b-it-bots
RoboCup@Work
Team Description Paper

Shehzad Ahmed, Torsten Jandt, Padmaja Kulkarni, Oscar Lima, Arka Mallick, Alexander Moriarty, Deebul Nair, Santosh Thoduka, Iman Awaad, Rhama Dwiputra, Frederik Hegger, Nico Hochgeschwender, Jose Sanchez, Sven Schneider, Gerhard K. Kraetzschmar

Bonn-Rhein-Sieg University of Applied Sciences
Department of Computer Science
Grantham-Allee 20, 53757 Sankt Augustin, Germany

Email: <first_name>.<last_name>@inf.h-brs.de
Web: www.b-it-bots.de

Abstract. This paper presents the b-it-bots RoboCup@Work team and its current hardware and functional architecture for the KUKA youBot robot. We describe the underlying software framework and the developed capabilities required for operating in industrial environments including features such as reliable and precise navigation, flexible manipulation and robust object recognition.

1 Introduction

The b-it-bots RoboCup@Work team at the Bonn-Rhein-Sieg University of Applied Sciences (BRSU) was established in the beginning of 2012. Participation in various international competitions has resulted in several podium positions, including runner-up at the world championship RoboCup 2014 in Brazil. The team consists of Bachelor of Computer Science and Master of Science in Autonomous Systems students, who are advised by one professor. The results of several research and development (R&D) as well as Master's thesis projects have been integrated into a highly-functional robot control software system. Our main research interests include mobile manipulation in industrial settings, omni-directional navigation in unconstrained environments, environment modeling and robot perception in general.

2 Robot Platform

The KUKA youBot [2] is the applied robot platform of our RoboCup@Work team (see Figure 1). It is equipped with a 5-DoF manipulator, a two finger gripper and an omni-directional platform. In the front and the back of the platform,

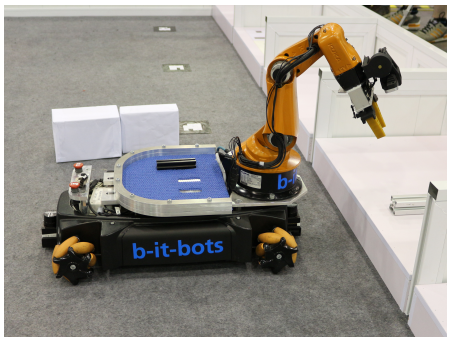


Fig. 1: b-it-bots robot configuration based on the KUKA youBot.

two Hokuyo URG-04LX laser range finders are mounted to support robust localization, navigation and precise placement of the omni-directional base. Each laser scanner is configured with an opening angle of 190° to reduce the blind spot area to the left and right of the robot. For perception-related tasks, a close-range RGB-D camera is mounted on the manipulator to the gripper palm. The Intel® RealSense™ F200 camera [1] supports a range from approximately 0.2 m up to 1.2 m and operates on 30 Hz for the RGB- and on 60 Hz for the depth stream. This sensor information is used for general perception tasks, such as: 3D scene segmentation, object detection, object recognition and visual servoing. The standard internal computer of the youBot has been replaced by one which has an Intel® Core i5 processor in order to perform the perception tasks; which are computationally intensive. Another custom modification has been made for the youBot gripper (based on the design of [3]) which enhances the opening range to 6.5 cm and enables grasping of a wider range of objects. The new gripper (see Figure 2) is actuated with two Dynamixel AX-12A servo motors which provide a position interface and force-feedback information. This information can be used to perform grasp verification. A final modification was performed on the back platform of the youBot. It has been replaced with a new light-weight aluminum version, the position of which can be manually adjusted forward and backward.

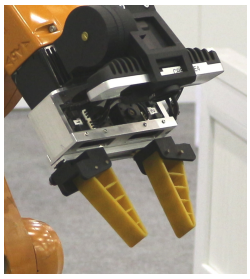


Fig. 2: Custom designed gripper for grasping a wider range of objects.

This allows to adapt the position of the plate for different tasks and objects easily.

All technical drawings to the previously described modifications, as well as various 3D printed sensor mounts, e.g. for the laser scanner and the RGB-D camera, have been made public [6].

3 Robot Software Framework

The underlying software framework is based on ROS, the Robot Operating System [9]. We use the ROS communication infrastructure to pass information as messages on topics between the functional components. Compared to services and actions, topics support non-blocking communication and the possibility of listening (e.g. by monitoring nodes) to the communication between two or more nodes at any time. Further, we refactored many of our components from nodes into nodelets, especially those components which pass large amount of data among each other, like pointclouds or images. The wide range of various tools provided by ROS are utilized for visualization, testing and debugging the whole system. In our development process, we focus on designing small, light-weight and modular components, which can be reused in various processing pipelines, e.g. in pipelines of different domains or even on a different robot platform, like the Care-O-bot 3 [10]. We have also standardized our nodes with the addition of `event in` and `event out` topics. Our components listen to the `event in` topic which expects simple command messages and allow for: starting, stopping or triggering (run once) of nodes. The components provide feedback of their status on the `event out` topic when they finish. This allows us to coordinate and control the components with either simpler state machines or task planning; in either case, the control flow and data flow between the components remains separated. This also allows us turn off computationally expensive nodes when they are not needed.

In 2014, the team published a snapshot of their repository [5] used for the KUKA youBot and the @Work league.

4 Navigation

Several components have been developed and integrated to move the robot from one place to another in cluttered and even narrow environments.

4.1 Map-based Navigation

The navigation components we use are based on the ROS navigation stack *move_ base* which uses an occupancy map together with a global and local path planner. For the local path planner a Dynamic-Window-Approach (DWA) is deployed which plans and executes omni-directional movements for the robot's base. This enhances the maneuverability, especially in narrow environments.

The vast amount of configuration parameters of the *move_base* component have been fine-tuned through experiments with several and differently structured environments in simulation (e.g. a corridor, narrow passages, maze, etc.). While conducting these experiments, several performance criteria, like run-time, minimum/maximum linear and angular velocity and the distance to obstacles, were collected and evaluated throughout the different test cases. In each run only one parameter was changed (e.g. inflation radius, maximum linear velocity, etc.). The experiments reveal, that there are strong dependencies between different parameters, which makes the fine-tuning even more complicated. After several iterations of the same experiments with different parameter sets in an simulation environment, similar experiments have been executed on the real robot, to validate the performance on the actual hardware. Our robot is able to navigate with a maximum linear velocity of 1.0 m/s and a maximum angular velocity of 1.5 m/s. The lateral speed is kept low due the relatively large blind spots to the left and right of the robot. Although these are not the maximum velocities of the actual hardware, there exists a tradeoff between speed and reliability, and these velocities were found to perform well in our test environments. With these velocities the robot is still able to react fast enough to avoid, decelerate or brake, when there are objects moving dynamically in the environment.

In order to avoid unnecessary rotations while following a path, particularly upon arriving at a target location, we extended the default *move_base* global path planner. This extension calculates an orientation for each pose of the generated global plan based on the length of the overall path. The desired behavior is that a short path will be driven in a fully omni-directional manner, while a long path is divided into three segments. Within the first and the last segment, the robot drives in omni-directional mode. In between those two segments, the robot moves in differential mode. Currently, these orientations are calculated and added to the global plan, but not yet taken into account by default local planner. Future development focuses on a modified local planner (based on e.g. DWA and/or Elastic Band), which incorporates these orientations and produce a much smoother and more efficient motion.

4.2 Force-Field Recovery Behavior

In certain situations, especially in narrow passages, the robot can get stuck; for example, due to an overshoot the robots circumscribed radius is now in an area of the costmap which is annotated as an obstacle. The ROS navigation stack does not yet provide a proper behavior to recover from such kind of situation. We use a recover behavior which moves the robot away from obstacles in its vicinity. First, a subsection of the local costmap with lethal cost in a certain radius is taken in to account. Each obstacle in the subsection is considered as a vector applying a repulsive force on the center of the robot. By summing up all force vectors to one overall force, we obtain the best possible direction in which the robot should move to no longer be stuck. The resultant force is multiplied with a velocity scale factor and send as linear and angular velocities to the base controller. This has proved to be very helpful for our navigation and has freed

the robot in many situations. The behavior is implemented as a plugin for the ROS navigation stack and has been tested on several other robots such as the Care-O-bot 3.

4.3 Base Positioning in Front of a Workspace

To enhance the position and orientation accuracy especially in front of a workspace, a simple and effective approach has been used to align the robot perpendicular and in a certain distance to a workspace. A linear regression model is applied to fit a line to each point of the front laser scan. The resulting orientation and distance to this line is fed forward to a closed loop controller which minimizes both errors (i.e. angular and distance), by commanding linear and angular velocities to the robot's base. This component is used to compensate for the insufficient accuracy of the ROS navigation.

5 Perception

Several components have been developed for processing the image and point cloud data from the arm-mounted camera.

5.1 Object Recognition

Perception of objects relevant for industrial environments is particularly challenging. The objects are typically small and often made of reflective materials such as metal. We use a RGB-D camera which provides both intensity and depth images of the environment. This enables effective scene segmentation and object clustering. But the spatial resolution is low even at the close range, and a significant degree of flickering corrupts the depth images. The information captured in a single frame is often not sufficient to determine the object type. Thus, for the object recognition subtask a three-stage pipeline (see Figure 3) has been devised which involves segmentation, accumulation and classification.

The first stage is concerned with scene segmentation, or, more precisely, finding the workspace. We capture a single point cloud and downsample it using a voxel grid filter in order to reduce the computational complexity. We then apply passthrough filters to restrict the FOV which removes irrelevant data and further reduces the computational burden. In order to perform plane segmentation, we first calculate the surface normals of the cloud and use a sample consensus method to segment a single horizontal plane. The convex hull of the segmented plane is computed and represented as a planar polygon. Finally, we shrink the polygon by several centimeters to make sure that it does not include the edge of the workspace. This first stage of the pipeline is composed entirely of ROS nodelets provided in the *pcl_ros* package. All the nodelets can be loaded into a single nodelet manager which is deployed as a single process. Compared to the communication between standard ROS nodes, where the pointclouds are passed over the loopback network interface (if all nodes run on the same machine),

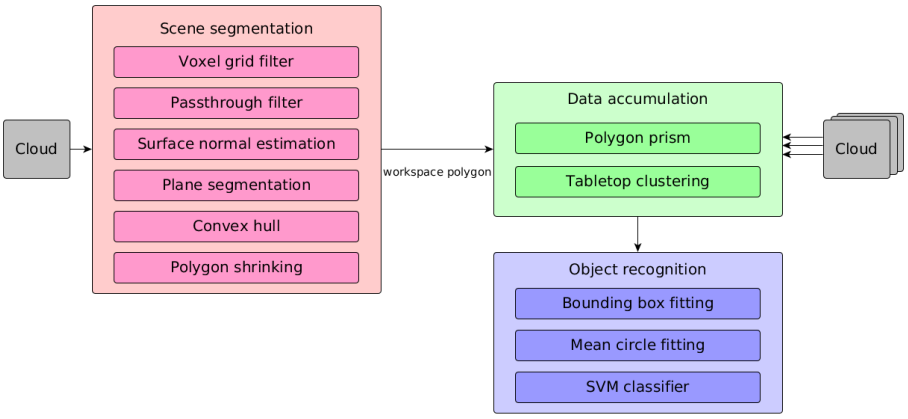


Fig. 3: Object perception pipeline

nodelets communicate over shared memory and there for reduce the communication overhead tremendously.

The second stage is data accumulation. We filter each frame to keep only the points above the workspace polygon (the prism above the polygon), which we then merge into an occupancy octree. Our experiments have shown that five frames are a reasonable tradeoff between the run-time performance and the amount of information accumulated. The accumulated cloud is then partitioned using Euclidean clustering.

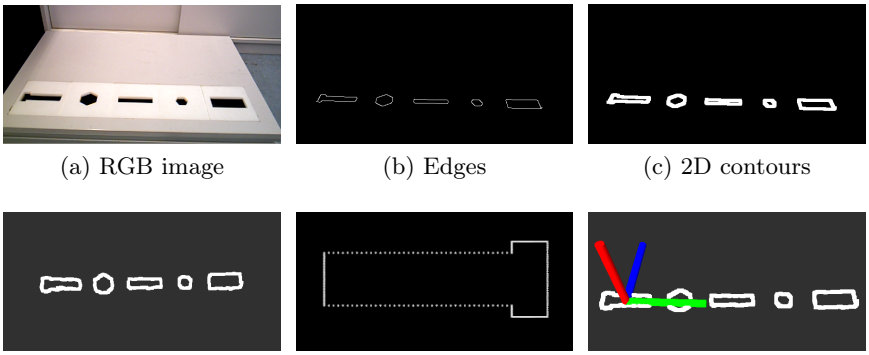
The final stage is object recognition. For each cluster we fit a minimal bounding box and find the mean circle fit perpendicular to the third principal axis of the cluster. Additionally, mean circles are fit on slices made along the first principal axis. The dimensions of the bounding box, radius, fitting error of mean circles, and the average color of all points serve as feature vector for the previously trained SVM classifier. Based on this information, it outputs the predicted object type along with the probability of correct classification.

5.2 Cavity Recognition

For certain tasks, the robot is required to insert objects into cavities (e.g. in a peg-in-hole task). The correct cavity has to be chosen and the respective object needs to be precisely placed into it. A combined 2D and 3D approach has been developed to select the correct cavity for a given object.

Canny edge detection is applied to a grayscale image of the workspace to extract the edges around the cavities and the workspace. The resulting edges are dilated and combined into contours which form the outline of each cavity. Since there are similar cavities with different scales, the 2D contours are not sufficient to distinguish between them. Hence, they are converted into 3D contours using the point cloud of the same scene. The 3D contours are then matched with

known templates of each cavity in order to find the best match. The pose of each cavity is estimated using the centroid of the 3D contour with the x-axis along its principal axis. Figure 4 depicts the output of the different stages of the pipeline.



(a) RGB image (b) Edges (c) 2D contours

(d) 3D contours (e) 3D contour template (f) Matched contour

Fig. 4: Intermediate outputs of the cavity recognition pipeline.

6 Object Manipulation

In order to grasp objects reliably, several components have been developed and integrated on the robot.

From the object recognition, the pose of the object to be grasped is retrieved. This pose is the input to a *pre-grasp planner* component which computes a pre-grasp pose with an offset to the object's pose. The rationale behind this offset is to move the robot's end-effector to a pose near the object instead of directly onto the object to make sure the end-effector will not collide with the surface. Furthermore, the pre-grasp planner outputs a joint configuration for the arm that has it's end-effector aligned (position-wise and orientation-wise) with the object.

These two outputs, pre-grasp pose and joint configuration, are then used by two components that prepare and send motion commands to the base and arm controller of the robot. This results in a simultaneous motion of the robot's base and arm such that the final position of the end-effector, given by the computed joint configuration, is on top of the object. Once the pre-grasp pose is reached, the operation mode is switched to visual servoing which overcomes alignment errors due to object perception and/or motion of the base and arm.

After visual servoing accurately aligns base and arm, the next step is to move the arm to a grasp position. Approaching the gripper to the object is achieved with a component that plans a linear motion in Cartesian space for

the end-effector of the robot. It uses the current pose of the end-effector and the specified distance, from the end-effector to the object grasp pose, to compute a new grasp pose for the object. The two poses are the input for MoveIt! [4] which returns a plan in joint configuration space. This plan is then executed such that the end-effector of the arm moves towards the grasp pose of the object following a linear path in Cartesian space.

Once the end-effector has reached the grasp pose, the gripper of the robot is closed to grasp the object. A grasp monitor check whether the object is grasped successfully or not by using the force and position feedback of the two Dynamixel motors. The grasped object is then placed at a pre-configured position on the back platform of the robot.

7 Task Planning

Many robot application, especially in competitions, have been developed using finite-state machines (FSM). But even for apparently simple tasks, such a FSM can be very complex and thus become easily confusing for humans. Therefore, our current FSMs have been replaced with a task planner.

For preliminary experiments, the Mercury planner [8] has been selected. It achieved the highest score during the International Planning Competition 2014 in a domain which is very similar to the transport domain of RoboCup@Work. The planner allows to specify various cost information. In terms of RoboCup@Work, these costs and can be e.g. distances between locations or probabilities of how good a particular object can be perceived or grasped.

In order to integrate the new planner into our current architecture, the existing FSMs have been refactored to very small and clear state machines covering only basic actions, like `move-to-location`, `perceive-object`, `grasp-object` or `place-object`. For a particular task, the planner then generates a sequence of those actions in order to achieve the overall goal. Finally, this plan is being executed and monitored. In case of a failure during one of the actions, replanning is being triggered and a new plan is generated based on the current information available in the *knowledge base*.

By a utilizing a task planner instead of FSMs, the maintenance has become easier due to the fact that only small state machines need to be modified or tested. Further, the previous FSMs designer does not need to consider and construct all possible error cases by hand.

8 Conclusion

In this paper we presented several modifications applied to the standard youBot hardware configuration as well as the functional core components of our current software architecture. Besides the development of new functionality, we also focus on developing components in such a manner that they are robot independent and can be reused for a wide range of other robots with even a different hardware configuration. We applied the component-oriented development approach defined

in BRICS [7] for creating our software which resulted in high feasibility when several heterogeneous components are composed into a complete system.

Acknowledgement

We gratefully acknowledge the continued support of the team by the b-it Bonn-Aachen International Center for Information Technology and the Bonn-Rhein-Sieg University of Applied Sciences. Finally, we also acknowledge the support by our sponsors MathWorks, SolidWorks and igus GmbH.

References

1. Intel RealSense F200 camera. `https://software.intel.com/en-us/RealSense/F200Camera`. (Online: 26.08.2015.).
2. KUKA youBot. `http://www.youbot-store.com`. (Online: 26.08.2015.).
3. LUHbots - RoboCup@Work team. `http://www.luhbots.de`. (Online: 26.08.2015.).
4. MoveIt! motion planning. `http://moveit.ros.org`. (Online: 26.08.2015.).
5. Software repository of b-it-bots team used at RoboCup 2014 in Brazil. `https://github.com/mas-group/robocup-at-work`. (Online: 26.08.2015.).
6. Technical drawings of BRSU youBot modifications. `https://github.com/mas-group/technical_drawings`. (Online: 26.08.2015.).
7. R. Bischoff, T. Guhl, E. Prassler, W. Nowak, G. Kraetzschmar, H. Bruyninckx, P. Soetens, M. Haegele, A. Pott, P. Breedveld, J. Broenink, D. Brugali, and N. Tomatis. Brics - best practice in robotics. In *In Proceedings of the IFR International Symposium on Robotics (ISR 2010)*, Munich, Germany., June 2010.
8. Michael Katz and Hoffmann Jörg. Mercury planner: Pushing the limits of partial delete relaxation. In *International Planning Competition (IPC)*, pages 43–47, 2014.
9. Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
10. U. Reiser, C. Connette, J. Fischer, J. Kubacki, A. Bubeck, F. Weisshardt, T. Jacobs, C. Parlitz, M. Hagele, and A. Verl. Care-o-bot 3 - creating a product vision for service robot applications by integrating design and technology. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1992–1998, Oct 2009.