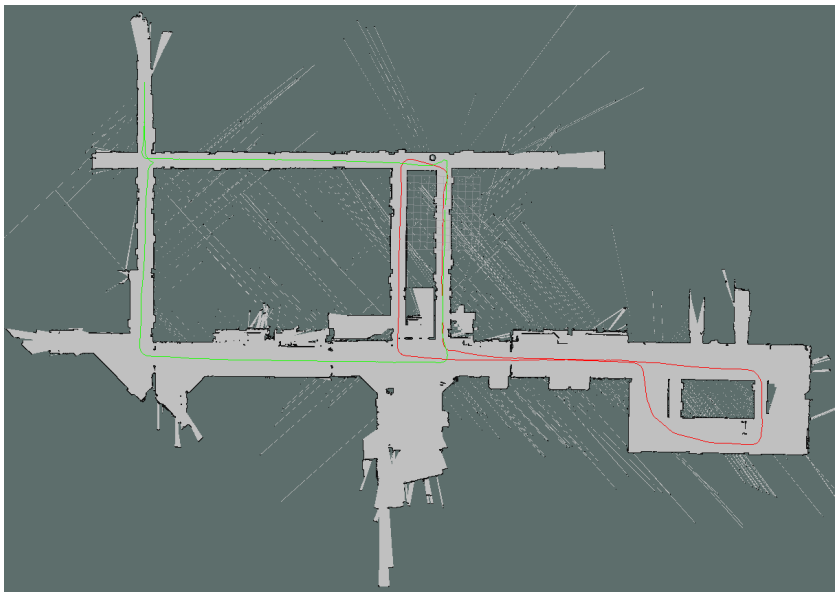

Fig. 2. Mapping Example. Image depicts the resulting map of two cooperating robots. The red and green lines illustrate the estimated trajectory of both units. This map shows an additional feature of ohm-tsd-slam, the approach is able to map comparably large areas as the reconstructed map had an edge length of 122 m. *Source: Koch et. al.[6].*

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 the Iterative Closest Points (ICP) algorithm (Chen and Medioni [2], Besl et. al. [1], Zhang [9] and Random Sample Consensus (RANSAC) algorithm (Fischler et. al. [5])) estimates the transformation between the reconstructed laser data 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. [3]), 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[5].

6 Localization

Currently, the team considers three different strategies concerning the localization problem. The first one is using a self developed particle filter algorithm. The second is to use the amcl package from ROS. And the third strategy is to use the self developed ohm_tsd_slam algorithm mentioned in chapter 5. In the future, all strategies will be tested and compared.

6.1 Particle-Filter (TH-Nuernberg)

The team works currently on an own particle filter algorithm. Its functionality is close to amcl localization, [4] and [8].

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.

The compatibility of amcl algorithm and our SLAM approach has already been tested. It is possible to record a map with ohm_tsd_slam and afterwards localize and navigate in that map via amcl and navigation stack from ROS.

6.3 SLAM for Localization

It is possible to disable the mapping in the ohm_tsd_slam package, using a localize only mode. In this mode, the software loads a previously recorded map, localizes the robot using its sensor data but does not alter the map. The main weakness of this approach compared to a particle filter is a possible wrong localization. A particle filter is able to redetect a lost pose using global localization, the ohm_tsd_slam package does not provide this feature.

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

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. A state Idle is entered after that which waits until a task is received from the referee box. The complete task is divided into smaller subtasks and managed in stateNext.

The first step is always driving to a specific position with a specific orientation. In this case, the path planner estimates a trajectory on the map. The robot drives to the desired position by following the calculated waypoints. Once the moving task is finished the state machine returns to StateNext, to process the next task.

In case of grasping, the robot approaches the service area and the state machine enters stateFindDesiredObject. After localizing this object, it is grasped and stored on the back of the robot.

In case of delivering an object, the robot also moves into the service area, picks the object from its back and delivers it to the service area.

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

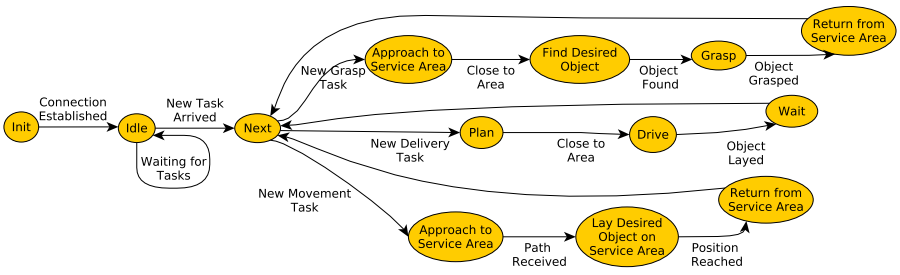

Fig. 3. Structure of the Statemachine

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-camera will be activated to measure the final gripping pose. Because manipulation is an upcoming issue in our robotic institute, a self developed inverse kinematic is deployed.

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 the software is managed in standalone packages, which eases integrating them into other systems. By adding code to our laboratory's library[6], which includes many standard solutions to robotic problems, rewriting or copying of code is prevented and software maintenance is simplified.

10 Conclusion and Future Work

This paper describes the participation of team AutonOHM in the RobuCup@work league. It provided detailed information of localization, autonomy, image processing and object manipulation.

In future work, 3D perception will be used for object localization. Moreover, the team plans on deploying self developed software for localization and navigation to improve the robot autonomy.

Furthermore, it is planned to mount a second laser scanner facing backwards, in order to improve localization quality. Moreover, the second sensor would allow backwards driving as obstacles behind the robot can be detected as well.

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

References

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. S. Thrun, W. Burgard, and D. Fox. *Probablistic Robotics*. Massachusetts Institute of Technology, 2006.
9. Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *Int. J. Comput. Vision*, 13(2):119–152, October 1994.