Tech United Eindhoven @Home 2022 Team Description Paper

M.F.B. van der Burgh , J.J.M. Lunenburg, R.P.W. Appeldoorn, L.L.A.M. van Beek, J. Geijsberts, L.G.L. Janssen, P. van Dooren, H.W.A.M. van Rooy, A. Aggarwal, S. Narla 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,
https://github.com/tue-robotics

Abstract. This paper provides an overview of the main developments of the Tech United Eindhoven RoboCup @Home team. Tech United uses an advanced world modeling system called the Environment Descriptor. It allows straightforward implementation of localization, navigation, exploration, object detection & recognition, object manipulation and robot-robot cooperation skills based on the most recent state of the world. Other important features include object and people detection via deep learning methods, a GUI, speech recognition, natural language interpretation and a chat interface combined with a conversation engine. Recent developments include grasp detection, door opening and a topological action planner.

1 Introduction

Tech United Eindhoven[1] (established 2005) is the RoboCup student team of Eindhoven University of Technology[2] (TU/e), which joined the ambitious @Home League in 2011. The RoboCup @Home competition aims to develop service robots that can perform everyday tasks in dynamic and cluttered 'home' environments. Multiple world vice-champion titles have been obtained in the Open Platform League (OPL) of the RoboCup @Home competition during previous years, and last year, whilst competing in the Domestic Standard Platform League (DSPL), the world championship title was claimed.

Tech United Eindhoven consists of (former) PhD and MSc. students and staff members from different departments within the TU/e. The software base is developed to be robot independent, which means that the years of development on AMIGO and SERGIO are currently being used by HERO. Thus, a large part of the developments discussed in this paper have been optimized for years, whilst the DSPL competition has only existed since 2017[3]. All the software discussed

[1] http://www.techunited.nl
[2] http://www.tue.nl
[3] https://athome.robocup.org/robocuphome-spl

in this paper is available open-source at GitHub[4], as well as various tutorials to assist with implementation.

2 Environment Descriptor

The TU/e Environment Descriptor (ED) is a Robot Operating System (ROS) based 3D geometric, object-based world representation system for robots. ED is a database system that structures multi-modal sensor information and represents this such that it can be utilized for robot localisation, navigation, manipulation and interaction. Figure 1 shows a schematic overview of ED.
ED has been used on our robots in the OPL since 2012 and was also used this year in the DSPL. Previous developments have focused on making ED platform independent, as a result ED has been used on the PR2, Turtlebot, Dr. Robot systems (X80), as well as on multiple other @Home robots.

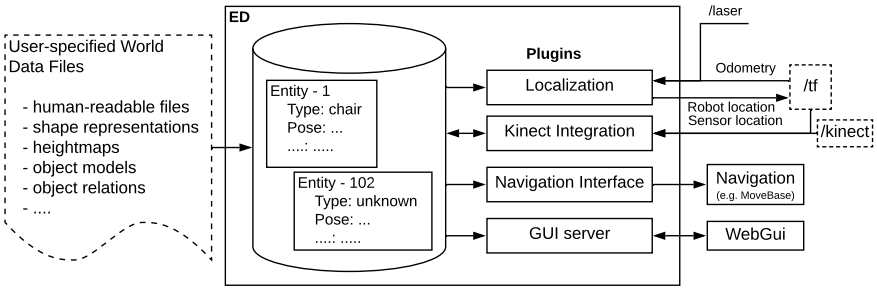

Fig. 1. Schematic overview of TU/e Environment Descriptor. Double sided arrows indicate that the information is shared both ways, one sided arrows indicate that the information is only shared in one direction.

ED is a single re-usable environment description that can be used for a multitude of desired functionalities such as object detection, navigation and human machine interaction. Improvements in ED reflect in the performances of the separate robot skills, as these skills are closely integrated in ED. This single world model allows for all data to be current and accurate without requiring updating and synchronization of multiple world models. Currently, different ED plug-ins exist that 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 in RViz and through a web-based GUI, as illustrated in Figure 6. ED allows for all the different subsystems that are required to perform challenges to work together robustly. These various subsystems are shown in Figure 2, and are individually elaborated upon in this paper.

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

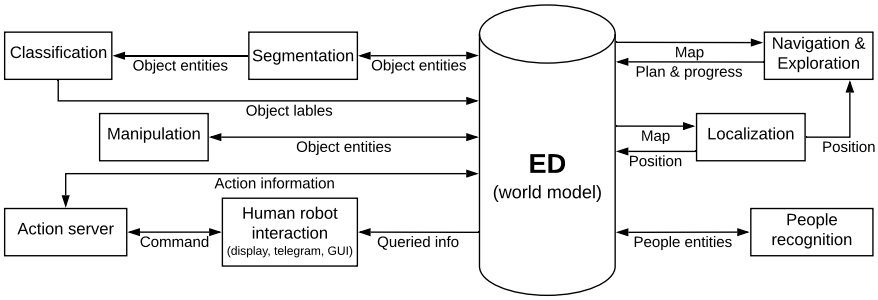

Fig. 2. A view of the data interaction with robot skills that ED is responsible for.

2.1 Localization, Navigation and Exploration

The *ed_localization*[5] plugin implements AMCL based on a 2D render of the central world model. With use of the *ed_navigation* plugin[6], an occupancy grid is derived from the world model and published. With the use of the *cb_base_navigation* package[7] the robots are able to deal with end goal constraints. The *ed_navigation* plugin allows to construct such a constraint w.r.t. a world model entity in ED. This enables the robot to navigate not only to areas or entities in the scene, but to waypoints as well. Figure 3 shows the navigation to an area. Modified versions of the local and global ROS planners available within *move_base* are used.

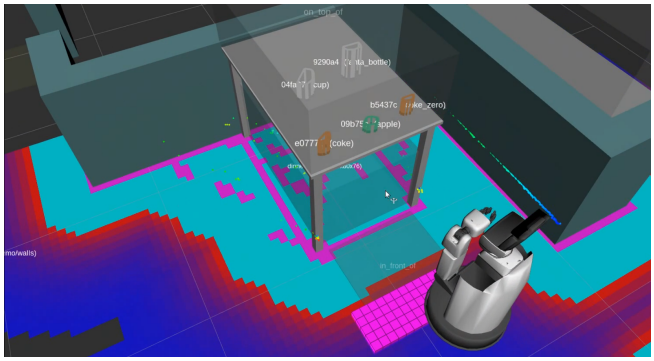

Fig. 3. A view of the world model created with ED. The figure shows the occupancy grid as well as classified objects recognized on top of the cabinet.

The *topological_action_planner*[8] facilitates navigation when manipulation is required to reach a goal, e.g., when a door is closed.

[5] `https://github.com/tue-robotics/ed_localization`
[6] `https://github.com/tue-robotics/ed_navigation`
[7] `https://github.com/tue-robotics/cb_base_navigation`
[8] `https://github.com/tue-robotics/topological_action_planner`

2.2 Detection & Segmentation

ED enables integrating sensors through the use of the plugins present in the *ed_sensor_integration* package. Two different plugins exist:

1. *kinect_plugin*: Enables world model updates with use of data from a RGBD camera. This plugin exposes several ROS services that realize different functionalities:
   (a) *Segment*: A service that segments 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. All points outside the segmented area are ignore for segmentation.
   (b) *FitModel*: A service that fits the specified model in the sensor data of a RGBD camera. 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. Classification is done in *ed_perception* plugin[9] package.

2.3 Object grasping, moving and placing

The system architecture developed for object manipulation is focused on grasping. In the implementation, its input is a specific target entity in ED, selected by a Python executive and the output is the grasp motion joint trajectory. Figure 4 shows the grasping pipeline.

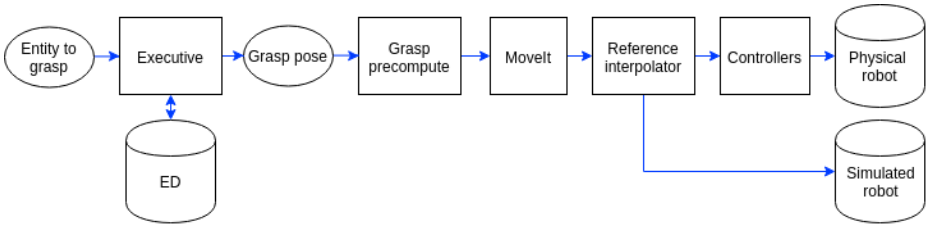

Fig. 4. Custom grasping pipeline base on ED, MoveIt and a separate grasp point determination and approach vector node.

MoveIt! is used to produce joint trajectories over time, given the current configuration, robot model, ED world model (for collision avoidance) and the final configuration.
The grasp pose determination uses the information about the position and shape of the object in ED to determine the best grasping pose. The grasping pose is a vector relative to the robot. Placing an object is approached in a similar manner to grasping, except that when placing an object, ED is queried to find a suitable placement pose.

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

3 Image Recognition

The *image_recognition* packages apply state of the art image classification techniques based on Convolution Neural Networks (CNN).

1. **Object recognition:** Tensorflow™ with retrained top-layer of a Inception V3 neural network.
2. **Face recognition:** OpenFace[10], based on Torch.
3. **Pose detection:** OpenPose[11].

Our image recognition ROS packages are available on GitHub[12] and as Debian packages: *ros-kinetic-image-recognition*

4 People Recognition

As our robots need to operate and interact with people in a dynamic environment, our robots' need people detection skills. For this a generalized system capable of recognizing people in 3D is developed. An RGB-D camera is used to capture the scene information after which a recognition sequence is completed in four steps. First, people are detected in the scene using OpenPose and if their faces are recognized as one of the learned faces in the robots'database, they are labeled using their known name using OpenFace. The detections from OpenPose are associated with the recognitions from OpenFace by maximizing the IoUs of the face ROIs. Then, for each of the recognized people, additional properties such as age, gender and the shirt color are identified. Furthermore, the pose keypoints of these recognitions are coupled with the depth information of the scene to re-project the recognized people to 3D as skeletons. Finally, information about the posture of each 3D skeleton is calculated using geometrical heuristics. This allows for the addition of properties such as "pointing pose" and additional flags such as 'is_waving', 'is_sitting', etc.

4.1 Pointing detection

In the previous section, our approach to people recognition is explained. This recognition includes information about the posture of each 3D skeleton. Once the people information is inserted into the world model, additional properties can be added to the persons that take also other entities in the world model into account, e.g. 'is_pointing_at_entity'. This information is used by the toplevel state machines to implement challenges. However an additional check is inserted to ensure that the correct operator is found, this check is based on spatial queries. By using such a query it is possible to filter out people based on their location.

[10] `https://cmusatyalab.github.io/openface/`
[11] `https://github.com/CMU-Perceptual-Computing-Lab/openpose`
[12] `https://github.com/tue-robotics/image_recognition`

Finally, to determine at which entity the operator is pointing, ray-tracing is implemented. Figure 5 shows an example of the ray-tracing.

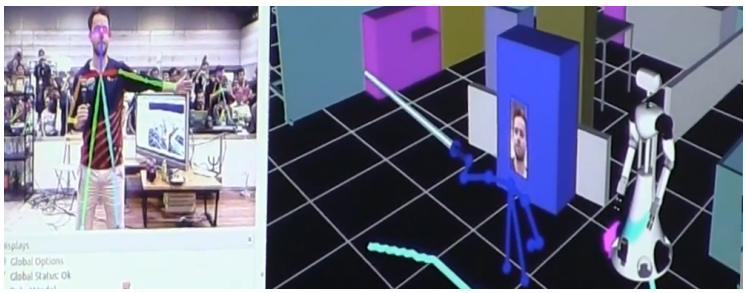

Fig. 5. Ray-tracing based on pose detection with AMIGO.

5 Human-Robot Interface

We provide multiple ways of interacting with the robot in an intuitive manner: WebGUI, subsection 5.1, and Telegram™ interface, subsection 5.2, which uses our *conversation_engine*, subsection 5.2.

5.1 Web GUI

In order to interact with the robot, apart from speech, we have designed a web-based Graphical User Interface (GUI). This interface uses HTML5[13] with the Robot API written in Javascript and we host it on the robot itself.

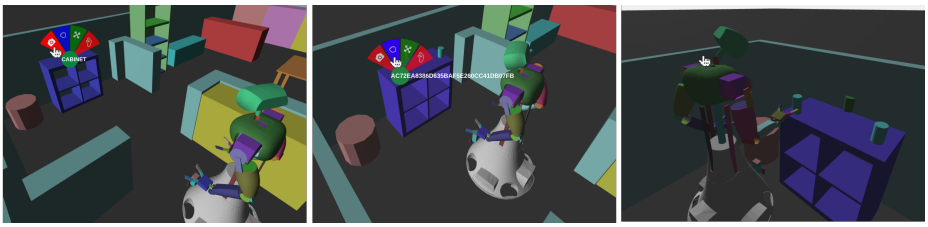

Fig. 6. Illustration of the 3D scene of the WebGUI with AMIGO. User can long-press objects to open a menu from which actions on the object can be triggered

Figure 6 represents an instance of the various interactions that are possible with the Robot API.

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

5.2 Telegram™

The Telegram interface[14] to our robots is a ROS wrapper around the *python-telegram-bot* library. The software exposes four topics, for images and text resp. from and to the robot. The interface allows only one master of the robot at a time. The interface itself does not contain any reasoning. This is all done by the *conversation_engine*, which is described in the following subsection.

Conversation Engine
The *conversation_ engine*[15] bridges the gap between text input and an action planner (called action_server). Text can be received from either Speech-to-Text or from a chat interface, like Telegram™. The text is parsed according to a (Feature) Context Free Grammar, resulting in an action description in the form of a nested mapping. In the action description, (sub)actions and their parameters are filled in. This may include references such as "it". Based on the action description, the action_server tries to devise a sequence of actions and parameterize those with concrete object IDs. To fill in missing information, the conversation_engine engages with the user. When the user supplies more information, the additional input is parsed in the context of what info is missing. Lastly, it keeps the user "informed" whilst actions are being performed by reporting on the current subtask.

Custom Keyboard, Telegram HMI
The user interface modality as explained above has been extended to reduce the room for operator error by only presenting the user with a limited number of buttons in the Telegram app. This has been realized through Telegrams *custom_ keyboards*[16] feature. This feature has been employed to compose commands word-for-word. After the user has already entered, via text or previous buttons, for example "Bring me the ..." the user is presented with only those words that might follow that text according to the grammar, eg. "apple", "orange" etc. This process iterates until a full command has been composed. This feature is called *hmi_ telegram*[17]

5.3 Head Display

For most people, especially people who do not deal with robots in their day-to-day life, interaction with robots is not as easy as one would like it to be. To remedy this, the head display of HERO is used. On this display that is integrated in the Toyota HSRs' 'head', a lot of useful information can be displayed. Through

[14] `https://github.com/tue-robotics/telegram_ros`
[15] `https://github.com/tue-robotics/conversation_engine`
[16] `https://github.com/tue-robotics/https://github.com/tue-robotics/telegram_ros/tree/rwc2019`
[17] `https://github.com/tue-robotics/hmi_telegram`

the *hero_ display*[18] a few different functionalities are integrated. As per default, our Tech United @Home logo with a dynamic background is shown on the screen, as depicted in Figure 7. When the robot is speaking the spoken text is displayed, when the robot is listening a spinner along with an image of a microphone is shown and it is possible to display images.

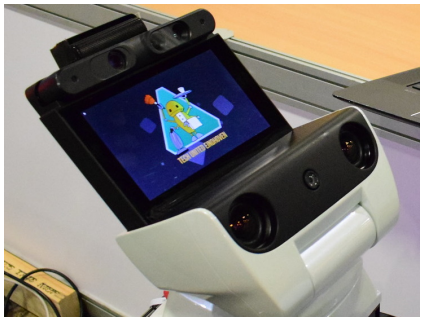

Fig. 7. The default status of HERO's head display.

6 Re-usability of the system for other research groups

Tech United takes great pride in creating and maintaining open-source software and hardware to accelerate innovation. Tech United initiated the Robotic Open Platform website[19], to share hardware designs. All our software is available on GitHub[20]. All packages include documentation and tutorials. Tech United and its scientific staff have the capacity to co-develop (15+ people), maintain and assist in resolving questions.

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

[18] `https://github.com/tue-robotics/hero-display`
[19] `http://www.roboticopenplatform.org`
[20] `https://github.com/tue-robotics`

7 HSR's Software and External Devices

We use a standard Toyota™ HSR robot. To differentiate our unit, we named it HERO. We wanted to link it's name to our AMIGO and SERGIO domestic service robots.
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[21].

Table 1. Software overview

Operating system	Ubuntu 16.04 LTS Server
Middleware	ROS Kinetic [1]
Simulation	Gazebo
World model	Environment Descriptor (ED), custom `https://github.com/tue-robotics/ed`
Localization	Monte Carlo [2] using Environment Descriptor (ED), custom `https://github.com/tue-robotics/ed_localization`
SLAM	Gmapping
Navigation	CB Base navigation `https://github.com/tue-robotics/cb_base_navigation` Global: custom A* planner Local: modified ROS DWA [3]
Arm navigation	MoveIt!
Object recognition	image_recognition_tensorflow `https://github.com/tue-robotics/image_recognition/tree/master/image_recognition_openface`
People detection	Custom implementation using contour matching `https://github.com/tue-robotics/ed_perception`
Face detection & recognition	image_recognition_openface `https://github.com/tue-robotics/image_recognition/tree/master/image_recognition_openface`
Speech recognition	Windows Speech Recognition
Speech synthesis	Toyota™ Text-to-Speech
Task executors	SMACH `https://github.com/tue-robotics/tue_robocup`

External Devices *HERO relies on the following external hardware:*

- Official Standard Laptop
- USB power speaker
- Gigabit Ethernet Switch
- Wi-Fi adapter

Cloud Services *HERO connects the following cloud services:* None

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