Exception Handling for Experience-based Mobile Cognitive Systems in Restaurant Environments Exemplified by Guest Detection

Liwei Zhang, Sebastian Rockel, Jianwei Zhang
TAMS, Department of Informatics
University of Hamburg, Germany
{*lzhang, rockel, zhang*}*@informatik.uni-hamburg.de*

***Abstract*— Recording and exploiting past experiences to handle an unforeseen situation is an important asset of human beings. However, in current robot architectures, experience-based learning has mainly been realized at sub-symbolic levels. In this paper, we present how to handle exception for an experience-based artificial cognitive system which is able to fulfill unforeseen situations in restaurant environments. Guest detection is employed to demonstrate the exception handling functionality. Experiments were executed to validate and evaluate the effectiveness of this artificial cognitive system.**

***Index Terms*— Experience Learning, Exception Handling, Cognitive System, Guest Detection.**

I. Introduction

It is well-known that recording and exploiting past experiences is an important asset of human beings. However, in current robot architectures, experience-based learning has mainly been realized at sub-symbolic levels. And on the other hand, if humans face an unforeseen situation, e.g. a blocked access, they try to recall experiences about such a situation and apply relevant memory records to handle the current situation. For example, in a restaurant environment, if the guest sits on a different side of the table other than specified in the planning domain, the robot waiter still brings the mug to the same place as before (which is now in front of an empty seat). But this situation will not happen when the waiter is a human being. The human waiter will try to find a solution to deal with such an exception situation. This solution must then be adapted to the current situation (which may not be the same in all respects as the recalled situation).

We are inspired by this situation and try to construct an experience-based artificial cognitive system which is able to fulfil unforeseen situations. The artificial cognitive system has been integrated in the EU project RACE (Robustness by Autonomous Competence Enhancement). The overall objective of the project is to enable a robot to obtain increased robustness by exploiting experiences. We will demonstrate robot behavior in an experimental restaurant domain. The robot will carry out tasks of a waiter, bringing food and coffee to any of several tables, placing dishes properly in front of guests, cleaning tables, etc.

To achieve this goal, we will integrate state-of-the-art approaches in an attempt to develop several learning and rule extraction methods developed in previous work and in the AI research community. The work in RACE is primarily directed at developing scientifically founded engineering solutions, but results may have a considerable impact on future applications, e.g. in health care, service robotics and industrial applications.

Work in RACE was integrated from several research areas: (1) multi-level knowledge representation tools and a formalisms framework which are based on an ontology; (2) a software reasoning framework to deal with hybrid and diverse knowledge; (3) a semantic interpretations framework for recording complex robot and environment activities; (4) learning approaches for gathering and exploiting all levels of recorded experiences. All this is integrated on a PR2 [1] platform, which is one of the most advanced general-purpose robot systems available today, and is assumed to possess the capabilities that more commonly available robots will possess in the near future.

Several scenarios in the restaurant domain have been set to validate and evaluate the experience-based artificial cognitive system. The idea is to let the robot discover a generalization of the experiences in experience-gathering phase, which are followed by an experiences-exploitation phase. In the experiences-exploitation phase, different exceptions will be designed to test the capability of exception handling. In the experiments, the guest sits on different side of the table other than specified in the planning domain, guest detection will be employed to handle the exception of a "ServeACoffee" task. In the experiences-gathering phase, the robot fails to place the mug in the targeted area on the table, i.e. in front of the guest. But the robot will complete the "ServeACoffee" task when the robot has the experience of "mug must be placed in front of the guest".

The rest of this work will be organized as follows: section 2 presents related work. In Section 3, we present the architecture of the mobile cognitive system which is being developed in ongoing research. Several scenarios are conducted in section 4 to evaluate and compare the efficiency

of the proposed system and, finally, in section 5, conclusions and future work are presented.

II. Related work

A. Exception Handling and Experience Learning

I. J. Cox and N. H. Gehani [2] discussed the construction of robust and reliable robot systems able to handle errors arising from abnormal operating conditions. It is assumed that the robot program is logically correct but fails due to hardware or external state errors. Frédéric Souchon et. al [3] discussed how to increase reliability in multi-agent systems (MASs) and focused on the study of an appropriate exception handling system (EHS). Chrysanthos Dellarocas and Mark Klein [4] presented an experimental evaluation of a set of domain-independent services designed to handle the failure modes ("exceptions") that can occur in open multi-agent systems.

There are also some related projects which investigate experiences. The project RoboEarth (http://www.roboearth.org/) represents experiences using OWL (Web Ontology Language). RoboEarth aims for obtaining a sharable representation of the environment by combining the experiences of many robots. RACE focuses on using the experiences of a single robot to improve its future performance by an integrated sub-symbolic/symbolic and top-down/bottom-up approach. Another related project is XPERO (http://www.xpero.org/), which emphasizes on active robot experimentation to enable inductive learning.

B. Guest Detection

Human body detection and tracking is a very active area in computer vision and biometrics community. This topic has been studied for nearly 30 years and has been applied in robotics, security and entertainment. In this section, we will introduces some packages of ROS (Robot Operating System) [5]. We do not focus on the details of implementations. But we will pose emphasis on which package can provide functionality we need as the main goal is to integrate the state of the art guest detection functionality into our robot system.

To our knowledge, there are two mainstream detection approaches used for guest detection: skeleton detection and face detection. The face detection technique is mature now. The skeleton detection and tracking technique provides by RGB-D sensor like Kinect [6] is also popular and stable. The Kinect for Windows sensor and software development kit (SDK) provides the limitless possibilities offered by the Kinect technology. Nevertheless, many functions are only developed under Windows. The skeleton tracking library provided by PrimeSense provides a convenient way to extract and track people from a depth image.

pi_face_tracker [7] is a ROS package and employs OpenCV's Haar face detector together with Good Features to Track and the Lucas-Kanade Optical Flow tracker to perform face tracking in a live video stream or recorded video file. The depth information of the RGB-D sensor (Kinect) can be used to reduce the number of false positive face detections. ROS topics like "/roi" and "/target_point" published by pi_face_tracker provide regions of interest around the tracked face and PointStamped centroid of the tracked cluster.

ProcRob face_recognition [8] package provides a simple actionlib server and client interface for implement different face recognition functionalities in video stream. It can also capture, train and recognize face images by commands with different parameters.

cob_people_detection [9] is also a ROS package which provides head detection, face detection, face recognition and detection tracking functionalities. The Head Detector node first detects heads in the depth image of RGB-D data (Kinect). Then the Face Detector node determines the locations of faces within these head regions. It publishes via a ROS topic an array of the detected heads and faces, includes bounding boxes of heads and faces and 3D coordinates. Then the other packages can subscribe the topic in order to retrieve the results. In this work, we employ this package to detect and track guests.

There are also many other approaches used by robot researchers. Chris Burbridge and Lorenzo Riano [10] used a Microsoft Kinect camera attached to the end of a 7 DOF Schunk manipulator to track persons. The points corresponding to the person are extracted from the RGB-D camera's point cloud when the robot is moving.

In this work, we mainly focus on how to use guest detection to handle exceptions. Therefore, we will use the state-of-the-art guest detection technique and existing packages to handle exceptions occur in restaurant environment.

III. System Architecture

In this section, we present the main components of the modular RACE architecture (Fig. 1).

A core module of the RACE architecture is the Blackboard [11], here all Fluents are stored and updated by other modules and provided to others. Fluents, as facts or instances with start and end time, represent various knowledge about a scene, episodes, experiences, predictions and environment events.

The different reasoners, e.g., ontological reasoning, temporal reasoning, spatial reasoning, scene interpretator, receive knowledge (either hand-coded or learned) from the OWL ontology (represents a T-box in Description Logics) and post the results of reasoning back to the blackboard for further processing.

When a new planning goal is entered by the user or triggered by the guest detection module, the HTN (Hierarchical task network) Planner [12] queries the Blackboard to build its initial planning state, then sends the generated plan back into the Blackboard. The Execution Manager receives the

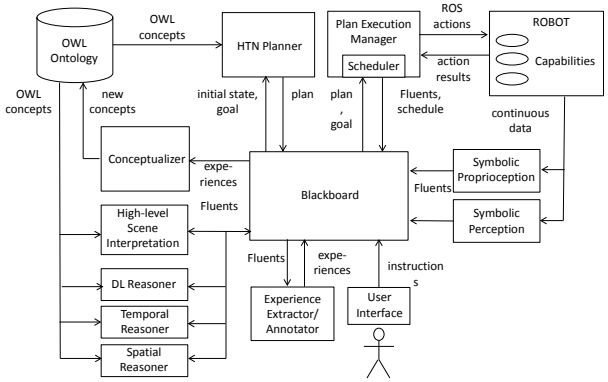

Fig. 1. The RACE Architecture

plan and starts dispatching the planned actions to the robot capabilities of the PR2. During execution, the Execution Manager monitors executed actions and writes success or failure information to the Blackboard.

In the RACE project, the PR2 employs ROS to control and execute the actions. ROS provides many basic capabilities (e.g., for manipulation and navigation) in the form of ROS actions. Some failure or exception cases can occur during the execution. The Execution Manager will be notified of the permanent failure and will in turn trigger re-planning. Therefore ROS itself can be seen as an abstract robot control architecture providing defined interfaces, e.g. via a Publish-Subscribe mechanism.

To autonomously implement and test developed algorithms and methods, our simulation infrastructure bases on ROS and Gazebo [13] is concerned. Gazebo is a 3D multi-robot simulator with a dynamic physics engine. Like Stage, it is capable of simulating a population of robots, sensors and objects, but does so in a three-dimensional world. It generates both realistic sensor feedback and physically plausible interactions between objects.

A. Guest Detection Pipeline

In RACE, a Guest Detection system is employed to provide three functionalities. The first is the task initiation of the "ServeGuest" activity. In RACE, two kinds of input can trigger the "ServeGuest" task: command input and Guest Detection input. In the situation of the "ServeGuest" task triggered by the guest detection, the robot will wait for the entrance of the guest at the door (point1 as shown in Fig. 2). Once the guest has been detected, the robot starts the "ServeGuest" task. The second function is to confirm the position of the guest before placing the mug or dish in front of the guest. As mentioned before, in a restaurant environment, if the guest sits on the opposite side of the table (or leave the sitting area) other than specified in the planning domain, the robot still brings the mug to the same place as before (which is now in front of an empty seat). This will not happen if the robot detects the position of the guest. The third function is to deal with human obstacles. For example, the robot is instructed to move mug1 to table1 and finds the path blocked by an obstacle. If the robot knows the obstacle is a person, it will wait until the person has freed the path. The latter two functions are typical cases of exception handling for an experience-based mobile cognitive system.

IV. Scenario Setup and Experiments

In RACE, we set out to prove that knowledge can enhance the competence of robots operating in complex environments such as a restaurant. In this section, several scenarios have been set to test and evaluate the effectiveness of the system. Exception handling is achieved by the Execution Manager. In this work, we focus on the exception handling functionalities provided by the guest detection package. The evaluation will test three scenarios, respectively. In each scenario, at least 5 experiments will be executed to obtain the experimental results. Some indicators like `move to mae3` have been checked with values obtained from all the experiments.

A. Scenario Setup

We first present the scenario setup in the restaurant domain which will be used for the ServeACoffee demonstration. The idea is to let the robot discover a generalization of the 3 experiences in scenarios A and B which will subsume the task in scenario C. As mentioned above, the first phase (scenario A and B) consists in an experiences-gathering process, followed by an experiences-exploitation phase (scenario C).

As shown in the Fig. 2 and Fig. 3, the counter lies near the southwestern corner of the room. At the middle of the room, there are two square tables. The robot drives from door position (point 1, close to the door) to the counter(point 2), looks for the coffee mug and picks it, moves and places it on the table(point 3).

In the restaurant environment, the robot has to transport meals and beverages to specified area. Hence we predefine the PlacingArea. The PlacingArea is the part of the table where mugs and dishes should be placed. It is a rectangle area with length of 350 mm and width of 300 mm. Distance from the edge of the table to the PlacingArea is 50 mm. The mugs, dishes and spoons can be placed anywhere in the PlacingArea. The PlacingArea of the counter is slightly different.

Scenario A: The restaurant floor plan is as shown in Fig. 3, where the robot knows the position of mug1 on the counter, the position of table1, the position of guest1 west of table1, and the regions for manipulation, sitting and placing. The user successively instructs the robot to move to counter1, grasp mug1, move to the manipulation region south of table1,

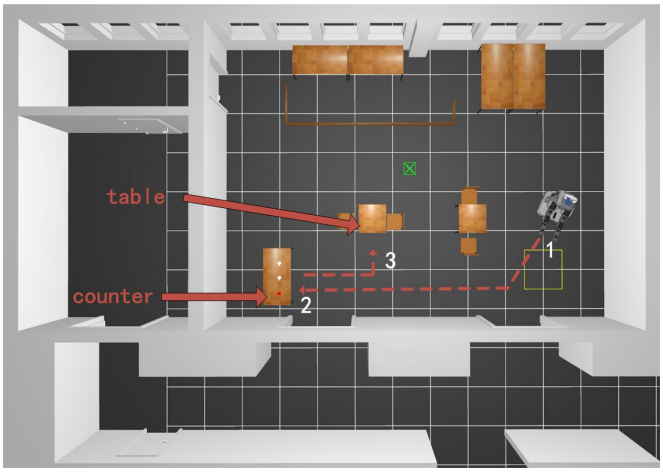

Fig. 2. Simulation Environment

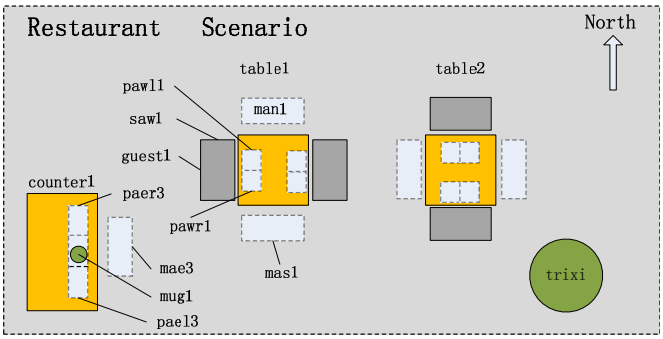

Fig. 3. Initial floor plan for ServeACoffee scenario A

and place mug1 at the placement region west of table1. The robot is told that this is a ServeGuest activity.

Scenario B: The same as scenario A, except guest2 is sitting east of table1 and the robot is instructed to move to the north of table1 and place mug1 in front of guest2. Again, the robot is told that this is a ServeGuest activity.

Scenario C: Guest3 is sitting south of table2 and the robot is simply instructed: Do a ServeGuest to guest3.

To monitor the execution of the robot, some discrepancies between the observed behavior and the ideal behavior must be checked. During the execution, four different types of errors can occur: Conceptual Errors, Perceptual Errors, Navigation and/or Localization Errors, Manipulation Errors. The Conceptual Error is used to measure the capability learning and use of knowledge to increase the robot performance. The latter three types of errors - perceptual, navigation and manipulation errors - are platform specific.

Conceptual errors arise from discrepancies between the knowledge used by the robot and the one encoded in the specification of the ideal behavior. We focus on four types of Conceptual Errors: Temporal inconsistencies, Spatial inconsistencies, Taxonomical inconsistencies, Compositional inconsistencies [14], [15].

Let $V0$ be the nominal (ideal) condition of the scenario (just like scenario A described). In $V1$ (just like scenario B and C described), the guest sits on the opposite side of the table (or leaves from the sitting area) other than specified in the planning domain. The robot still brings the mug to the same place as before (which is now in front of an empty seat). Then a perception error occurs. It means #spatial_inconsistencies $= 1$ according to the evaluation metrics [14], [15]. We can measure and evaluate the execution of the robot. The evaluation results can be found in [14], [15].

B. Experimental Results

The robot must also capture other happenings in its environment apart from plan-based activities, like guest activities and position changes of the objects in the environment. In the scenario A, the robot executes the "ServeACoffee" task according to the instructions given by a human. In scenario B, the situation is different. The guest sits on east side of the table1 other than specified in the planning domain in scenario A. According to the original plan given by the HTN planner, the robot still brings the mug to the same place (west of the table) as before. This exception will be detected by the Exception Manager. The next time the Exception Manager will ask the planer to change the plan. The guest detection will be triggered before the robot tries to place the mug on the table. If the guest detection package detect the position of the guest's face, it will first transform this frame (`/kinect_rgb_optical_frame`) to the robot frame (`/base_link`) or the world frame (`/map`), then add the position of the guest to the blackboard (a Fluent is added to the blackboard). If no guest is detected, an exception will be sent to the Exception Manager.

Fig. 4. Real environment for ServeACoffee scenario

The execution has been tested on the three scenarios, respectively. In each scenario, 5 experiments have been

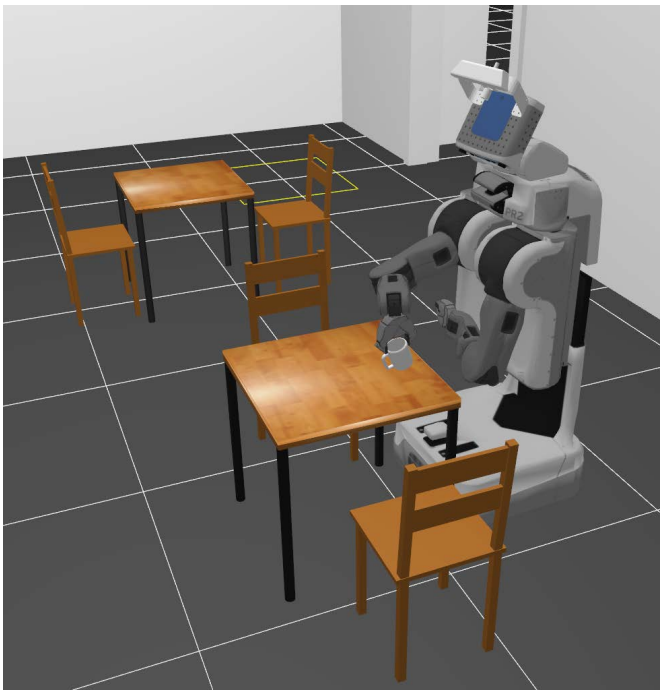

Fig. 5. Simulation environment: mug1 has been placed in pawr1

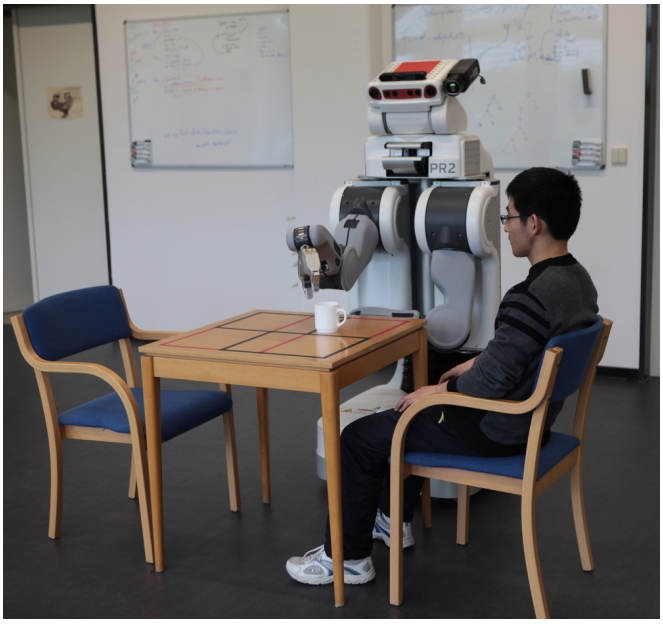

Fig. 6. Real environment: mug1 has been placed in pawr1

executed to obtain the experimental results. To collect spatial inconsistencies, the status of the robot and environment should be checked. In Tab. I, some items are listed with respect to scenario A. In scenario B and C, the corresponding item of `place mug1 in pawr1` will be `Failure` if the robot execute the task according the original planning. The reason of `Failure` is that the guest moves to a position at the table other than the one specified at the beginning. But the robot would bring the mug to the old position.

The performance of the "ServeACoffee" task is good since all the tasks have been successfully finished. The real environment for "ServeACoffee" scenario is shown in Fig. 4. Fig. 5 and Fig. 6 show that mug1 has been placed in pawr1 in the simulation and real environment, respectively. The face detection results are also shown in the following figures. Fig. 7 shows the result of the guest entrance detection situation, where the detected face is indicated in light green rectangle and the detected head is framed with a light blue rectangle. The name of the guest was also marked at the left corner of the rectangle. The pose of the detected guest face is also marked with a red arrow in the top of Fig. 7, where 3D points cloud and RGB-D results are shown.

Fig. 8 shows the guest detection results when the path has been blocked by a person. After a short while, the person frees the path and the robot completes its task. Fig. 9 shows the results of guest position confirmation before the robot tries to place the mug in front of the guest situation.

All this results show that the robot can successfully execute the "ServeACoffee" task and handle several exception situations employing the guest detection system.

Scenario A	A: ex 1	A: ex 2	A: ex 3	A: ex 4	A: ex 5
move to mae3	Success	Success	Success	Success	Success
detect mug1 on pae3	Success	Success	Success	Success	Success
grasp mug1 from pae3	Success	Success	Success	Success	Success
move to mas1	Success	Success	Success	Success	Success
place mug1 in pawr1	Success	Success	Success	Success	Success

TABLE I

SPATIAL INCONSISTENCIES METRICS OF SCENARIO A

V. CONCLUSIONS

In this paper, a Guest Detection subsystem is employed on an experience-based mobile cognitive system to handle exceptions occurring in restaurant environment. The "ServeACoffee" scenario was setup to validate and evaluate the effectiveness of the proposed approach. The experimental results show that Guest Detection succeeded with initiating the "ServeACoffee" activity and handling the exception of the guest sitting in a non-specific area and detecting the person who blocks the path. In the future, more dynamic factors and scenarios will be designed to evaluate exception handling capabilities of the experience-based mobile cognitive system in restaurant environment.

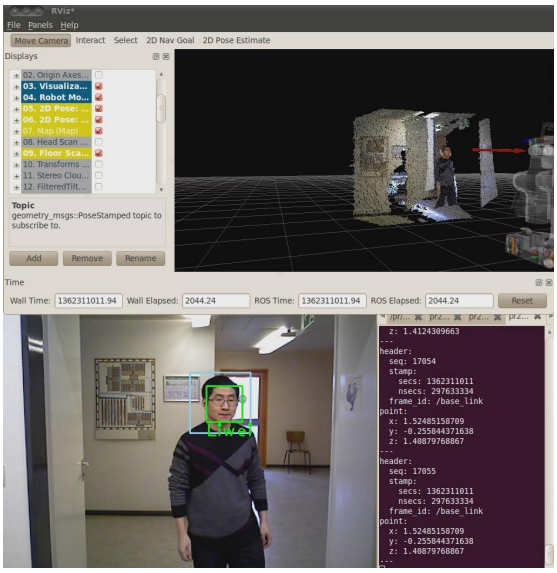

Fig. 7. Guest entrance detection

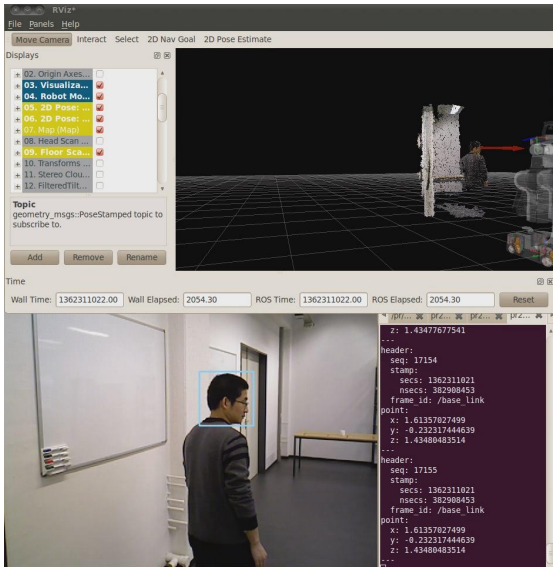

Fig. 8. Path blocked by a person

ACKNOWLEDGMENT

This work has been conducted as part of RACE, funded under the European Community's Seventh Framework Programme FP7-ICT-2011-7 under grant agreement no. 287752 (http://www.project-race.eu).

REFERENCES

[1] Willow Garage. Personal robot 2. http://www.willowgarage.com/pages/pr2/overview.

[2] I. J. Cox and N. H. Gehani. Exception handling in robotics. *Computer*, 22(3):43–49, March 1989.

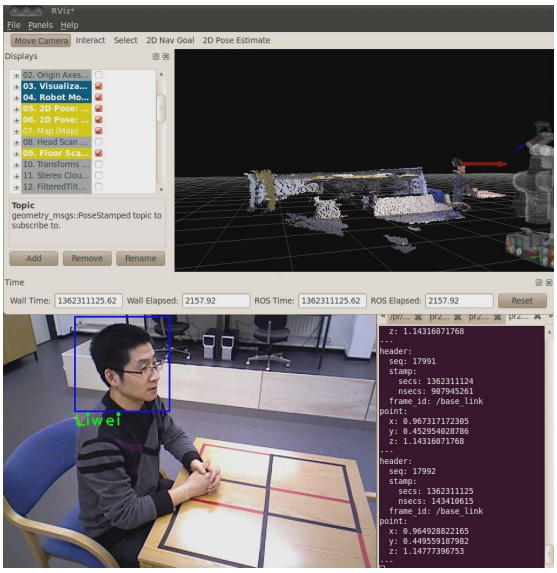

Fig. 9. Guest position confirmation

[3] Frédéric Souchon, Christophe Dony, Christelle Urtado, and Sylvain Vauttier. Improving exception handling in multi-agent systems. In *Software Engineering for Multi-Agent Systems II*, volume 2940, pages 167–188, 2004.

[4] Chrysanthos Dellarocas and Mark Klein. An experimental evaluation of domain-independent fault handling services in open multi-agent systems. In *Fourth International Conference on MultiAgent Systems*, pages 95–102, 2000.

[5] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, and Tully Foote. Ros: an open-source robot operating system. Technical report, Stanford University, 2009.

[6] Kinect for xbox 360. http://www.xbox.com/en-US/kinect.

[7] Patrick Goebel. pi_face_tracker. http://www.ros.org/wiki/pi_face_tracker.

[8] Pouyan Ziafati. Procrob face_recognition. http://www.ros.org/wiki/face_recognition.

[9] Richard Bormann. cob_people_detection. http://ros.org/wiki/cob_people_detection.

[10] Chris Burbridge and Lorenzo Riano. Person tracking and reconstruction from a mobile base with a 7 dof manipulator. git://github.com/cburbridge/uuisrc-ros-pkg.git.

[11] Sebastian Rockel, Bernd Neuman, Jianwei Zhang, Krishna S. R, Dubba, Anthony G. Cohn, Štefan Konečný, Masoumeh Mansouri, Federico Pecora, Alessandro Saffiotti, Martin Günther, Sebastian Stock, Joachim Hertzberg, Ana Maria Tomé, Armando J. Pinho, Luís Seabra Lopes, Stephanie von Riegen, and Lothar Hotz. An ontology-based multi-level robot architecture for learning from experiences. In *Designing Intelligent Robots: Reintegrating AI II, AAAI Spring Symposium*, Stanford (USA), March 2013.

[12] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[13] Gazebo. http://gazebosim.org/.

[14] Liwei Zhang, Sebastian Rockel, Federico Pecora, Luis Seabra Lopes, Alessandro Saffiotti, and Bernd Neumann. Deliverable d5.1 - evaluation infrastructure. Technical report, European Commission - Information and Communication Technologies - Seventh Framework Programme, November 2012.

[15] Liwei Zhang and Sebastian Rockel. Deliverable d5.2 - year-1 demonstrator. Technical report, European Commission - Information and Communication Technologies - Seventh Framework Programme, January 2013.