The b-it-bots@Home 2022 Team Description Paper: DSPL Simulation

Alex Mitrevski Ahmed Abdelrahman Munaib Al-Helali
Sushant Chavan Minh Nguyen Paul G. Plöger Arun Prabhu

January 22, 2022

Abstract. This paper presents the b-it-bots@Home team and its domestic mobile service robot called *Lucy*, a Toyota Human Support Robot. We present an overview of our robot architecture and the robot's capabilities, particularly focusing on functionalities currently used by our robot, which have been developed by the team members as well as in various research 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], such that our overall goal is the deployment of service robots to real-life applications. Our team consists of master's and PhD students that are advised by four tenured professors; since our research interests are related to our experience in this field, our software both contributes and benefits from various ongoing research projects taking place at our university.

Our team has considerable prior experience of participating in robot competitions. Our initial robot Johnny, a VolksBot platform, was used by the team from 2008 to 2010. Between 2011 and 2018, the team participated in multiple competitions with Jenny, a Care-O-Bot 3. Since 2018, we work with Lucy, a Toyota Human Support Robot (HSR), with which we participate in the RoboCup@Home Domestic Standard Platform League (DSPL). Starting from 2020, we have been participating in RoboCup@Home Simulation DSPL, in particular Japan Open in 2020 and the RoboCup World Cup in 2021.

In this paper, we present some of the components currently used on our robot, such that we focus on manipulation, object perception and scene understanding, as well as knowledge-based reasoning.

[1] `b-it-bots.de`
[2] `http://www.h-brs.de`

2 Manipulation

2.1 Manipulation Trajectory Representation and Execution

To increase the predictability of our physical robots during manipulation, we acquire manipulation trajectories using learning by demonstration. In particular, we use dynamic motion primitives (DMPs) [1] to represent Cartesian motion trajectories, which are acquired by tracking a marker array. As the reachability of the HSR's manipulator is limited, we synchronise arm and base motions when imitating demonstrated trajectories, as described in [2]. When using demonstrated trajectories, we additionally use MoveIt![3] for simple motions between predefined positions, such as moving to a pregrasp position or retrieving the arm back after executing a demonstrated trajectory.

Since the execution of demonstrated trajectories requires base motions, a rather accurate robot odometry is needed for preventing base drift. In the case of the HSR's simulation, this has been problematic since the odometry is less accurate than on the physical robot; the inaccurate odometry leads to significant base drift, particularly for manipulation goals that require considerable base motion. For this purpose, we primarily use MoveIt! in simulation, namely trajectory planning is performed during execution; this has the disadvantage of increasing the execution time, but the execution error is generally lower than with our controller for executing DMPs.

2.2 Object Grasping

To grasp objects of varied shapes and sizes, we use an adaptive strategy that chooses either a sideways or a top-down grasp for a given object. We particularly use a heuristic according to which a top-down grasp is selected if the height of an object, which is estimated based on the object's 3D bounding box, is below a predefined threshold; otherwise, a sideways grasp is selected. To align the robot's gripper for grasping, the orientation of an object is determined as the direction of the object's principal component when a top-down grasp is used, and as the object's planar orientation in the case of a sideways grasp.

We are additionally working towards a learning-based method [3], which has been verified for a small group of objects and does not require hand-coded heuristics for choosing a grasping strategy. This method uses dedicated grasping models that have been learned for particular objects or object classes; when generalising to new object classes, information about object similarity as encoded in an object ontology is combined with any prior generalisation experiences.

3 Object Detection and Scene Understanding

3.1 Object and Plane Detection

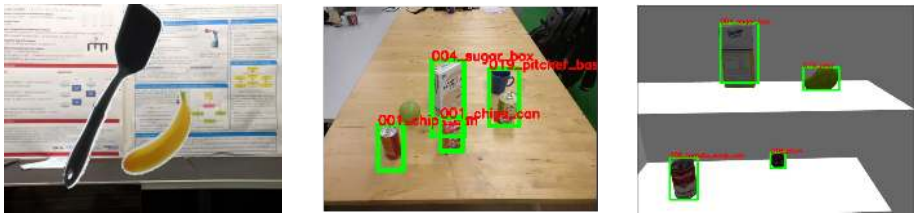
We detect 3D objects in the robot's view using an algorithm that processes an RGB-D point cloud. Detections from the algorithm can be used to avoid small

[3] `https://moveit.ros.org`

objects lying on the floor, which are undetected by a 2D range sensor during navigation, or to obtain the poses of objects to be picked up. The algorithm expects a few parameters, such as the ground plane height, the maximum size of objects to be detected, and the minimum closeness to occupied cells in an occupancy grid map; these help in filtering out large objects, such as tables and walls, as well as objects that are very close to or touching large objects. We use Eucledian clustering on the filtered and downsampled point cloud data to detect objects. The algorithm can also be configured to maintain a cache of previously detected objects for a certain duration of time; this particularly helps in avoiding obstacles on the ground after they have gone out of the robot's view during navigation. The algorithm is implemented using the Point Cloud Library (PCL).[4,5] For plane detection, we use the RANSAC algorithm [4] to fit a plane model to detected objects and find horizontal surfaces, such as tables or shelves.

3.2 Object Recognition

Our object recognition procedure first extracts an RGB image from the previously detected objects. A Faster R-CNN model [5], pre-trained on the COCO dataset[6] and finetuned on the YCB standard object data set [6], is then used to detect objects in the image. For finetuning, we perform data augmentation by projecting objects on various backgrounds and applying a range of transformations to them, such as translation and rotation, resizing, as well as colour modifications. The augmentation and object detection are illustrated in Fig. 1.



(a) Example training data augmentation

(b) Example real-world object detection

(c) Object detection in simulation

Fig. 1: Illustration of our data augmentation and object detection

3.3 Free Space Detection

When a robot is tasked with a series of picking and placing multiple objects on a planar surface, objects should not be placed one on top of another to avoid the

[4] http://docs.pointclouds.org/

[5] https://github.com/b-it-bots/mas_domestic_robotics/tree/kinetic/mdr_perception/mdr_cloud_object_detection

[6] https://cocodataset.org/

risk of them toppling over. This requires that the robot is able to identify free space on a planar surface where a picked up object can be placed. To achieve this, we implement an algorithm that works together with the object detector. The algorithm divides the placing surface into a grid of cells with a configurable cell size; the robot then performs object detection on the planar surface, marks all cells that have a certain overlap ratio with the detected objects as occupied, and finally identifies a candidate free cell for placing an object using a heuristic, such as a free cell that is furthest to the back of the placing surface. This algorithm is based on an assumption that the cell size is large enough to hold most objects to be placed. The free space detection procedure is illustrated in Fig. 2.

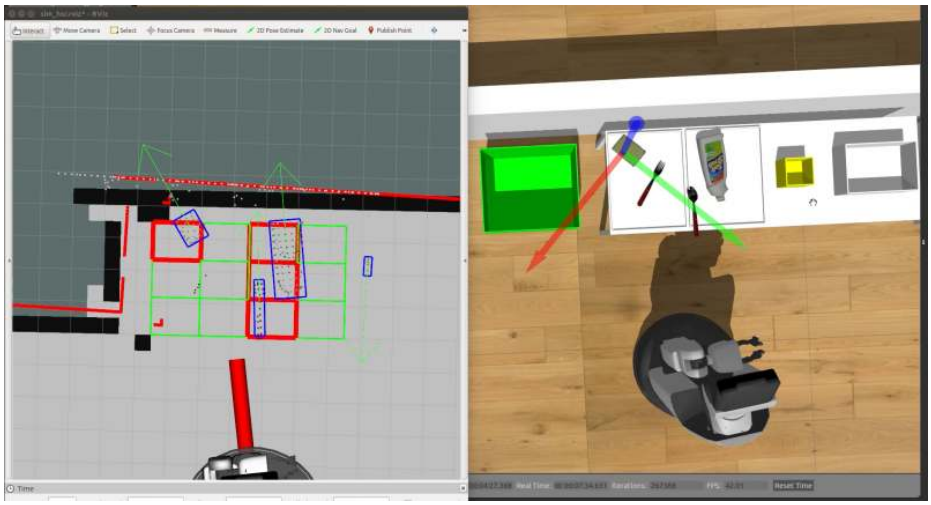


Fig. 2: Illustration of the free space detection procedure in the HSR simulation

4 Knowledge Representation, Planning, and Execution

4.1 High-Level Knowledge Representation and Reasoning

For representing encyclopedic knowledge about the world and in order to be able to perform automatic reasoning for additional pieces of knowledge, we use an ontology for domestic environments, which is a stripped-down version of KnowRob [7] and is thus also written in the OWL 2 Web Ontology Language.[7] We presently use an RDFLib-based[8] interface for interacting with the encoded knowledge; however, we are also working towards integrating the Owlready2 [8] Python package, which, in addition to facilitating access to the ontological knowledge,

[7] `https://www.w3.org/TR/owl2-overview/`
[8] `https://github.com/RDFLib/rdflib`

supports dynamically loading imported ontologies, thereby enabling the development of modular ontologies, as well as inferring new knowledge using reasoners such as HermiT[9] or Pellet [9]. Axioms needed for inferring such new knowledge can also be encoded in the OWL2 ontology as SWRL (Semantic Web Rule Language) rules; additionally, knowledge encoded in the ontology can be utilised during task planning.

4.2 Knowledge Visualisation

While having a dynamically updated knowledge base of environmental entities such as objects and people is beneficial to a robot, it is also necessary to be able to visualise this knowledge for monitoring or debugging the robot's operation. For this purpose, we have developed a simple RViz plugin[10] that interacts with the robot's knowledge base, which is stored in a MongoDB database using mongodb_store[11], and visualises the physical entities from the knowledge base as 3D models in RViz. The plugin, which is illustrated in Fig. 3, enables an operator to selectively visualise individual objects or entire classes of objects. Although we presently use a MongoDB database for storing the robot's knowledge, the plugin provides flexible interfaces that can be extended to work with different types of databases used to store knowledge.

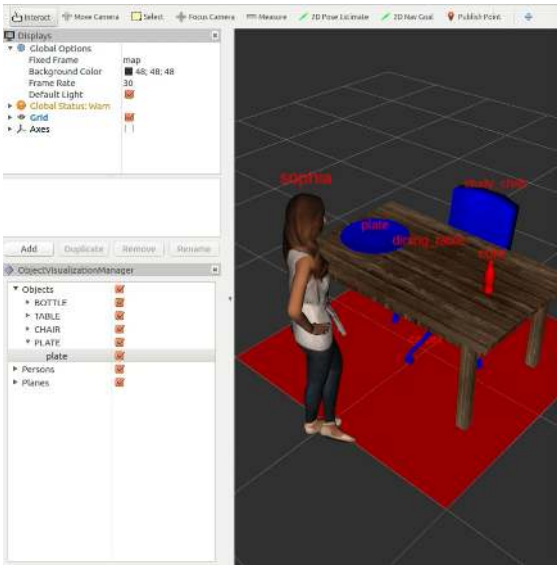


Fig. 3: Illustration of our knowledge base visualisation plugin

[9] http://www.hermit-reasoner.com/
[10] https://github.com/b-it-bots/rviz_3d_object_visualizer
[11] https://github.com/strands-project/mongodb_store

4.3 Task Planning and Execution

For increased robot autonomy and flexibility, we use a task planner in our architecture. We have particularly integrated ROSPlan [10], which allows problem generation, task planning, plan dispatching, and plan monitoring to be performed within a unified framework. After a detailed comparative evaluation of planners submitted to the "Satisficing" track of the 2018 edition of the International Planning Competition[12], we have integrated the LAMA planner [11] into ROSPlan for task planning. This planner is based on the Fast Downward classical planning system[13] and generates plans in an *anytime* fashion, thereby allowing the best possible plan to be obtained within a specified time-limit.

In addition to using a task planner, we make use of state machines for rapid prototyping as well as implementing complex robot behaviours. The state machine specification described in [12] is used for that purpose.

5 Open-Source Contributions

Our team is committed to making our work accessible to the robotics community; most of our code is thus available through our official GitHub organization.[14] Our HSR-specific code is not publicly available to the nature of our contract with Toyota, but all of our core robot-independent software can be found there, including the following major components:

- **mas_domestic_robotics**: Core robot-independent components for domestic applications (`https://github.com/b-it-bots/mas_domestic_robotics`)
- **mas_execution_manager**: A library for creating state machines and managing their execution [12] (`https://github.com/b-it-bots/mas_execution_manager`)
- **mas_knowledge_base**: A library exposing interfaces for interacting with an OWL ontology, the ROSPlan knowledge base, and mongodb_store (`https://github.com/b-it-bots/mas_knowledge_base`)
- **mas_perception_libs**: Robot-independent perception components, including a Boost Python wrapper to allow executing point cloud processing functionalities from Python, since the original implementation uses the C++-based PCL (`https://github.com/b-it-bots/mas_perception_libs`)
- **ros_dmp**: A library for learning and executing dynamic motion primitives (`https://github.com/b-it-bots/ros_dmp`)
- **dataset_interface**: A library with various utilites for data augmention, object detection, and face recognition (`https://github.com/b-it-bots/dataset_interface`)
- **explainable-robot-execution-models**: An implementation of the learning-based object grasping algorithm described above (`https://github.com/alex-mitrevski/explainable-robot-execution-models`)

[12] `https://ipc2018.bitbucket.io`
[13] `https://www.fast-downward.org/`
[14] `https://github.com/b-it-bots`

– **ftsm**: A library for generically implementing components with a so-called fault tolerant state machine (`https://github.com/ropod-project/ftsm`)

A diagram showing the interaction between our main open-source components is shown in Fig. 4. We additionally have a list of freely available tutorials for using some of our components.[15]

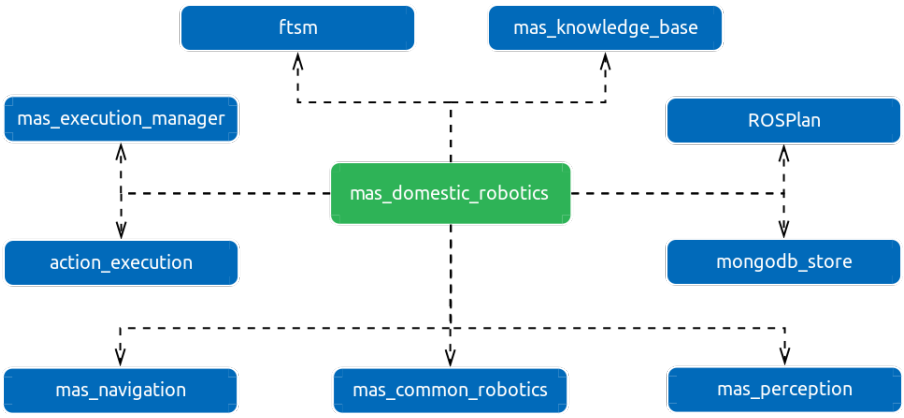


Fig. 4: Various components developed and/or used by our team[16]

6 Conclusions and Future Work

This paper presented the b-it-bots@Home team and various functionalities that are in active use and development by the team. While the described functionalities have already been integrated on the HSR and in its simulation, the integration and development of new functionalities is a continuous process driven by our research goals, which are reflected through several ongoing PhD projects, master's theses, and funded projects. Our ongoing work is aligned with the aspects described in this paper and includes long-term experience acquisition, execution monitoring, as well as communicating robot intentions for transparent execution, aspects which are particularly important for increasing the practical acceptance of domestic robotics.

Acknowledgement

The activities of the b-it-bots@Home team are supported by the Bonn-Aachen International Center for Information Technology (b-it) and Hochschule Bonn-Rhein-Sieg.

[15] `https://github.com/b-it-bots/mas_tutorials`
[16] Diagram taken from `https://github.com/b-it-bots/mas_domestic_robotics`

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. A. Mitrevski, A. Padalkar, M. Nguyen, and P. G. Plöger. "Lucy, Take the Noodle Box!": Domestic Object Manipulation Using Movement Primitives and Whole Body Motion. In *Proceedings of the 23rd RoboCup International Symposium*, 2019.
3. A. Mitrevski, P. G. Plöger, and G. Lakemeyer. Ontology-Assisted Generalisation of Robot Action Execution Knowledge. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pages 6763–6770, 2021.
4. M. A. Fischler and R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, June 1981.
5. S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*, abs/1506.01497, 2015.
6. B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. Benchmarking in Manipulation Research: Using the Yale-CMU-Berkeley Object and Model Set. *IEEE Robotics Automation Magazine*, 22(3):36–52, Sept. 2015.
7. 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.
8. J.-B. Lamy. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial Intelligence in Medicine*, 80:11–28, 2017.
9. E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007.
10. M. Cashmore et al. ROSPlan: Planning in the Robot Operating System. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 333–341, 2015.
11. S. Richter and M. Westphal. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal Of Artificial Intelligence Research*, 39:127–177, 2010.
12. A. Mitrevski and P. G. Plöger. Reusable Specification of State Machines for Rapid Robot Functionality Prototyping. In *Proceedings of the 23rd RoboCup International Symposium*, 2019.

Appendix: Lucy (Toyota HSR) Software

Fig. 5: Lucy

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

Robot's Software Description

For our robot, we are using the following software:

- *Platform*: Robot Operating Systems (ROS)
- *Navigation*: Built-in ROS-based functionalities provided by Toyota
- *Arm control*: In-house imitation learning framework and MoveIt!
- *Task planning*: ROSPlan using the LAMA planner
- *Object recognition*: Faster R-CNN
- *Speech recognition*: PocketSphinx and Kaldi (offline)
- *Natural language processing*: Rasa NLU and SocRob's NLU

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

Fig. 6: Lucy in the HSR Gazebo simulation

Cloud Services

Lucy is currently not connected to any cloud services