b-it-bots: Our Approach for Autonomous Robotics in Industrial Environments

Abhishek Padalkar, Mohammad Wasil, Shweta Mahajan, Ramesh Kumar, Dharmin Bakaraniya, Raghuvir Shirodkar, Heruka Andradi, Deepan Padmanabhan, Carlo Wiesse, Ahmed Abdelrahman, Sushant Chavan, Naresh Gurulingan, Deebul Nair, Santosh Thoduka(✉), Iman Awaad, Sven Schneider, Paul G. Plöger, and Gerhard K. Kraetzschmar

Department of Computer Science, Hochschule Bonn-Rhein-Sieg, Grantham-Allee 20, 53757 Sankt Augustin, Germany
{abhishek.padalkar,mwasil.wasil,shweta.mahajan,kumar.kumar, dharmin.bakaraniya,raghuvir.shirodkar,heruka.andradi,deepan.padmanabhan, carlo.wiesse,ahmed.abdelrahman,sushant.chavan, naresh.gurulingan}@smail.inf.h-brs.de, {deebul.nair,santosh.thoduka,iman.awaad,sven.schneider, paul.ploeger,gerhard.kraetzschmar}@h-brs.de
http://www.b-it-bots.de

Abstract. This paper presents the approach of our team, b-it-bots, in the RoboCup@Work competition which resulted in us winning the World Championship in Sydney in 2019. We describe our current hardware, including modifications made to the KUKA youBot, the underlying software framework and components developed for navigation, manipulation, perception and task planning for scenarios in industrial environments. Our combined 2D and 3D approach for object recognition has improved robustness and performance compared to previous years, and our task planning framework has moved us away from large state machines for high-level control. Future work includes closing the perception-manipulation loop for more robust grasping. Our open-source repository is available at https://github.com/b-it-bots/mas_industrial_robotics.

1 Introduction

The RoboCup@Work league was established in 2012 as a demonstration league, and became a regular league in 2016. It is aimed at robots working autonomously in industrial environments in collaboration with humans. A typical task involves transporting objects from one location to another, using skills such as autonomous navigation, object perception, manipulation and task planning.

A. Padalkar, M. Wasil, S. Mahajan, R. Kumar, D. Bakaraniya, R. Shirodkar, H. Andradi, D. Padmanabhan, C. Wiesse—Current team member.

S. Chalup et al. (Eds.): RoboCup 2019, LNAI 11531, pp. 591–602, 2019.
https://doi.org/10.1007/978-3-030-35699-6_48

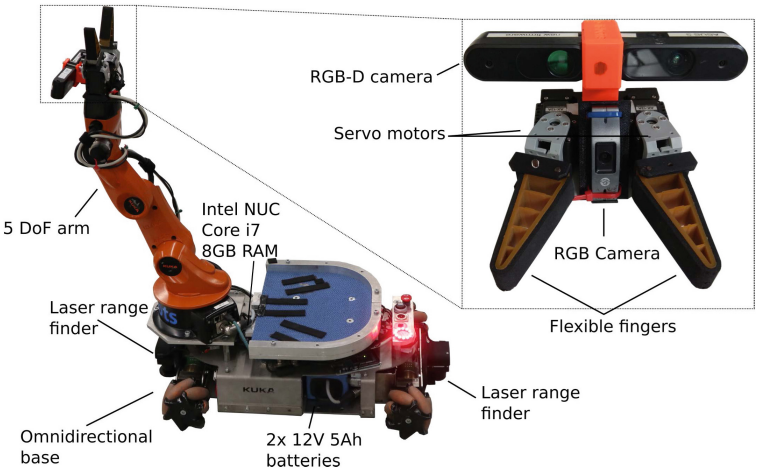

Fig. 1. b-it-bots robot configuration based on the KUKA youBot

Our team, b-it-bots, has participated in RoboCup@Work since its inception, and has regularly performed well at both the World Championships and RoboCup German Open. This year, our team achieved the first place after a closely fought competition with teams from Germany, Singapore, Iran and the Netherlands.

The team consists of Master of Science in Autonomous Systems students from Hochschule Bonn-Rhein-Sieg, who are advised by PhD students and professors. Although RoboCup activities are not part of the academic curriculum, the students participate voluntarily and results from class projects, research and development projects and Master's theses have been deployed on the robot over several years. Collaboration between team members is achieved by regular team meetings every Friday, development and issue tracking on Github and development sprints before competitions. Our main research interests include mobile manipulation in industrial settings, navigation in unconstrained environments, robot perception, task planning and failure detection and error recovery.

In this paper, we describe our approach for the RoboCup@Work competition including our robot platform in Sect. 2, software framework in Sect. 3, navigation in Sect. 4, our approach for different perception tasks in Sect. 5, manipulation and control in Sect. 6 and task planning in Sect. 7.

2 Robot Platform

As most other teams in the league, we use the KUKA youBot as the applied platform for RoboCup@Work (see Fig. 1). It is equipped with a 5-DoF manipulator, a two finger gripper and an omni-directional platform. The standard internal computer of the youBot has been replaced with an Intel NUC with a

Core i7 processor and 8 GB of RAM. Two Hokuyo URG-04LX[1] laser range finders are mounted vertically flipped in the front and the back of the platform. For perception-related tasks, the manipulator is fitted with an ASUS Xtion Pro Live[2] RGB-D camera on the gripper palm. Additionally, an RGB camera is fitted between the fingers of the gripper for close-range perception of objects.

Another custom modification has been made for the youBot gripper (based on the design of [6]) which enhances the opening range and enables grasping of a wider range of objects. The gripper (see Fig. 1) is actuated with two Dynamixel AX-12A servo motors which provide a position interface and force-feedback information. A final modification was performed on the back platform of the youBot. It has been replaced with a new light-weight aluminium version, the position of which can be manually adjusted forward and backward. All technical drawings to the previously described modifications, as well as various 3D printed sensor mounts have been made public[3].

3 Robot Software Framework

We use the Robot Operating System (ROS) [15] as the middleware to pass data and events between components. We primarily use topics since they support non-blocking communication and the possibility of monitoring the communication between two or more nodes at any time. We have standardized our nodes with the addition of *event in* and *event out* topics. Our components listen to the *event in* topic and expect simple command messages for starting, stopping or triggering (run once) the component. 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, following a similar approach as proposed in GenoM3 [13]. The software is organized in an hierarchical manner (see Fig. 2) with the low level components implemented as ROS nodes. Higher level actions are developed as SMACH [10] state machines wrapped around ROS action servers, and include actions such as *move base to location*, *perceive location*, *pick object*, and *place object*. For task planning, the Mercury task planner [11] is used within the framework of ROSPlan, with actions in the task plan corresponding to the actions mentioned earlier.

4 Navigation

For navigation, we use *move_base* from the ROS navigation stack, along with accompanying components such as the occupancy grid map, global and local

[1] https://www.hokuyo-aut.jp/search/single.php?serial=166.
[2] https://www.asus.com/us/3D-Sensor/Xtion_PRO_LIVE.
[3] https://github.com/mas-group/technical_drawings.

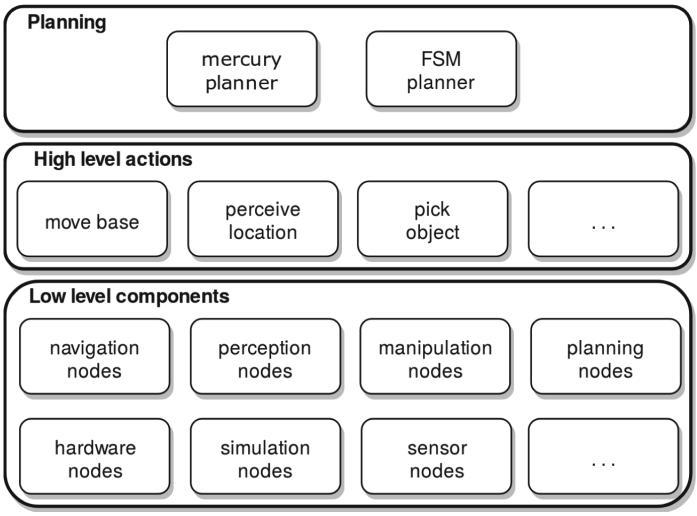

Fig. 2. Software architecture organization

path planner (Dynamic-Window-Approach) and Adaptive Monte Carlo Localization (AMCL). The parameters for *move_base*, such as inflation radius and maximum linear velocity, were configured by running experiments in simulation in various environments. Based on the experiments, we fixed 3 sets of parameters for low, medium and high speed navigation; currently the speed settings are chosen manually based on the complexity of the environment. We use the force-field recovery behaviour developed by smARTLab@Work [6], which moves the robot away from obstacles when it gets stuck.

A direct base controller has been implemented, which allows the robot to move in a local frame relative to its current position without requiring a map or global localization. This is used, for example, for local motions while picking objects from a workstation.

Yellow and black barrier tapes on the floor indicate areas in the environment to be avoided. Using the RGB-D camera, we detect the barrier tape in 2D using a colour filter and convert the pixels to 3D using the point cloud. The point cloud is republished as a laser scan which forms an additional sensor source for the costmap, hence treating the barrier tapes as static obstacles.

5 Perception

Several components have been developed for processing the image and point cloud data from the arm-mounted RGB-D camera. In the current configuration, the RGB-D camera faces downwards when the arm is fully stretched upwards; this allows the robot to perceive the entire area of the workstation.

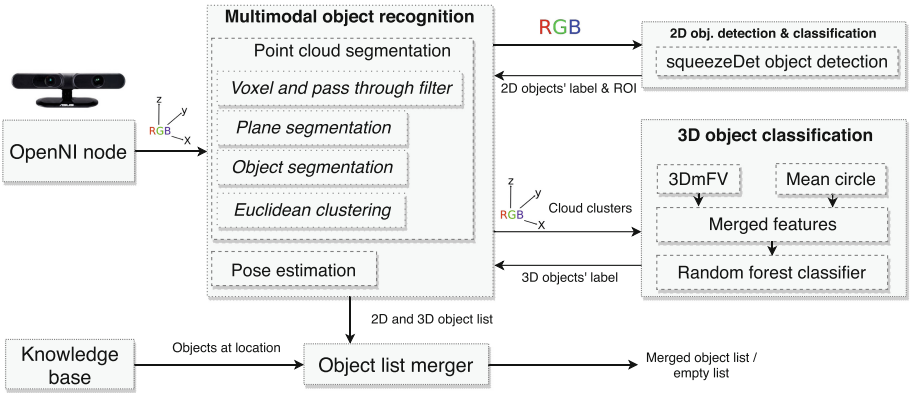

Fig. 3. Object perception pipeline

5.1 Object Recognition

Perception of objects relevant for industrial environments is particularly challenging because they are typically textureless and made of reflective materials such as metal. For the object detection and recognition task, we use the RGB-D camera with both 3D and 2D methods as outlined in Fig. 3.

3D Perception. For 3D segmentation and classification, we capture a single point cloud, downsample it using a voxel grid filter, and crop it using passthrough filters in order to reduce the computational complexity. To perform plane segmentation, we 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. The prism of points above the polygon are segmented and clustered to individual object point clouds.

We extract the features of each cluster using 3D modified Fisher vector (3DmFV) [4] and mean circle features [19]. The 3DmFV features represent a global description of the point cloud using a generative model which is formulated using Gaussian mixture model (GMM). The mean circle features include the radius of circles fit on slices of the point cloud along all three axes, dimensions of the bounding box, mean colour, etc. The combined 3DmFV and mean circle features are used to learn a random forest classifier as illustrated in Fig. 4. The training data for 3D perception consists of approximately 300 segmented point clouds per object.

2D Perception. In previous years, we used graph-based segmentation [9] to segment objects from the background. However, since the requirement to pick from arbitrary surfaces has been included in the rulebook for RoboCup@Work [2], an approach for object segmentation which does not rely on a uniform background is required. We use SqueezeDet [20], a convolutional neural network for object

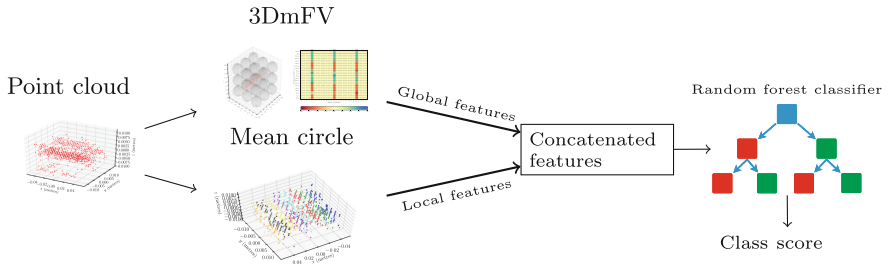

Fig. 4. A visualization of 3DmFV and mean circle features in the point cloud classification. 3DmFV aggregates the global feature of the point cloud while mean circle computes local features represented by slices on three different axes.

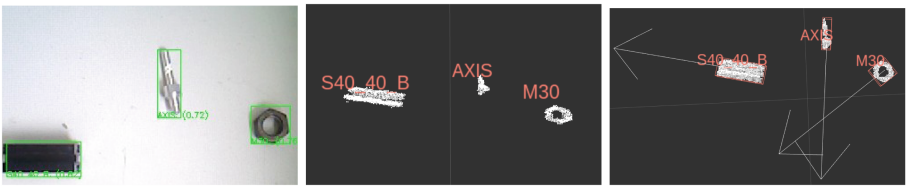

Fig. 5. Recognized objects using 2D (left) and 3D (middle) methods. The right image illustrates the estimated pose of each object.

detection in which the region proposal and classification are integrated into one stage. It uses the `ConvDet` layer to output coordinates for bounding boxes and the class probabilities for each bounding box.

Given the 2D bounding box, we find the corresponding 3D point for each pixel in the box and fit a 3D bounding box to determine the position and orientation of the object. Sample outputs are shown in Fig. 5. The training data for 2D classification consists of approximately 700 annotated images per object.

Inference from both models together runs at approximately 2 FPS. This is achieved by running inference on the two models (squeezeDet and random forest) in parallel. The two object lists are received by the *object list merger* node, which intelligently combines the lists into a unified one. The *object list merger* uses information such as class probabilities from both classifiers, prior knowledge regarding the accuracies for certain objects by each classifier, and inventory at the current location being perceived (which is retrieved from the *knowledge base* used by the task planner as described in Sect. 7).

For the precision placement test, the robot is required to insert objects into cavities. Given an input image of the workstation, canny edge detection is applied to extract the edges around the cavities and the workspace. The resulting edges are dilated and combined into contours, which are filtered based on their area. The cropped cavity images are classified using a MobileNetV2 [17] network. The pose of each cavity is estimated in a similar manner as the 2D objects - by fitting oriented 3D bounding boxes to the corresponding 3D points of the 2D contour.

5.2 Object Tracking

In the rotating table test, the robot is required to pick up a specified list of objects which are placed on top of a rotating table along with several other objects. The table spins at an arbitrary speed within a known range; this is set at the start of the run and remains fixed throughout. The robot needs to track the objects, classify them, and estimate their motion model in order to anticipate their arrival in front of the robot. Once an object has been identified for a grasp, the gripper-mounted RGB camera is used to estimate the precise grasp time.

For tracking, the robot's arm is moved to a predefined configuration such that more than half of the table is within the field of view of its RGB-D camera. At a rate of 10 Hz, 150 point clouds are captured and processed once capturing is complete. For each point cloud, object clusters are extracted in a similar manner as described in Sect. 5.1. In our current approach, we track objects by using the detections from every frame and establish temporal correspondences based on the distance between the centroids of all pairs of objects. This naive approach is possible since there are no occlusions and the objects all rotate at a constant speed. A list of tracked objects is maintained, where each tracked object consists of a list of centroid positions and timestamps.

Given the tracked positions of the objects, a least squares solution for the center of rotation is obtained using SVD, and the angular velocity is estimated as the median of angular velocities for each tracked object. The velocity and center of rotation are used to estimate the time and position at which a given object will arrive in front of the robot. Once an object has been classified and selected for grasping, the arm is positioned such that the gripper is directly above the expected grasp location. In order to determine the exact grasp time, background change detection is performed on the images of the gripper-mounted RGB camera during a small window around the estimated arrival time.

Future improvements aim at incorporating the size and orientation information of the objects for calculation of the exact grasp time using background change detection to allow more flexibility in the placement of objects.

6 Manipulation and Control

We use MoveIt! [1] as our base framework for basic manipulation tasks, but in order to grasp objects reliably, several additional components have been developed and integrated on the robot. We developed our own interpolation-based planner for MoveIt! as it produces more deterministic trajectories for the arm movement. This is used for motions of the arm to pre-defined positions.

For grasping, the grasp pose of the object is the input to the *pre-grasp planner*, which computes a pre-grasp arm configuration based on the type of grasp, a distance offset and constraints imposed by the robot's manipulator and end effector. Finally the trajectories to reach the pre-grasp and grasp pose are concatenated and executed in sequence. Once the end effector reaches the grasp pose, the gripper of the robot is closed. A grasp monitor checks whether the

object is grasped successfully utilizing the force and position feedback of the two Dynamixel motors.

6.1 Cartesian Motion

Given that the manipulator has only 5 degrees of freedom, an exact inverse kinematic solution which satisfies both position and orientation of a particular pose is not always possible. Hence, we deploy a Cartesian motion control framework which can provide an approximate kinematic solution for a particular pose by removing constraints on the orientation. The Cartesian motion control provides joint velocities for target task-space or Cartesian velocities; it relies on an approximate inverse kinematic velocity solution near singularities and joint limits of the arm. The method we use to obtain such inverse kinematic solutions for task space velocities is called Weighted Damped Least Square (WDLS) [8]. Our implementation uses the *WDLS inverse kinematic velocity solver* provided by the OROCOS KDL library. The use of such approximate solutions allows us to manipulate objects in scenarios where positional accuracy is not the primary objective, such as when inserting an object in a box. This method allows us to ignore constraints on specific degrees of freedom (angular degrees of freedom in this case) and the object can be placed in the box by reaching the area above it in any orientation. The Cartesian motion controller also allows the robot to execute motions subject to task-space constraints on the trajectory, such as curve-tracing. The Cartesian controller is an instantaneous incremental motion controller, hence online changes in the target position can be handled easily by the velocity controller.

6.2 Learning from Demonstration

Complex manipulation skills can be transferred to the robot by the learning from demonstration (LfD) framework developed using Dynamic Movement Primitives and Cartesian control. Dynamic movement primitives use nonlinear dynamic equations to learn the shape of the motion [18]. Skills are demonstrated to the robot by a teacher using an arUco marker board; the robot records the motion by tracking the board with a camera and the motion is then learned using the LfD framework [14][4]. This framework was implemented and used to teach the robot motion profiles which are hard to engineer. We plan to use this for teaching motion profiles such as those for picking from or placing on shelves.

7 Task Planning

Finite state machines (FSMs) are sometimes used for high-level control in robotic applications, and in particular in competition settings where the objects involved, locations and the tasks are well-specified and the action set is relatively small. Such FSMs of whole scenarios may, over time, become complex

[4] https://youtu.be/jEtlm96KAbA.

and difficult to understand and maintain. In our architecture, small, easily-understandable, debuggable and maintainable FSMs (in the form of SMACH scripts) are used for executing actions such as *perceive location*, *pick object* and *move base to location*. The high-level control is handled through the use of a task planner which produces a sequence of fully ordered actions to achieve a goal. Listing 1.1 shows a sample plan for transporting a *bearing* from *WS03* to *WS10*.

Listing 1.1. Sample plan

```
(move_base youbot-brsu start ws03)
(perceive youbot-brsu ws03)
(pick youbot-brsu ws03 bearing-00)
(stage youbot-brsu platform_left bearing-00)
(move_base youbot-brsu ws03 ws10)
(unstage youbot-brsu platform_middle
      bearing-00)
(place youbot-brsu ws10 bearing-00)
```

Listing 1.2. PDDL definition for *pick*

```
(:action pick
    :parameters (?r - robot ?l - location ?o -
          object)
    :precondition  (and    (on ?o ?l)
                           (at ?r ?l)
                           (perceived ?l)
                           ( gripper_is_free  ?r)
                           (not (holding ?r ?o))
                           (not (heavy ?o))
                   )
    :effect  (and  (holding ?r ?o)
                   (not (on ?o ?l))
                   (not ( gripper_is_free  ?r))
                   (increase (total-cost) 2)
             )
)
```

We have, for the past couple of years, been using the Mercury task planner [11], a winner at the 2014 International Planning Competition (IPC). Our planning domain contains seven basic operators: *move base to location*, *perceive location*, *pick object*, *stage object to robot platform*, *unstage object from robot platform*, *place object*, and *insert object*. As an example, the PDDL representation of the *pick* operator is shown in Listing 1.2. The limited number of operators also serves to help with tractability, which remains a problem, as is often the case with task planners, in particular given the limits on time in the arena to complete the various tests. In addition to the operators, the domain also includes literals representing locations, the robot, objects and the robot platform (representing the spaces on the robot which are used to transport up to three objects at a time). Eleven predicates such as *on*, *occupied*, *gripper is free* and *perceived* are also included. Further details on the modelling of the domain and the planner's integration into the system can be found in [7] and [12].

As the task to be carried out is received from the referee box, it is parsed and the facts that make up the initial state and the goal statement are added to the knowledge base (KB) within the ROSPlan framework. Due to the time necessary to generate an optimal plan, the *problem generator* chooses three goals to be achieved and outputs the PDDL problem file which is then sent to the planner along with the PDDL domain file containing the operators, their preconditions and effects. The first plan that it may produce is far from optimal. As the planner continues, the quality of the generated plan improves (see [11]). Here, a time limit of 10 s is set. The latest generated plan at that time is then sent to the *plan executor* which schedules each action and dispatches it for execution by triggering the SMACH scripts which are tailored specifically to perform the actions given runtime constraints such as location type and object type. The *plan executor* monitors the action execution and, upon completion, updates the KB

Table 1. Quantitative comparison of task planners for the @Work problem

Planner	Time limit (s)	Plan length	Plan cost	Plan redundancies
Mercury 2014	10	65	570	5
	15	65	570	5
FDSS(2) 2018	10	79	588	0
	15	65	486	0
LAMA 2011	10	57	420	0
	15	56	400	0

to reflect the changes to the world that the action has resulted in (the action's effects), thus ensuring that the current state is available in case any replanning is needed. In the case of a failure during the execution of an action, and depending on the action, the *plan executor* either retries the action or triggers a re-plan.

Parallel execution of base and arm motion has helped to reduce execution time as the robot is able to partially execute part of the following action along with the current one. For example, as the robot is arriving at a workstation (the current action) it may move the arm in preparation for a perceiving action (the following action). This is handled entirely within the state machines which have knowledge of the next action in the plan.

Ideally, the planner should take into consideration the different points awarded for different items/locations in the RoboCup@Work scenarios, the time constraints of the tests and the tendency for certain actions to succeed more often than others. The planning system would then be able to provide a plan that prioritises actions which are highly likely to succeed and high-reward goals. The problem would then be a resource-constrained planning problem with expectation maximisation in a Markov decision process (MDP). MDPs are known to be hard to solve and are, therefore, usually solved offline and assume that all possible situations are in the model. This kind of a solution is not readily available off-the-shelf (and to our knowledge remains an open research topic) and yields a more complex modelling process than what is currently being used in our system.

Looking ahead, and following the results of a comparative study [3] of planners submitted to the 2018 IPC that was carried out to investigate whether to continue with our current planner or switch to another, the LAMA planner [16] will be integrated and tested, mainly due to its ability to produce near-optimal plans in limited time (see Table 1).

8 Current Research Activities

Currently, robustness in perception and grasping of objects is an area for improvement. Our current approach involves perceiving the objects once and performing an open-loop grasp with only minimal grasp verification after the

grasp is complete. We would like to improve this by closing the perception-manipulation loop, incorporating continuous perception before and during the grasp, and continuous grasp monitoring after grasping. We plan to use neural compute sticks such as Intel Movidius[5] or Google Coral[6] to reduce the inference time for object recognition. Additionally, smart machine vision cameras such as JeVois[7] are being considered for use at the end-effector. For manipulation, we aim to achieve coordinated motion between the base and the arm by modeling the entire robot as an 8 DOF manipulator (3 DOF base + 5 DOF arm).

9 Conclusion

In this paper, we presented our approach for the RoboCup@Work competition including modifications applied to the standard youBot hardware configuration as well as the functional core components of our current software architecture. Our combined 2D and 3D approach to object recognition has resulted in improved performance compared to previous years, while keeping up with the increased complexity of the tasks over the years. The task planning framework allows us to easily attempt new types of tasks, without having to deal with large state machines. We applied the component-oriented development approach defined in BRICS [5] for creating our software, allowing us to compose several heterogeneous components into a complete system and even reuse components on different robots. Our current work is focussed on improving the robustness of grasping, by incorporating continuous perception during manipulation and additional sensors for grasp monitoring. Additionally, we eventually aim to move our software to a different platform since the youBot is no longer in production.

Acknowledgement. We value the advice and guidance over the years by Professor Gerhard Kraetzschmar, who sadly passed away this year. He was one of the founders of RoboCup@Work and actively involved in the team activities of b-it-bots. We gratefully acknowledge the continued support of the team by the b-it Bonn-Aachen International Center for Information Technology, Bonn-Rhein-Sieg University of Applied Sciences and AStA H-BRS.

References

1. MoveIt! motion planning. http://moveit.ros.org. Accessed 07 October 2019
2. RoboCup@Work Rulebook 2019. https://atwork.robocup.org/wp-content/uploads/2019/02/Rulebook_2019.pdf. Accessed 05 September 2019
3. Abdelrahman, A., Chavan, S., Tran, N.: Evaluation of New Planners. Unpublished manuscript. Hochschule Bonn-Rhein-Sieg (2019)
4. Ben-Shabat, Y., Lindenbaum, M., Fischer, A.: 3DmFV: three-dimensional point cloud classification in real-time using convolutional neural networks. IEEE Robot. Autom. Lett. **3**(4), 3145–3152 (2018)

[5] https://software.intel.com/en-us/neural-compute-stick.
[6] https://coral.withgoogle.com/products/accelerator/.
[7] http://www.jevois.org/.

5. Bischoff, R., et al.: BRICS - best practice in robotics. In: Proceedings of the IFR International Symposium on Robotics (ISR 2010), Munich, Germany, June 2010
6. Broecker, B., Claes, D., Fossel, J., Tuyls, K.: Winning the RoboCup@Work 2014 competition: the smARTLab approach. In: Bianchi, R.A.C., Akin, H.L., Ramamoorthy, S., Sugiura, K. (eds.) RoboCup 2014. LNCS (LNAI), vol. 8992, pp. 142–154. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18615-3_12
7. Carrion, O.L.: Task planning, execution and monitoring for mobile manipulators in industrial domains. Master's thesis, Bonn-Rhein-Sieg University of Applied Sciences, Grantham Allee 20, 53757 St. Augustin, Germany, April 2016. https://github.com/oscar-lima/isr_planning/blob/kinetic/oscar_lima_master_thesis.pdf
8. Chiaverini, S., Siciliano, B., Egeland, O.: Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator. IEEE Trans. Control Syst. Technol. **2**(2), 123–134 (1994)
9. Felzenszwalb, P.F., Huttenlocher, D.P.: Efficient graph-based image segmentation. Int. J. Comput. Vision **59**(2), 167–181 (2004)
10. Tim Field. SMACH documentation, July 2011. http://www.ros.org/wiki/smach/Documentation
11. Katz, M., Hoffmann, J.: Mercury planner: pushing the limits of partial delete relaxation. In: IPC 2014 Planner Abstracts, pp. 43–47 (2014)
12. Lima, O., Ventura, R., Awaad, I.: Integrating classical planning and real robots in industrial and service robotics domains. In: Proceedings of the 6th Workshop on Planning and Robotics (PlanRob) at ICAPS (2018)
13. Mallet, A., Pasteur, C., Herrb, M., Lemaignan, S., Ingrand, F.: GenoM3: building middleware-independent robotic components. In: 2010 IEEE International Conference on Robotics and Automation, pp. 4627–4632. IEEE (2010)
14. Mitrevski, A., Padalkar, A., Nguyen, M., Plöger, P.G.: "Lucy, take the noodle box!": domestic object manipulation using movement primitives and whole body motion. In: Chalup, S., Niemueller, T., Suthakorn, J., Williams, M.-A. (eds.) RoboCup 2019. LNCS (LNAI), vol. 11531, pp. 189–200. Springer, Cham (2019)
15. Quigley, M., et al.: ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software (2009)
16. Richter, S., Westphal, M., Helmert, M.: LAMA 2008 and 2011. In: International Planning Competition, pp. 117–124 (2011)
17. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C.: MobileNetV2: inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510–4520 (2018)
18. Schaal, S.: Dynamic movement primitives - a framework for motor control in humans and humanoid robotics. In: Kimura, H., Tsuchiya, K., Ishiguro, A., Witte, H. (eds.) Adaptive Motion of Animals and Machines, pp. 261–280. Springer, Tokyo (2006). https://doi.org/10.1007/4-431-31381-8_23
19. Thoduka, S., Pazekha, S., Moriarty, A., Kraetzschmar, G.K.: RGB-D-based features for recognition of textureless objects. In: Behnke, S., Sheh, R., Sarıel, S., Lee, D.D. (eds.) RoboCup 2016. LNCS (LNAI), vol. 9776, pp. 294–305. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68792-6_24
20. Wu, B., Iandola, F., Jin, P.H., Keutzer, K.: SqueezeDet: unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 129–137 (2017)