Tech United Eindhoven @Home SPL 2017 Team Description Paper

J. Scholtes, M.F.B. van der Burgh, J.J.M. Lunenburg, R.P.W. Appeldoorn, R.W.J. Wijnands,
T.T.G. Clephas, M.J.J. Baeten, L.L.A.M. van Beek, R.A. Ottervanger and
M.J.G. van de Molengraft

Eindhoven University of Technology,
Den Dolech 2, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
http://www.techunited.nl, techunited@tue.nl

Abstract. This is an abstract

1 Introduction

Tech United Eindhoven is the RoboCup student team of Eindhoven University of Technology that (since 2005) successfully competes in the robot soccer Middle Size League (MSL) and later (2011) joined the ambitious @Home League. The Tech United @Home team currently holds the runner up title of RoboCup 2016 in Leipzig and is the reigning European Champion. (2016 RoboCup European Open, Eindhoven)

Our ambition is to extend our team, efforts and expertise with a new winner: the Toyota Human Support Robot (HSR). This standard platform will enable us to further increase our impact on the robotics open source community.

We are driven by our personal believe that service robots will have an immensely positive impact on the quality of live and independence of people; especially elderly and physically impaired people can benefit from this ground-breaking technology. It is not only valuable technology, but also has urgency as most of the western economies have an aging society. We are therefore intrinsically motivated to contribute to the open source community to move this technology to new boundaries that allow robots to safely, robustly and effectively function and work around people.

It is our firm believe that the impact, acceptance and use of new technology greatly depends on standardization of hardware and software systems. Metaphorically this is the foundation upon which great technology can be build and scaled. We see the availability of the HSR as a step forward in creating an eco-system that fosters new innovation and pushed the boundaries of what's possible.

This proposal will explain our development plans with the Toyota HSR system.

2 Description of the approach planned to be implemented on the robot

This section will explain the methods and software that will be used to solve the RoboCup@Home challenges with the Toyota HSR system.

2.1 Environment Descriptor (ED)

The TUE Environment Descriptor (ED) is a Robot Operating System (ROS) based 3D geometric, object-based world representation system for robots. In itself ED is database system that structures multi-modal sensor information and represents this in an object-based world representation that can be utilized for robot localization, navigation, manipulation and interaction functions. See Figure 1 for a schematic overview of ED. [1] Moreover, ED is used as extension of the ROS middle ware and is necessary to execute efficient collaboration between multiple robots. ED allows multiple robots to use and update the same world model. Static and dynamic objects in the environment space are shared which allows 'easy' cooperation when if for example comes to picking up objects. Robot-Robot task distribution can easily be distributed without the need of additional communication protocol that translates the perception of on robot to the other. ED is one re-usable environment description that can be used for a multitude of needed functionalities. Instead of having different environment representations for localization Adaptive Monte Carlo

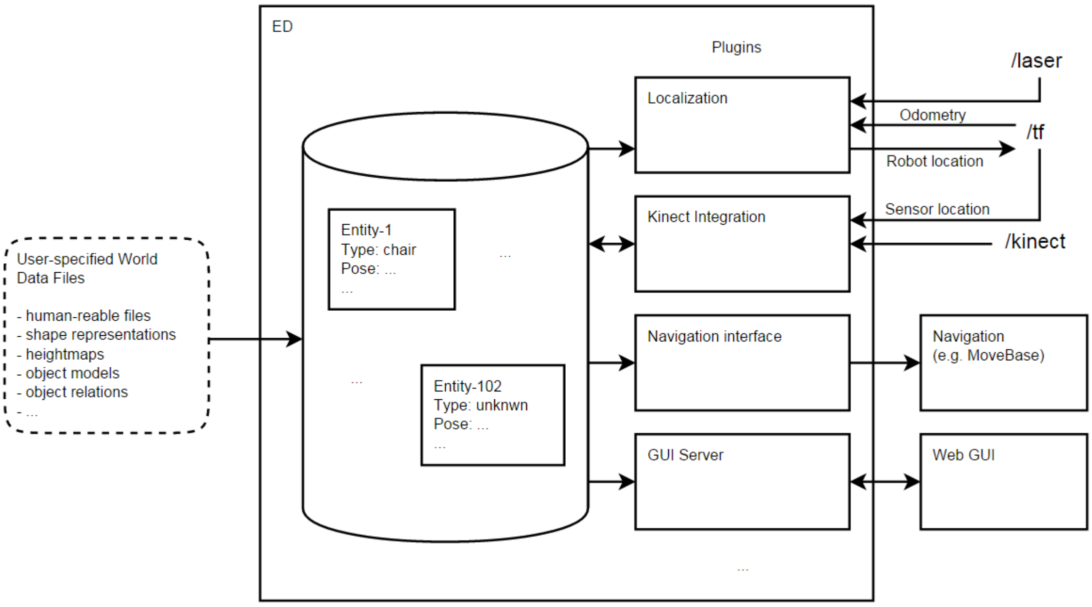


Fig. 1: schematic overview of TUE Environment Descriptor.

[1] ED is an evolution of World Information for Robotic Environments (WIRE) that was created in the FP7 RoboEarth Project. Secondly, the Environment Descriptor is utilized within the RoboCup @home competition (also read the TU/e 2015 technical description paper for RoboCup). More information, software, installation manual and tutorial can be found on `https://github.com/tue-robotics/ed`

Localization (AMCL), navigation (MoveBase [2]), manipulation (MoveIt![3]), interaction, etc. An improvement in this single, central world model will reflect in the performances of the separate robot capabilities. It omits updating and synchronization of multiple world models. At the moment different ED modules exist which enable robots to localize themselves, update positions of known objects based on recent sensor data, segment and store newly encountered objects and visualize all this through a web-based GUI.

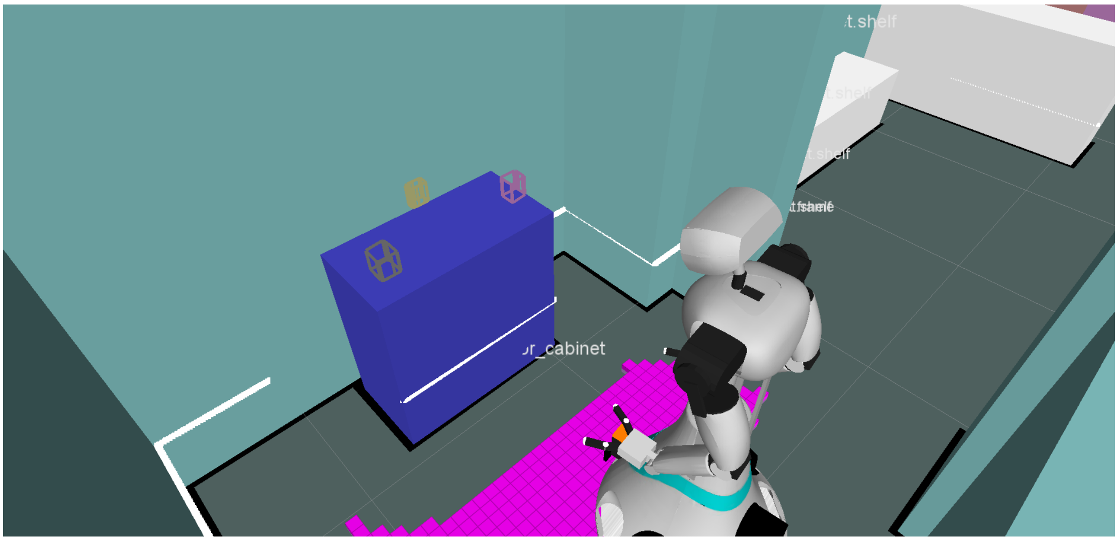


Fig. 2: A view of the world model created with ED. The figure, show the occupation grid as well as (unknown) objects recognized on top of the cabinet.

2.2 Localization

The ED-localization plugin[4] implements Monte Carlo localization based on the environment description in ED, laserscan readings (sensor_msgs/Laserscan) and a valid transformation between the /odom and the /base_link frame of the robot (tf). It does not differ much from the ROS AMCL package[5]; where the AMCL package uses a grid map as representation, the ED-localization plugin uses a 2D render from its world model. Slam_gmapping[6] - ROS simultaneous localization and mapping (SLAM) component - was configured for usage on PR2 and Toad robots. It requires only laser scanner (messages of type sensor_msgs/LaserScan) and odometry (TF transform between robot base and odometry reference frame). When there is a static environment as in the use case it can be used to obtain 2D map and then robot can localize against this map by the AMCL package. The AMCL also requires only laser scanner and odometry and provides reasonably robust localization. Full description of both components is given on their respective ROS wiki pages.

[2] http://wiki.ros.org/move_base
[3] http://moveit.ros.org/
[4] https://github.com/tue-robotics/edlocalization
[5] http://wiki.ros.org/amcl
[6] http://wiki.ros.org/gmapping

2.3 Navigation and Exploration

In order to navigate, a model of the environment is required. This model is stored in the Environment Descriptor (ED). From this model, a planning representation is derived that enables using the model of the environment for navigation purposes. With use of the ed_navigation plugin [7], an occupancy grid is derived from the world model and published as a nav_msgs/OccupancyGrid, as illustrated in the picture above. This representation can be used by a motion planner to perform searches in the configuration space of the robot.

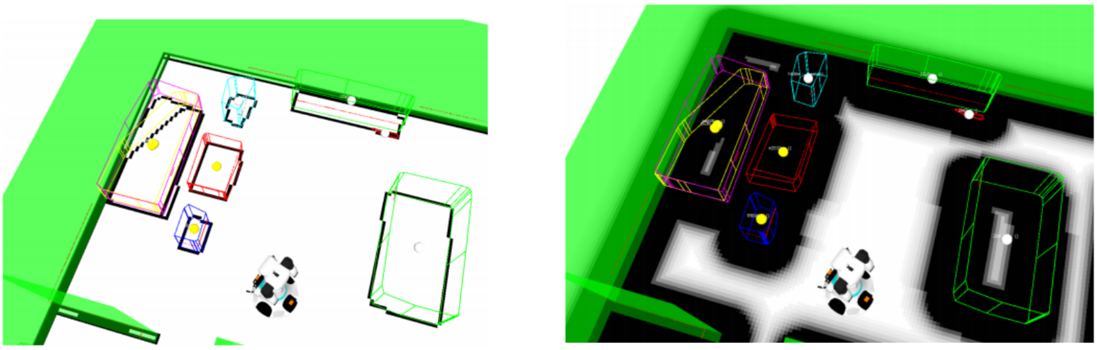
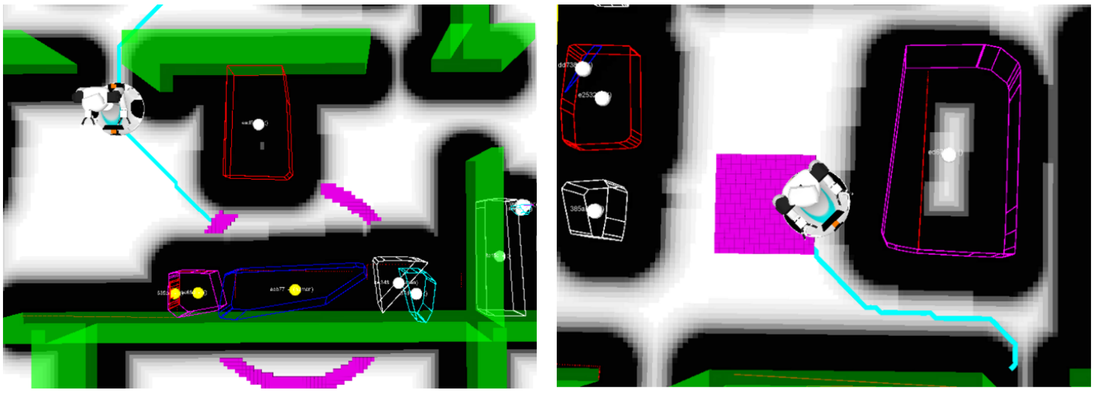


Fig. 3: Grid representation of the environment.

An example of such a motion planner is move_base[8] or the cb_base_navigation[9] ROS package. The latter software package, in contradiction to the move_base software package is able to deal with end goal constraints. With use of a ROS service, provided by the ed_navigation plugin, an end goal constraint can be constructed w.r.t. a specific world model entity described by ED. This enables the robot to not only navigate to poses but also to areas or entities in the scene, as illustrated by the following picture.

[7] https://github.com/tue-robotics/ed_navigation
[8] http://wiki.ros.org/move_base
[9] https://github.com/tue-robotics/cb_base_navigation



(a) Circle constraint, $x^2+y^2 > a^2$ and $x^2+y^2 < b^2$, frame *kitchenblock*

(b) Rectangular constraint, $x > a$ and $x < b$ and $y > c$ and $y < d$, frame *livingroom*

Fig. 4: Navigation position constraints w.r.t. other entities in the environment

As for the local and global motion planners, somewhat modified versions of the ROS planners available within move_base are used. The configurations for move_base node has now been created for the AMIGO and SERGIO robots, the HSR node will be added. The node integrates local and global path planning and local control. Its actionlib interface has become in fact standard for sending 2D navigation goals to robots in ROS.

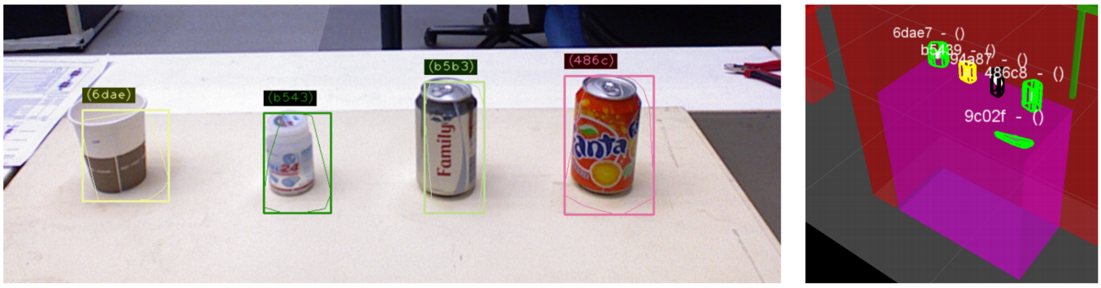
2.4 Object detection and recognition

The Environment Descriptor (ED) enables integrating sensor data with use of the plugins present in the ed_sensor_integration1 package. Two different plugins do exist: 1. laser_plugin: Enables tracking of 2D laser clusters. This plugin can be used to track dynamic obstacles such as humans. 2. kinect_plugin: Enables world model updates with use of Kinect data. This plugin exposes several ROS services that realize different functionalities: a. Segment: Service that segment sensor data that is not associated with other world model entities. Segmentation areas can be specified per entity in the scene. This allows to segment object 'on-top-of' or 'in' a cabinet. b. FitModel: Service that fits the specified model in the sensor data of the Kinect. This allows updating semi-static obstacles such as tables and chairs.

The ed_sensor_integration plugins enable updating and creating entities. However, new entities are classified as unknown entities. In order the classify or train unknown entities, the ed_perception[10] plugin exposes ROS Services. The following ROS Services are exposed:

1. **AddTrainingInstance.srv** This service adds training data to an ed_object_model
2. **Classify.srv** This service tries to classify an unknown entity in the scene. The following classifiers are used
   (a) Size matcher
   (b) SIFT matcher
   (c) Color matcher
   (d) QR detector

[10] https://github.com/tue-robotics/ed_perception

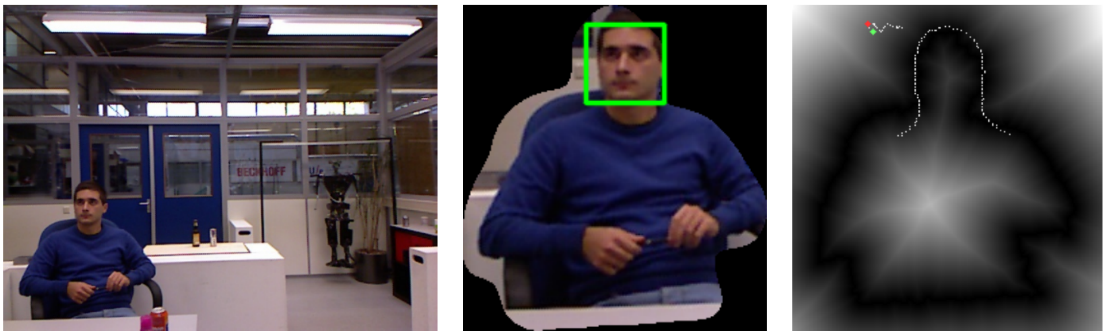


(a) Cup, Pills, Diet cok, Fanta - Kinect view

(b) World representation

   (e) Face detector (face detection / template matching)
3. **LearnPerson.srv** Service that learns the face of an entity in the scene

For object recognition, the classify service is used. The classify services takes as an argument the id of the entity and returns the classification. This can either be a trained object or person or just the label “human”. In order to create a new training set for specific objects, the ed_perception package contains several tools for segmenting objects and annotation.



(a) Full RGB Image

(b) Face detection

(c) Template matching

2.5 Object grasping, moving and placing

As for manipulating objects, the architecture is only focused on grasping. The input is the specific target entity in the world model (ED). The output is the grasp motion, i.e. joint positions for all joints in the kinematic chain over time. The following figure shows the grasping pipeline: The python executive queries the current pose of the entity from ED. The resulting grasp pose goes to the grasp precompute component which makes sure that we approach the object in a proper way. Finally, MoveIt will solve the IK solution and produces joint trajectories over time with use of the current configuration, the URDF model and the final configuration. Note that MoveIt currently does not take any information from ED into account. Finally, the trajectories are sent to the reference interpolator which sends the trajectories either to the controllers or the simulated robot.

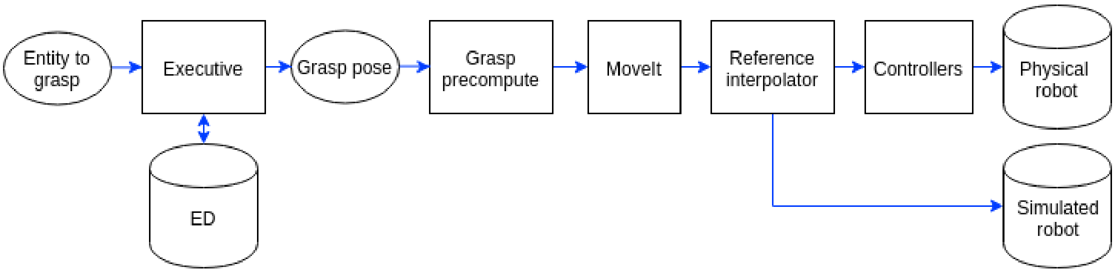


Fig. 5: Grasping pipeline.

2.6 Reasoning

The reasoning layer of the AMIGO ROS based software consists of a set of finite state machines (robot_smach_states1) that build upon the robot's skill layer (robot_skills2). These state machines are useful if you want the robot to execute some complex plan, where all possible states and state transitions can be described explicitly. The implementation is done with use of the open-source SMACH3 package. SMACH is a task-level architecture for rapidly creating complex robot behaviours. At its core, SMACH is a ROS-independent Python library to build hierarchical state machines. SMACH is a new library that takes advantage of very old concepts in order to quickly create robust robot behaviour with maintainable and modular code.

3 List of externally available components that are planned to be implemented

An overview of the software used by the Tech United Eindhoven @Home robots can be found in Table 1. All our software is developed open-source at GitHub[11]

Table 1: Software overview of the robots.

Operating system	Ubuntu 14.04 LTS Server
Middleware	ROS Indigo [1]
Low-level software	Orocos Real-Time Toolkit [2]
World model	Environment Descriptor (ED), custom `https://github.com/tue-robotics/ed`
Localization	Monte Carlo [3] using Environment Descriptor (ED), custom `https://github.com/tue-robotics/ed_localization`
SLAM	Gmapping: `http://wiki.ros.org/gmapping`
Navigation	Global: custom A* planner Local: modified ROS DWA [4]
Arm navigation	Custom implementation using MoveIt and Orocos KDL
Object recognition	Combination of size matching (custom), color matching (custom) and Objects-of-Daily-Use Finder `http://wiki.ros.org/objects_of_daily_use_finder`
People detection	Custom implementation using contour matching
Face detection	OpenCV Face detection `http://docs.opencv.org/trunk/doc/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html`
Face recognition	OpenCV Face recognition `http://docs.opencv.org/trunk/modules/contrib/doc/facerec/facerec_tutorial.html`
Speech recognition	Dragonfly + Windows Speech Recognition `http://code.google.com/p/dragonfly/`
Speech synthesis	Philips Text-to-Speech
Task executors	SMACH `http://wiki.ros.org/smach`

3.1 Innovative technology and scientific contribution

Whole body motion control Object recognition GUI Verslag Rokus

Papers, theses

3.2 Focus of research / research interests

Halve A4

3.3 Re-usability of the system for other research groups

All of our software is open-source designed. Our repositories are available on GitHub, https://github.com/tue-robotics. All packages are equipped with documentation and tutorials. In example, our ROS wrapper for openface is forked by the Toyota Research Institute. Nog iets verder uitbreiden

[11] `https://github.com/tue-robotics`

3.4 Applicability of the approach in the real world

ED enables the robots to operate in dynamic environments used by humans. The constraint-based navigation is robust to obstacles. . . . nog iets uitbreiden

References

1. 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.
2. H. Bruyninckx. Open robot control software: the orocos project. In *Proceedings of the 2001 IEEE International Conference on Robotics & Automation*, 2001.
3. D. Fox. Adapting the sample size in particle filters through kld-sampling. *The International Journal of Robotics Research*, 22(12):985–1003, 2003.
4. D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Magazine on Robotics & Automation*, 4(1):23–33, 1997.