The b-it-bots@Home 2019 Team Description Paper

Alex Mitrevski Argentina Ortega Sainz Patrick Nagel
Maximilian Schöbel Minh Nguyen Roberto Cai Wu
Rohan Haseloff Paul G. Plöger Gerhard K. Kraetzschmar

October 28, 2018

Abstract. This paper presents the b-it-bots@Home team and one of its mobile service robots called *Lucy* – a Human Support Robot manufactured by Toyota. We present an overview of our robot control architecture and the robot's capabilities, namely the added functionalities from various research and development projects carried out by the Autonomous Systems group at Hochschule Bonn-Rhein-Sieg.

1 Introduction

The b-it-bots@Home team[1] was established in 2007 and functions as part of the international Autonomous Systems master's program at Hochschule Bonn-Rhein-Sieg (HBRS)[2]. Our team consists of bachelor's, master's, and PhD students that are advised by three tenured professors, such that we have a long history of participation at RoboCup@Home competitions.

Our initial robot Johnny, a VolksBot platform, was used by the team from 2008 to 2010. Since 2011, the b-it-bots@Home team has been working with Jenny, a Care-O-Bot 3, and has successfully participated in multiple competitions, including RoboCup, German Open, and RoCKIn. In February 2018, the team added Lucy, a Toyota HSR, to its available platforms, and as of 2019 is participating in the RoboCup@Home Domestic Standard Platform League (DSPL).

While participation at competitions has always been an integral part of our activites, we foster a research-oriented culture above all, such that we have several ongoing PhD and master's theses projects that are very closely related to the team. In parallel, one of our goals is to deploy service robots to real-life applications. Since our research interests are highly linked to our experiences in the field, our software both contributes and benefits from the ongoing research projects taking place at our university.

[1] `https://mas-group.inf.h-brs.de/?page_id=622`
[2] `http://www.h-brs.de`

The rest of this paper presents some of the work carried out by current and previous team members, such that we focus on some of our main research interests: execution monitoring, fault detection and diagnosis, robust manipulation, real-time and adaptive perception, as well as knowledge-based reasoning.

2 Robust Manipulation

2.1 Imitation Learning

Our robots - Lucy included - have generally used MoveIt![3] for planning arm trajectories, but this has often lead to unpredictable and suboptimal behaviour in practical scenarios, such as picking and placing objects. In order to increase the predictability of our robots during manipulation and allow more optimal trajectories to be performed, we are slowly moving towards a manipulation framework based on learning by demonstration. In one recent project, we investigated dynamic motion primitives [1] as a generalisable representation of robot motion trajectories, such that we performed an experimental evaluation of dynamic motion primitives and developed an imitation-based framework based on which Cartesian trajectories are acquired by tracking a marker array; imitation is then achieved by performing synchronised arm and base motions[4]. Our manipulation pipeline still makes use of MoveIt!, but only for simple motions and as a backup for the motion primitives.

2.2 Grasp Representation and Learning

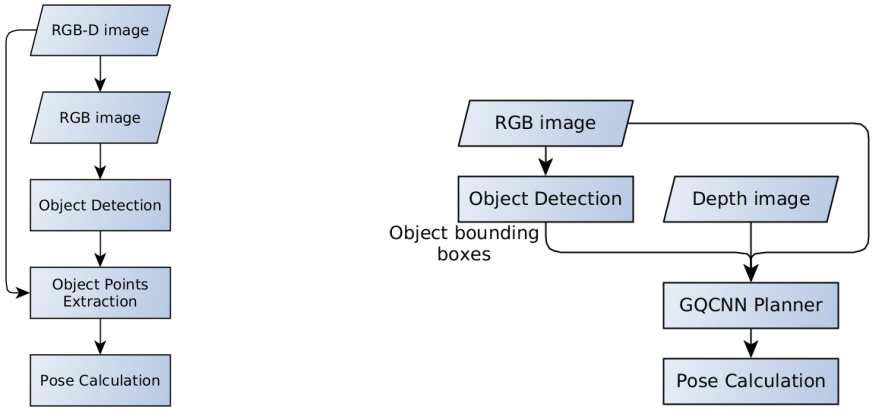
In order to improve the grasp robustness of our robot, one major research interest of the team concerns learning-based grasp synthesis methods. In this context, one of our recent projects looked into four aspects most relevant to data generation for training supervised grasp evaluation models, namely: (i) feature extraction from perceptual data, (ii) object-grasp representation, (iii) grasp evaluation methods, and (iv) data generation techniques. Additionally, we have integrated a complete object grasping pipeline, starting from object detection to grasp pose detection and grasp execution.

For finding a grasp pose for detected objects, a simple pose estimation algorithm and the Grasp Quality CNN (GQCNN) model [2] are integrated (Figure 1). The first method directly uses the object points extracted from an RGB-D cloud to calculate the grasping position; the grasp approach is then chosen to be the positive x-axis in the coordinate frame of the robot base. On the other hand, GQCNN trains a CNN model on the Dex-Net 2.0 synthetic grasp dataset to predict the probability of success of a grasp candidate using a pair of depth and RGB object images. Figure 2 shows a sample grasp detected using GQCNN[5].

[3] `moveit.ros.org`

[4] A video showing some of the preliminary results of this work can be found at `https://www.youtube.com/watch?v=jEtlm96KAbA`.

[5] A video of some grasp experiments performed using the integrated grasping pipeline can be found at `https://www.youtube.com/watch?v=OC7vttt4-Jo`. These experiments also show the integration of the previously described imitation framework.



(a) Object pose estimation (b) GQCNN planner

Fig. 1: Integrated grasp planning methods

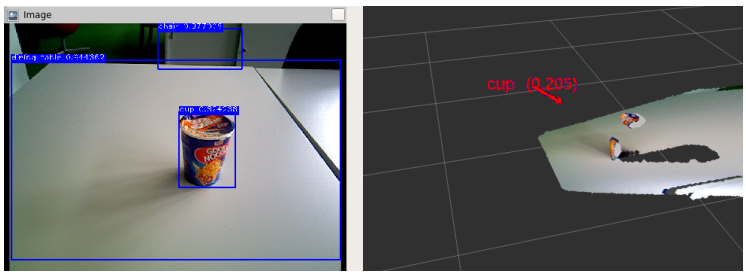


Fig. 2: A successful GQCNN grasp plan. Arrow and number on the right indicate grasp pose and grasp quality returned from the GQCNN planner

3 Object Detection and Scene Understanding

The architecture of our object detection pipeline provides a standard and extensible way of integrating new models and approaches to our code base: new implementations only need to extend the *ImageDetector* base class for performing detection. Our current instantiation of the pipeline first extracts RGB data from an RGB-D point cloud; a Single-shot Multi-box Detector model (SSD) [3], trained on the COCO dataset[6], is then used to detect objects in the image. Independently, a RANSAC algorithm fits a plane model using 3D points downsampled from the RGB-D cloud to find horizontal surfaces (such as tables or shelves) associated with the detected objects. The detection result returned from the SSD model is then used to extract object points from the original RGB-D cloud and estimate the objects' 3D positions. We also provide a `Boost Python` wrapper

[6] `http://cocodataset.org/`

to allow executing point cloud processing functionalities from Python, since the original implementation uses the C++-based Point Cloud Library (PCL)[7].

4 Planning, Reasoning, and Operation Monitoring

4.1 Task Planning

Robots operating in dynamic environments have to be designed to be robust and flexible; we are thus working towards a flexible plan-based architecture for high-level reasoning and recovery. In a first step, we have integrated ROSPlan[8] as a planning framework that covers the whole lifecycle of a task, starting from problem generation, task planning, plan dispatching, and plan monitoring.

In order to extend the plan generation with expert knowledge, we are also working towards integrating the hierarchical task network planner JSHOP2[9] into ROSPlan. The motivation for this comes from the work by Awaad et al. [4], as well as the task planning, execution, and monitoring system developed by Shpieva and Awaad [5], where JSHOP2 has been used successfully.

4.2 High-Level Knowledge Representation and Reasoning

In order to perform purposeful tasks, domestic robots need to be able to understand their environment and reason about it. While several aspects about the world, such as the locations of objects or people, have to be estimated and updated dynamically as a robot is operating, it would be suboptimal to let a robot learn everything about the environment from scratch; instead, it is more pragmatic to guide the robot's reasoning process by using an ontology that represents encyclopedic knowledge about the world (namely known facts about objects, their properties, and relations between each other). In this respect, we are working towards an ontology for domestic environments that is motivated by the KnowRob ontology [6], but represents a stripped-down version of it. Just as KnowRob, our ontology is written in the OWL Web Ontology Language[10], such that we are working on an RDFLib-based[11] interface for interacting with the encoded knowledge.

4.3 Execution and Component Monitoring

Due to the complexity of domestic environments, such as the high variability of objects and the presence of other agents, domestic robots are quite failure-prone. This raises the need for both appropriate recovery strategies during execution as well as learning mechanisms for improving the execution process, but also means

[7] `http://docs.pointclouds.org/`
[8] `https://github.com/KCL-Planning/ROSPlan`
[9] `http://www.cs.umd.edu/projects/shop/description.html`
[10] `www.w3.org/TR/owl-ref/`
[11] `https://github.com/RDFLib/rdflib`

that robots need to be transparent about their actions so that their policies can be understood more easily. In order to model and express action execution knowledge, we are using an action execution library[12] for (i) representing knowledge about actions, namely their inputs, outputs, and known failure cases and (ii) logging execution-relevant data. Motivated by the work in [7] and [8], we are currently using this library for executing placing actions. In addition to that, we are in the process of adapting the component monitoring framework described in [9] so that component failures can be detected early enough, which should allow us to prevent undesired events (e.g. the robot colliding with a table due to a malfunctioning arm joint) and, if automatic recovery is not possible, communicate such failures to a human operator.

5 Natural Language Processing and Understanding

For robots in domestic environments, interaction with humans, particularly in a verbal manner, is an indispensable component. This section provides an overview of how we are addressing the challenge of understanding humans and responding to them.

5.1 Speech recognition

For detecting speech from an audio snippet and transforming it into a machine-readable format, we differentiate between online and offline methods. Online speech recognition is conducted using Google's speech recognition API [13], thereby leveraging the model's high recognition rates and robustness; however, since we cannot assume that our robot will always have a stable internet connection, we have also integrated PocketSphinx[14], a speech recognition library developed by Carnegie Mellon University, for understanding commands even when our robot is offline. The results of a project that has compared open source speech recognition toolkits for domestic environments[15] point out that the speech recognition toolkit Kaldi [10] is more suitable for everyday household tasks; we are thus in the process of switching from PocketSphinx to Kaldi as our primary offline speech recognition tool.

5.2 Speech matching

Even with a highly accurate speech recognition model, recognised speech might be faulty or incomplete, which generally means that it needs to be further processed by comparing what was recognised with what a robot already knows (such

[12] `https://github.com/b-it-bots/action-execution`
[13] `https://cloud.google.com/speech-to-text/`
[14] `https://github.com/cmusphinx/pocketsphinx`
[15] Conducted as part of the RoboLand project: https://www.h-brs.de/de/roboland-telepraesenz-roboter-im-haeuslichen-lebens-und-pflegearrangement-von-personen-mit-demenz-im

as a database of known questions and commands) and react to that accordingly. For comparing recognised and known speech, we use the *Levenshtein distance*, which is a string comparison metric that measures the difference between two sentences; if the similarity between the recognised speech and a sentence in a previously created database is above a certain threshold, the robot would reply in case of a question or execute a corresponding action in case of a command.

Although the Levenshtein distance provides acceptable practical performance, we have additionally implemented *Double Metaphone*, which is an approach for indexing words by sound, as an additional speech verification method. This allows us to refer to the results of two different methods, thereby increasing the robustness of the system even if the speech recognition was only partially correct.

5.3 Natural Language Understanting

As part of an ongoing project, we are also investigating natural language processing toolkits for use in our robots. The project is still ongoing, but some preliminary results suggest that SocRob's NLU ROS package[16] is a promising candidate, so we have started integrating it for further testing.

6 Face and Expression Recognition

We use a real-time vision system[17] for performing the tasks of face detection, gender classification, and emotion classification simultaneously in a single step. The facial expression recognition system present on our robot can recognise expressions of joy, surprise, sadness, neutrality, and anger. This component is also being used by the SocRob team in the RoboCup@Home OPL league[18]

Our approach consists of creating a Convolutional Neural Network (CNN) architecture for emotion recognition, with accuracies of 96% on the IMDB gender dataset and 66% on the FER-2013 emotion dataset.

We have additionally developed a real-time guided back-propagation visualization technique that displays the dynamics of the weight changes and evaluates the learned features.

[16] `https://github.com/socrob/mbot_natural_language_processing`
[17] `https://github.com/oarriaga/face_classification`
[18] `https://github.com/socrob/face_classification`

7 Open-Source Contributions

In the last year, we open-sourced most of our code and made it available through our official GitHub organization[19]. By making our source code public, we expect that our contributions can be adopted by the community and we can cooperate with other teams more easily.

Due to the nature of our contract with Toyota, our HSR-specific code is not on GitHub, but all of our core robot-independent software can be found there:

- **mas_domestic_robotics**: Core robot-independent components for domestic applications, which are currently used by our Care-O-Bot and Toyota HSR
- **mas_execution_manager**: A small library for creating state machines and managing their execution
- **mas_common_robotics:** Robot-independent ROS packages shared with our b-it-bots@work team
- **mas_perception**: Our robot-independent perception components
- **mas_navigation**: Our robot-independent navigation stack

Our wiki[20] documenting our development process and many of our functionalities is also publicly available.

Our organization has also open-sourced our Care-O-Bot software[21] as well as other open-source repositories to which our current and past members have been contributing. A few highlights can be found below:

- An implementation of the execution models described by Mitrevski et al. [8] and an Unreal Engine simulation for reproducing some of the results presented there[22]
- The motion detection framework described by Thoduka et al. [11][23]
- A ROS-based implementation of the tactile slip detector described by Sanchez et al. [12][24]

8 Conclusions and future work

This paper presented the robot platform of the b-it-bots@Home team and its capabilities. While the described functionalities have already been integrated on our platform, integration is a continuous process driven by our research goals, which are reflected through several ongoing PhD projects, master's theses, and funded projects. Our current focus areas go in line with all aspects described in this paper and include long-term experience acquisition, skill generalisation, transparent execution, communicating robot intentions, large-scale and multi-floor mapping, as well as autonomous exploration, all of which are particularly important in the context of domestic robotics.

[19] `https://github.com/b-it-bots`
[20] `https://github.com/b-it-bots/wiki`
[21] `https://github.com/b-it-bots/mas_cob`
[22] `https://github.com/alex-mitrevski/delta-execution-models`
[23] `https://github.com/sthoduka/fmt_motion_detection`
[24] `https://github.com/mas-group/tactile_slip_detector`

Acknowledgement

We gratefully acknowledge the continued support of the team by the b-it Bonn-Aachen International Center for Information Technology and Hochschule Bonn-Rhein-Sieg.

References

1. A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors. *Neural Computation*, 25(2):328–373, 2013.
2. J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg. Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics. *CoRR*, abs/1703.09312, 2017.
3. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A. C. Berg. SSD: Single Shot MultiBox Detector. In *Computer Vision – ECCV 2016*, pages 21–37, 2016.
4. I. Awaad, G. K. Kraetzschmar, and J. Hertzberg. The role of functional affordances in socializing robots. *International Journal of Social Robotics*, 7(4):421–438, March 2015.
5. E. Shpieva and I. Awaad. Integrating Task Planning, Execution and Monitoring for a Domestic Service Robot. *Information Technology*, 57(2):112–121, March 2015.
6. M. Tenorth and M. Beetz. KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots. *Int. Journal of Robotics Research (IJRR)*, 32(5):566–590, Apr. 2013.
7. A. Kuestenmacher, N. Akhtar, P. G. Plöger, and G. Lakemeyer. Towards Robust Task Execution for Domestic Service Robots. In *Journal of Intelligent & Robotic Systems*, volume 76, pages 5–33, September 2014.
8. A. Mitrevski, A. Kuestenmacher, S. Thoduka, and P. G. Plöger. Improving the Reliability of Service Robots in the Presence of External Faults by Learning Action Execution Models. In *Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4256–4263, 2017.
9. A. Mitrevski, S. Thoduka, A. Ortega Sáinz, M. Schöbel, P. Nagel, P. G. Plöger, and E. Prassler. Deploying robots in everyday environments: Towards dependable and practical robotic systems. In *29th Int. Workshop Principles of Diagnosis DX'18*, 2018.
10. D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely. The kaldi speech recognition toolkit. In *IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 1–4. IEEE Signal Processing Society, 2011.
11. S. Thoduka, F. Hegger, G. K. Kraetzschmar, and P. G. Plöger. Motion Detection in the Presence of Egomotion Using the Fourier-Mellin Transform. In *Proceedings of the 21st RoboCup International Symposium*, 2017.
12. J. Sanchez, S. Schneider, N. Hochgeschwender, G. K. Kraetzschmar, and P. G. Plöger. Context-Based Adaptation of In-Hand Slip Detection for Service Robots. In *Proceedings of the IFAC Symposium on Intelligent Autonomous Vehicles (IAV)*, 2016.
13. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

Alex Mitrevski, Argentina Ortega Sainz, Patrick Nagel, Maximilian Schöbel, Minh Nguyen, Roberto Cai Wu, Rohan Haseloff, Paul G. Plöger, Gerhard K. Kraetzschmar

Lucy (Toyota HSR) Software and External Devices

We use a standard Human Support Robot (HSR) from *Toyota*. No modifications have been applied.

Fig. 3: Lucy: a Toyota HSR

Robot's Software Description

For our robot, we are using the following software:

- *Platform*: Robot Operating Systems (ROS) [13]
- *Navigation*: Built-in ROS-based functionalities provided by Toyota
- *Arm control*: In-house imitation learning framework and MoveIt![25]
- *Object recognition*: Single-shot Multi-box Detector (SSD) [3]
- *Speech recognition*: Google Speech (online), PocketSphinx and Kaldi (offline)
- *Natural language processing*: SocRob's NLU[26]
- *Gender recognition*: In-house CNN model[27]

Most of our software is publicly available at `https://github.com/b-it-bots`.

External Devices

Lucy relies on the following external hardware:

- Alienware 15", Intel Core i9 processor and GTX 1080

Cloud Services

Lucy connects to the following cloud services:

- Speech recognition: Google Speech[28]

[25] `moveit.ros.org`
[26] `https://github.com/socrob/mbot_natural_language_processing`
[27] `https://github.com/oarriaga/face_classification`
[28] `https://cloud.google.com/speech-to-text/`