Tech United Eindhoven @Home 2016 Team Description Paper

J.J.M. Lunenburg, S. van den Dries, R.P.W. Appeldoorn, R.W.J. Wijnands 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 paper provides an overview of the main developments of the Tech United Eindhoven RoboCup@Home team. Our main research efforts this year have focused on i) fitting furniture objects to update our object-oriented world model, thereby improving localization, navigation and object segmentation ii) developing a WebGUI to provide the user with a platform-independent way to interact with the robot and iii) natural language interpretation to ease the specification of written commands for the robot and to make speech recognition more robust.

1 Introduction

Tech United Eindhoven is the RoboCup team of the Eindhoven University of Technology, competing in the Middle Size League and the @Home League. Tech United has been competing in the @Home league since 2011, winning the 2015 RoboCup German Open and finishing fourth at RoboCup 2015 in Hefei. Tech United Eindhoven consists of (former) PhD and MSc students and staff members from different departments within the Eindhoven University of Technology.

This Team Description Paper is part of the qualification package for RoboCup 2016 in Leipzig, Germany and describes the current status of the @Home activities of Tech United Eindhoven. Our main research efforts this year have focused on fitting furniture objects to update our object-oriented world model (Section 2), developing a WebGUI to provide the user a platform-independent way to interact with the robot (Section 3) and natural language interpretation (Section 4).

2 World Model Furniture Fitting

To successfully perform a RoboCup@Home challenge, a robot needs a basic understanding of the environment: it needs to know how to get from A to B without colliding, it must be able to detect humans and objects with which it has to interact and it must localize itself with respect to the objects it has to manipulate. Therefore, Environment Descriptor (ED), a 3D, object-based, volumetric world model was developed. The datastructures and algorithms used in this world model are described in the 2015 TDP. More information and tutorials can be found on our GitHub page[1].

ED depends on having an accurate description of the world, including the locations of walls and furniture. However, tables, chairs and even cabinets may move in a household environment. If the world model is not updated accordingly, two important problems arise:

[1] https://github.com/tue-robotics/ed

– The robot will not position itself correctly with respect to the furniture (cabinet, table, etc) when it needs to interact with or inspect these entities
– Since ED uses background subtraction for object segmentation, segmentation may not work correctly

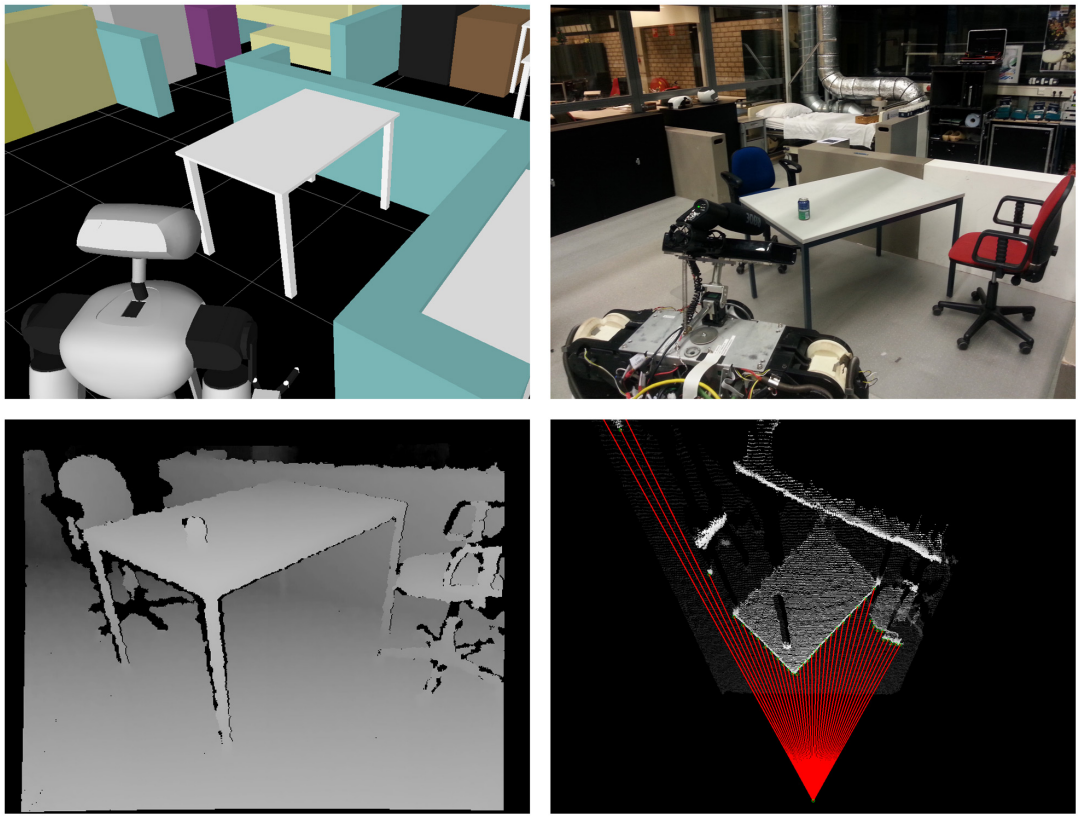

Fig. 1: World model furniture fitting algorithm. Top-left: world model visualization as the robot thinks the world is. Top-right: actual situation: the table is rotated and moved. Bottom-left: depth image obtained from the depth sensor (brighter means closer). Bottom-right: top-down projected point cloud (brighter means closer); dividing the points into beams, 2D range data can be extracted and used for efficient template fitting.

To deal with this and to get rid of pre-defined location definitions in general, an ED-plugin was developed which can estimate the positions of furniture in the room. This pose estimation algorithm works as follows (also see Figure 1):

1. The camera pose with respect to the floor is obtained using the robot location and forward kinematics
2. The point cloud obtained from the depth camera is transformed to the robot's base frame using the camera pose
3. All points belonging to the floor are filtered using a simple height check per point. All other points are projected onto the floor (resulting in a top-down view).

4. Then, all projected points are divided into beams that originate at the camera origin. The closest point per beam is stored. This results in a 1-dimensional array of ranges, very similar to the sensor data obtained from a laser range finder. However, the ranges obtained from this algorithm do not simply represent a planar cross-section of the world, but a projection of all points except the ground. This allows the data to also correctly represent a table (while a laser range finder would probably scan above or below the table top)
5. A top-down 2D contour model is created from the 3D model that needs to be fitted into the sensor data
6. A simple template-matching algorithm is used to fit the 2D contour model into the 1-dimension range data, resulting in the 2D-pose (x, y, yaw) of the model

The conversion from a 3D point cloud to a 1-dimension range array enables the use of a very efficient fitting algorithm, which allows for constant tracking of entities at high update rates. Do note, however, that the following assumptions are made:

- The entity that needs to be fitted is standing on the ground, and is merely translated along the floor plane, and rotated along the vertical axis (only yaw, no roll or pitch)
- The entity is not occluded for large parts by other entities
- A 2D top-down contour model is available, or can be constructed from a 3D model

Fortunately, these assumptions hold in many situations when trying to estimate the poses of furniture. Several experiments in real situations have shown that the fitting strategy is indeed efficient and effective in many situations.

3 WebGUI

Overview

In order to interact with the robot aside of speech, a web-based Graphical User Interface (GUI) has been designed. The interface has been made with HTML5[2] and is hosted on the robot itself. This allows multiple users on different platforms (*e.g.* Android, iOS) to access functionality of the robot.

The interface is implemented in JavaScript with AngularJS and it offers a graphical interface to the Robot API[3] which exposes all the functionality of the robot. Figure 2 gives an overview of the connections between these components.

[2] `https://github.com/tue-robotics/tue_mobile_ui`
[3] `https://github.com/tue-robotics/robot-api`

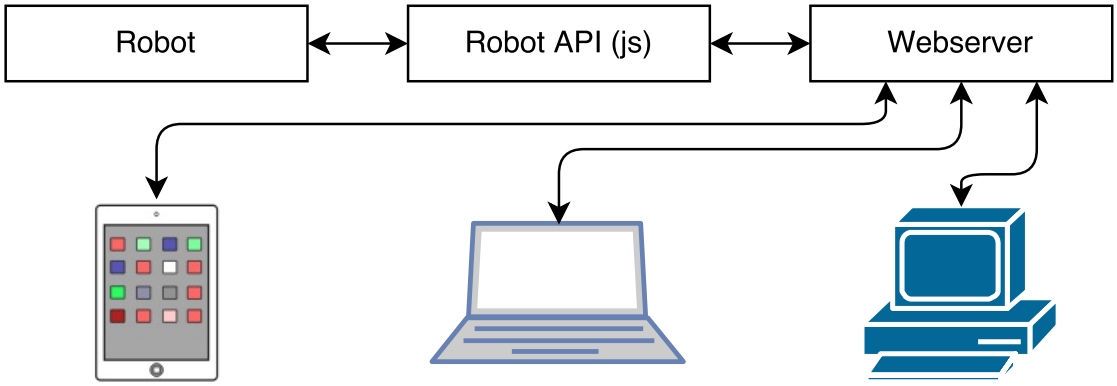

Fig. 2: Overview of the WebGUI architecture. The robot's functionalities are exposed with the Robot API that is implemented in JavaScript. The webserver that is hosting the GUI connects this Robot API to a graphical user interface that is offered to multiple clients on different platforms.

Figure 3 gives an example of various user interactions that are possible with the GUI and the different commands that can be given to the robot while interacting with the virtual scene.

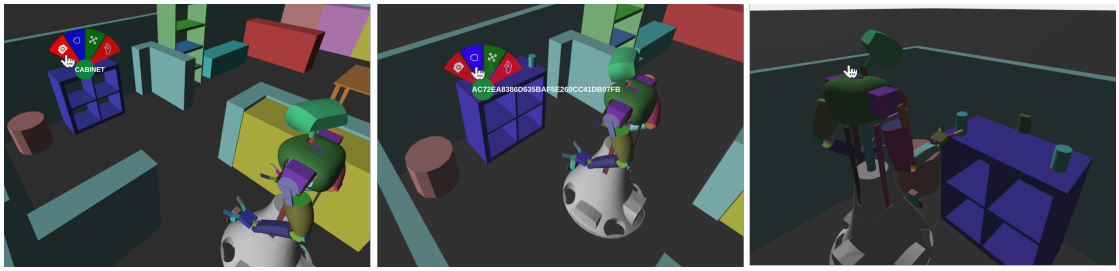

Fig. 3: Illustration of the 3D scene of the WebGUI. Users can interact with use of the menu that appears when long pressing an object in the scene. On the left figure, the user commands the robot to inspect the selected object, which is the 'cabinet'. When the robot has inspected the 'cabinet', it has found entities on top of it. In the middle figure a grasp command is given to the robot to pick up an object from the cabinet. The last figure show the robot executing that action.

These actions are passed via the Robot API to the action server that schedules the actions the robot has to perform. The action server enables various clients to give task commands to the robot without interfering with each other. Other clients that make use of this action server are Speech and the Natural Language Console, explained in Section 4.

4 Natural Language Interpretation

Being able to communicate what you want the robot to do is as vital as having it carry out those tasks successfully, and the more natural this communication is, the easier it is to express the actions. For example, typing:

```
robot.grasp("drink-123")
```

is more cumbersome than saying or typing:

```
robot grab the drink
```

In the first command, you have to know the exact names of the actions the robot can perform. Furthermore, the exact world model identifier of the drink has to be known. If natural language is used, intuition can be used to express the command, without needing further knowledge of the robots internals.

In the RoboCup@Home setting, a natural language interpreter is useful for two reasons:

- It is vital for using speech recognition (*e.g.*, for the GPSR challenge)
- Testing basic actions becomes much easier and less cumbersome. The following command allows us to test object segmentation, classification and manipulation: "inspect the cabinet and grab the drink"

A natural language interpreter needs to do two things: parse the structure of the sentence (checking syntax), and converting it to a concise representation of its meaning (semantics). This is done using a feature context free grammar (FCFG). Such a grammar represents the possible structure of the command and at the same time explains how the components of this structure add up into a meaning. An example of such an FCFG is the following:

```
COMMAND[{"action": A, "entity": X}] -> V[A] NP[X]

V["inspect"] -> inspect | look at
V["grab"] -> pick up | grab
V["navigate-to"] -> go to | navigate to

NP[X] -> DET N[X]

NP[{"id", "cabinet-123"}] -> cabinet-123
N[{"type": "cabinet"}] -> cabinet
N[{"type": "drink"}] -> drink

DET -> the | a
```

The capitalized terms represent the structure types encountered within a sentence. For example `'NP'` stands for 'Noun Phrase', `'DET'` for 'Determinant', `'V'` for verb. The rule

```
A -> B C
```

means that `A` can be composed of `B` and `C`. A vertical bar '|' means one of the terms can be used. For example, `DET` can be either 'the' or 'a'. The semantics of the grammar are captured within the square brackets. If a rule applies, the variables on the left hand side are replaced with the content of the variables on the right hand side. For example the sentence:

```
pick up a drink
```

will result in the JSON string:

```
{"action" : grab, "entity" : {"type": "drink"}}
```

which expresses that an entity of type drink must be picked up. This command can then be send to the action server, which is responsible for executing actions based on structured JSON commands. Notice that the grammar allows for specifying expressive, generic syntax and semantics. It is for example very easy to provide synonyms for verbs.

Aside from it being easy to use, an explicit data model (grammar) for the interpretation of sentences gives us the added advantage that it can also be used for speech recognition. Limiting the amount of options speech recognition can process greatly benefits the recognition quality. By 'telling' speech recognition what it should be able to hear, hearing bad sentences or meaningless commands can be avoided. For example, the grammar above can easily be updated such that "grab the cabinet" cannot be parsed, but "inspect the cabinet" is still possible. By providing speech recognition this updated grammar "grab the cabinet" will never accidentally be heard.

Since new objects may be discovered over the course of a challenge, the grammar is dynamically updated at run-time based on input from the world model. This means that if a table is added to the world model, the id and type of this object are also added as rules to the grammar. For example, in such a case the following rules could be added:

```
NP[{"id", "table-7"}] -> table-7
N[{"type": "table"}] -> table
```

Allowing interpretation of commands such as:

```
inspect table-7
go to the table
```

A natural language console was created which allows typing natural commands to the robot. Using the grammar, tab-completion could easily be integrated into this console.

5 Conclusions

In this paper, this year's main developments of Tech United Eindhoven have been discussed:

- Localization, navigation and object segmentation all benefit from the updating the pose of furniture objects by means of a novel fitting algorithm.
- Interaction with the robots is possible from a large variety of platforms due to the WebGUI.
- A Natural Language Interpreter allows an easier specification of commands to the robot and a robust speech recognition by excluding meaningless commands from the options that the robot is able to understand.

With these improvements, we hope to improve on last year's performance. We are looking forward to RoboCup 2016 in Leipzig!

Robot Hardware Descriptions

AMIGO (Autonomous Mate for Intelligent Operations, see Fig. 4) has competed in RoboCup@Home since 2011. Its design is based on a Middle Size League soccer robot, equipped with two Philips Experimental Robotic Arms mounted on an extensible upper body. Based on our experiences with AMIGO, SERGIO (Second Edition Robot for Generic Indoor Operations, see Fig. 5) has been developed. The main differences with AMIGO are the use of Mecanum wheels which are compliantly suspended, the torso with two degrees of freedom and the modular setup. The core specifications of AMIGO and SERGIO can be found in Table 1. More details about the robots can be found on the Robotic Open Platform[4], where all CAD drawings, electrical schemes and CAD files are published.

[4] http://www.roboticopenplatform.org/

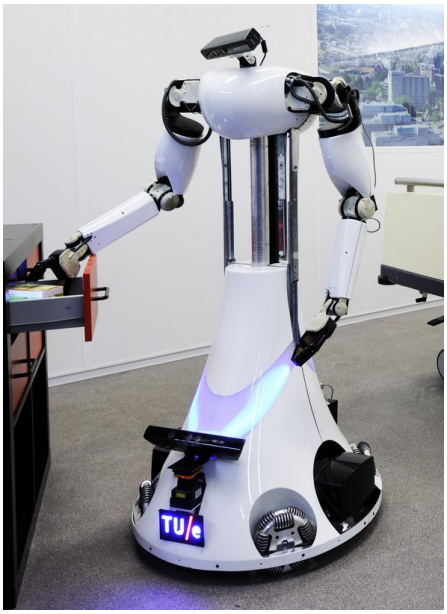

Fig. 4: The AMIGO robot.

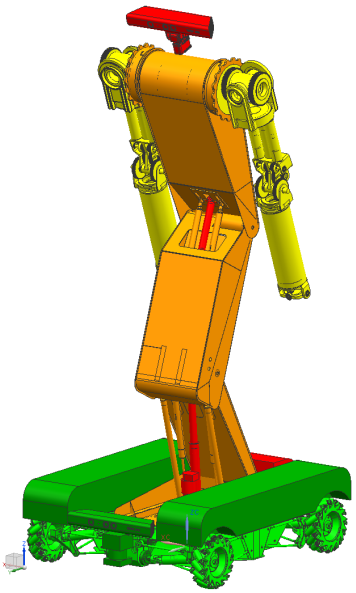

Fig. 5: CAD drawing of SERGIO.

Table 1: Core specifications of AMIGO and SERGIO

	AMIGO	SERGIO
Name	Autonomous Mate for IntelliGent Operations	Second Edition Robot for Generic Indoor Operations
Base	Fully holonomic omni-wheel platform based on a soccer robot	Fully holonomic Mecanum wheel platform with independent wheel suspension system
Torso	1 vertical DoF using a ball screw	1 nearly vertical DoF using a coupled ankle and knee joint, 1 rotational hip joint
Manipulators	2 7-DoF Philips Experimental Robotic Arms	2 7-DoF custom arms
Neck	Pan-tilt unit using two Dynamixel RX-28 servo actuators	Pan-tilt unit using two Dynamixel RX-64 servo actuators
Head	Kinect for XBox 360	Kinect for XBox 360
External devices	Wireless emergency button	Wireless emergency button
Dimensions	Diameter: 0.75 m, height: ±1.5 m	Base: 0.7 m × 0.6 m, height: ±1.65 m
Weight	±70 kg	±70 kg
Additional sensors	Hokuyo UTM-30LX laser range finder on base and torso	Hokuyo UTM-30LX laser range finder on base and torso (tilting)
Microphone	RØDE Videomic	RØDE Videomic Pro
Batteries	4× Makita 24 V, 3.3 Ah	4× Makita 24 V, 3.3 Ah
Computers	4× AOpen Mini PC with Core-i7 processor and 8 Gb RAM	3× Gigabyte mini ITX board with Core-i7 processor and 16 Gb RAM

Robot Software Description

An overview of the software used by the Tech United Eindhoven @Home robots can be found in Table 2. All our software is developed open-source at GitHub[5]

Table 2: Software overview of the robots.

Operating system	Ubuntu 14.04 LTS Server
Middleware	ROS Indigo
Low-level software	Orocos Real-Time Toolkit
World model	Environment Descriptor (ED), custom `https://github.com/tue-robotics/ed`
Localization	Monte Carlo using 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
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`

References

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