Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Project paper 1

SoSe2013

Florian Johannßen

NAO in the cloud

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Florian Johannßen

Florian.Johannssen@haw-hamburg.de

Thema

Nao in the cloud

Stichworte

Knowledge sharing, Cloud Robotics, Nao, RoboEarth, Robot Operating System

Kurzzusammenfassung

Diese Projektausarbeitung berichtet über erste praktische Erfahrungen mit dem Thema „Knowledge sharing for robots via cloud services“, welches im Rahmen der Masterarbeit bearbeitet wird. Die Thesis verfolgt einen Wissensaustausch zwischen den humanoiden Nao Robotern über cloud services, um deren Lernmeachanismen zu verbessern. Damit die Nao Roboter mit der cloud kommunizieren können, wird im Rahmen des Projektes eine Archtektur erarbeitet, welche von den Hardware-Eigenschaften der Roboter abstrahiert und Mechanismen bereitstellt, um auf Informationen des cloud service RoboEarth zuzugreifen. Diese Arbeit führt zwei Experimente mit der Architektur durch, um erste Testergebnisse mit den Komponenten, Schnittstellen und Abstraktionsschichten zu erzielen.

Florian Johannßen

Title of the paper

Nao in the cloud

Keywords

Knowledge sharing, Cloud Robotics, Nao, RoboEarth, Robot Operating System

Abstract

This paper presents the first practical experiences with the topic “Knowledge sharing for robots via cloud services” which is part of the master thesis. The intention of the thesis is to describe a knowledge sharing among several humanoid Nao robots for improving their learning mechanism with the aid of cloud services. This project introduces an architecture for providing a cloud-enabled Nao robot which is able to access information from the cloud service RoboEarth. Besides, it shows some tests to get experiences with the components and middlewares of the architecture.

Table of content

1 Introduction

1.1 Motivation

The Internet has become one of the most important communication media. It provides the opportunity to publish and retrieve knowledge globally. We are able to solve unknown tasks efficiently and share knowledge with other people. If you are involved in the research area of robotics, it would be preferable to apply this paradigm to robots. Nowadays, companies like Aldebaran Robotics[1] and Willow Garage[2] are able to deliver wireless capable and programmable robots with abstract interfaces. The specific tasks, such as face recognition, voice recognition and path planning are mostly solved. Thus the preconditions have been created to connect robots with the internet. Kuffner [1] and Quintas et al. [2] have introduced the topic Cloud Robotics. This idea provides a physical separation between the hardware and software components of the robot. The conventional hardware devices of a robot, such as sensors, actuators, cameras and speakers are still on the robot. The difference to the usual approach, which designs the software on the robot, is that the brain of the robot is outsourced to remote servers. This approach can be used as in Inaba [3] to outsource time consuming tasks on powerful remote servers. In addition, it offers the possibility that robots communicate with each other to improve their learning mechanism. The idea of knowledge sharing for robots describes the problem how to exchange information between heterogeneous robots to benefit from the experience of others.

1.2 Vision

The concrete target of my master thesis [4] deals with the realization of the still unexplored approach of knowledge sharing for robots via cloud services. It explores how heterogeneous robots are able to improve their learning behavior with exchanging knowledge via cloud services and it presents the challenges which have to be solved. Firstly, the practical part of the thesis will handle this topic with two homogenous robots which are connected with a cloud service for downloading and executing robot plans. The Nao[3] robot and the cloud service RoboEarth will be used for the implementation. The master thesis will include the implementation of an interface between the humanoid robot Nao and the RoboEarth[4] cloud service as well as the realization of a scenario in which several Nao robots download and execute information from the cloud service.

[1] http://www.aldebaran-robotics.com
[2] http://www.willowgarage.com
[3] http://www.aldebaran-robotics.com/en/Discover-NAO/videos-gallery.html
[4] http://www.roboearth.org

This project realizes the preconditions to connect the Nao robot with the cloud. It explores and evaluates the required components for implementing this approach.

2 Infrastructure

This chapter describes the components and interfaces which are needed to connect the Nao-Platform with the cloud service RoboEarth.

2.1 Architecture

One of the main problems of the robots is, that they are limited in many ways. The computing power and hardware resources are bounded by costs and their physical properties. Moreover, robots are usually on one´s own and only programmed for one specific area. So the behavior of the robot is greatly limited by inflexible programming. The standard SDK[5] of a robot doesn´t provide solutions for sharing executable information with heterogeneous robots. The figures [1] and [2] present an architecture which provides a knowledge sharing among different robot platforms with the aid of cloud services to solve this problem.

Figure 1: Robots in the cloud [5,6,7]

Firstly, we implement this scenario only with the Nao robot to avoid problems because of the heterogeneity of the robots.

[5] Software Development Kit

The required components and middlewares which provide a cloud-enabled Nao are already installed on each laptop. The robots communicate with the cloud by accessing this infrastructure on a remote laptop. It is possible to install these components directly on the Nao robot as well to avoid this extra node, but the remote connected approach is more secure.

The next figure shows the components and middlewares which are installed on the laptop and it provides an architecture to connect the Nao robot with the cloud service RoboEarth. These components and middlewares will be described in the following chapters.

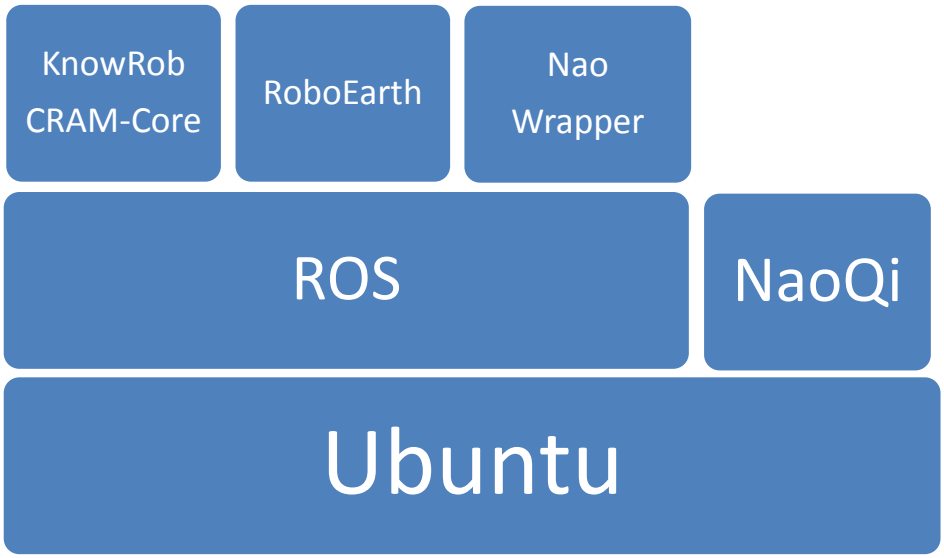

Figure 2: Architecture

2.2 Operating system

The open source operating system Ubuntu 12.04 is already installed on the laptop for the practical experiments, because it is superior compatible with the other components than other operating systems like Windows or Mac OS X.

2.3 Nao SDK

The humanoid robot Nao by Aldebaran Robotics provides the NaoQi[6] SDK. This includes a programming interface to develop applications with different programming languages like C++, Java or Python. With the aid of NaoQi, developers are able to control the Nao hardware components. The experiments are executed using the NaoQi SDK 1.12.5. The core of the NaoQi SDK represents the main broker which acts as a server on the robot.

[6] http://www.aldebaran-robotics.com/documentation/dev/naoqi/index.html

Aldebaran Robotics [8] presents this framework which includes several modules to access the joints, face recognition and other functions. The main broker decides which modules should be executed. Figure 3 shows the main broker with most of the Nao modules.

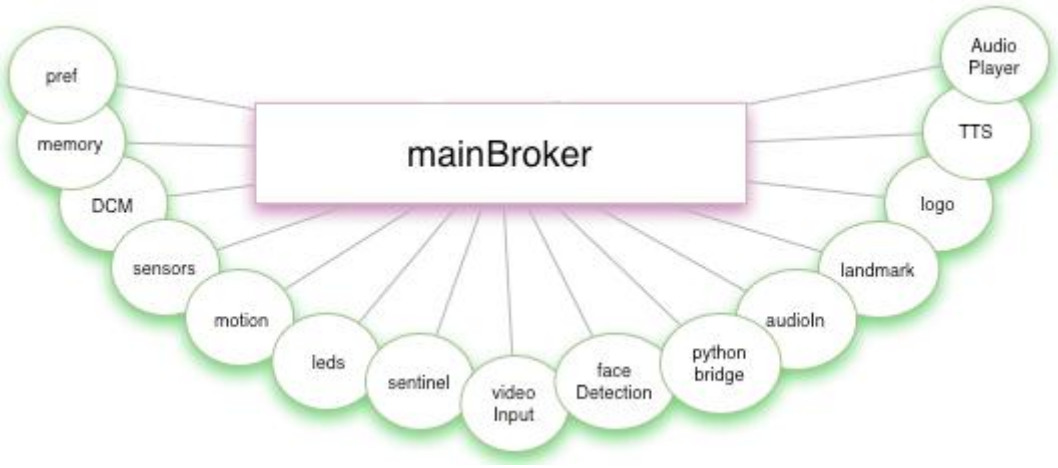

Figure 3: NaoQi [8]

2.4 Middleware

The first target of the master thesis deals with knowledge sharing among homogeneous Nao robots. Furthermore, this approach should be applied on heterogeneous robots with different capabilities and hardware resources. An abstraction layer over the robot specific hardware is needed to satisfy this requirement. I have evaluated some middlewares like Microsoft Robotics Developer[7] Studio and ROS[8] by Willow Garage. ROS has been chosen as preferred abstraction layer because of the popularity and the support of many different robot platforms. With the aid of ROS, it is possible to command the robot to walk from place A to place B without thinking about which walking algorithm will be used. This meta operating system is extendable with modules, called stacks. There are modules for many robots like Nao, TurtleBot[9] or PR2[10].

2.5 Stacks

Due to the modularity of ROS we are able to install different stacks for our requirements. This chapter describes the ROS-Stacks which are used and how they communicate with each other.

[7] http://www.microsoft.com/robotics
[8] Robot Operating System: http://www.ros.org
[9] http://turtlebot.com
[10] http://www.willowgarage.com/pages/pr2/overview

Nao-Wrapper

Firstly, we have installed an abstraction layer over the NaoQi SDK. This Stack is developed by the University of Freiburg [9] and wraps the NaoQi API in Python to control the Nao robot on an abstract level. The Nao-Stack provides access to the sensors, odometry, cameras, teleoperation and speech recognition. Besides, it is possible to manipulate the joints of the Nao. The Nao-Wrapper mainly includes the following packages:

Nao package	Description
nao_robot	Provides basic functionality to access odometry, cameras, sensors and joints
humanoid_msg	Includes some basic services for humanoid robot navigation
nao_common	Contains tools to run the Nao robot remotely on the PC. It provides mechanisms to monitor the joints and odometry remotely.

Table 1: Nao-Wrapper packages

RoboEarth

Zweigle et al. [10] introduced RoboEarth which represents a World Wide Web for robots. This cloud service can be used by heterogeneous robots to share information among each other. Due to RoboEarth robots are able to share information and experiences about objects, actions and environments. This work aims to realize a knowledge sharing among Nao robots with the aid of the RoboEarth-Stack which is integrated in ROS, which is shown in figure 4.

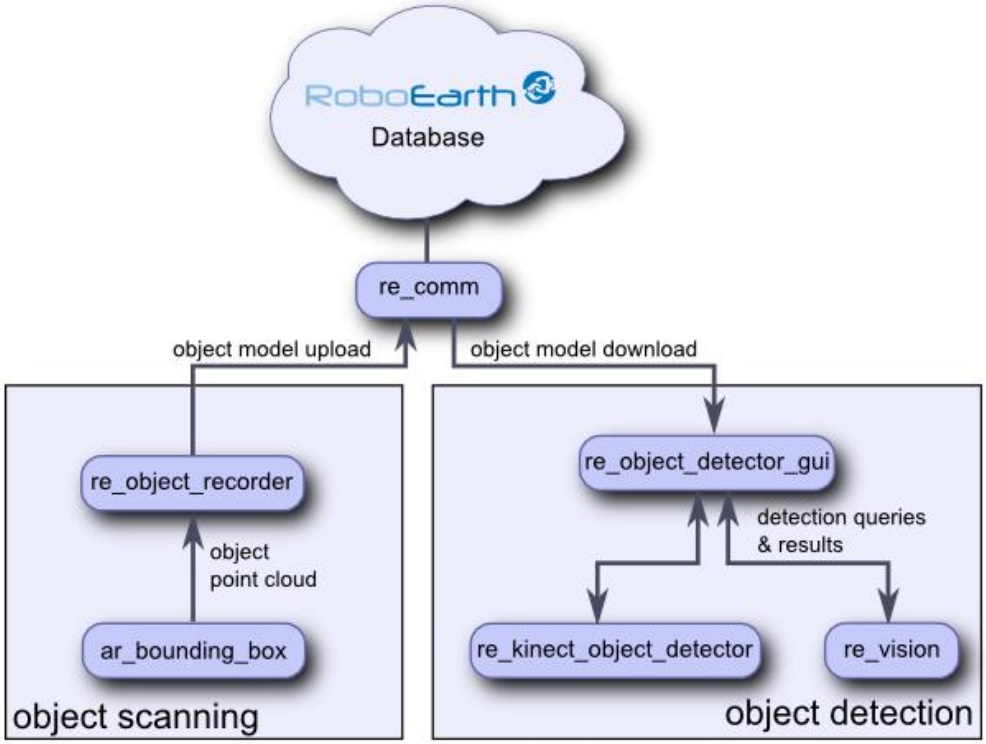

Figure 4: RoboEarth-Stack [11]

It contains packages for object scanning, object detection and interfacing RoboEarth. Note, that the packages re_comm and re_ontology are mainly relevant for the master thesis.

RoboEarth package	Description
re_comm	Provides an interface to download, upload, update and delete action recipes, object models and environments.
re_ontology	Contains the base RoboEarth ontology. The OWL ontology extends KnowRob with detailed descriptions of actions and objects which are required for RoboEarth.
re_object_recoder	Provides a user interface to create, store and upload object models using a Kinect camera.
re_object_detector_gui	Represents a user interface to download object models from the RoboEarth database.

Table 2: RoboEarth packages

KnowRob

This component is developed by Tenorth [12] and represents a robotic framework to acquire and to reason about robot specific knowledge. With the aid of KnowRob, robots are able to send semantic requests for a plan to the database. If a robot sends a request for a plan like grasping a bottle to RoboEarth, KnowRob scans the database. Furthermore the KnowRob component checks the capabilities of the robot against the requirements of the task and generates the CRAM plan for the requested task. Besides, the KnowRob component reasons for example that milk is drinkable and can be found in the fridge. If a robot doesn´t find milk in the kitchen it can request KnowRob for assistance. The next table shows the most important subpackages:

RoboEarth package	Description
ros_prolog	Represents a simple ROS-Wrapper around swi-prolog
ias_knowledge_base	Includes the KnowRob ontology which is defined in the Ontology Web Language (OWL)
knowrob_common	Contains utilities to handle OWL-files
knowrob_objects/ actions/ enviroments	Provides basic functionality to access and to handle objects, actions and environments
mod_vis	Visualization
mod_srdl	Includes descriptions of robot components and capabilities to match these against requirements of actions

Table 3: KnowRob packages

CRAM Plan Language

Beetz et al. [13] have developed the Cognitive Robotic Abstract Machine Plan Language which extends the functional programming language Common Lisp to describe abstract robot control programs. The KnowRob component is able to translate the abstract plan from the RoboEarth database like grasp a bottle to the CRAM Plan Language. CRAM is also providing a ROS-Stack by Mösenlechner [14] which is called cram_core. It includes the following substacks:

RoboEarth package	Description
cram_language	Represents a Common Lisp extension to program abstract robot plans
cram_reasoning	Includes a full-featured Prolog interpreter which is implemented in Common Lisp and provides algorithms for pattern recognition
cram_designators	Provides meta information about objects and actions
cram_process_modules	Represents the interface to a specific robot

Table 4: CRAM packages

3 Experiments

During the project I have processed some experiments to make the first experiences with the ROS-middleware and the several stacks.

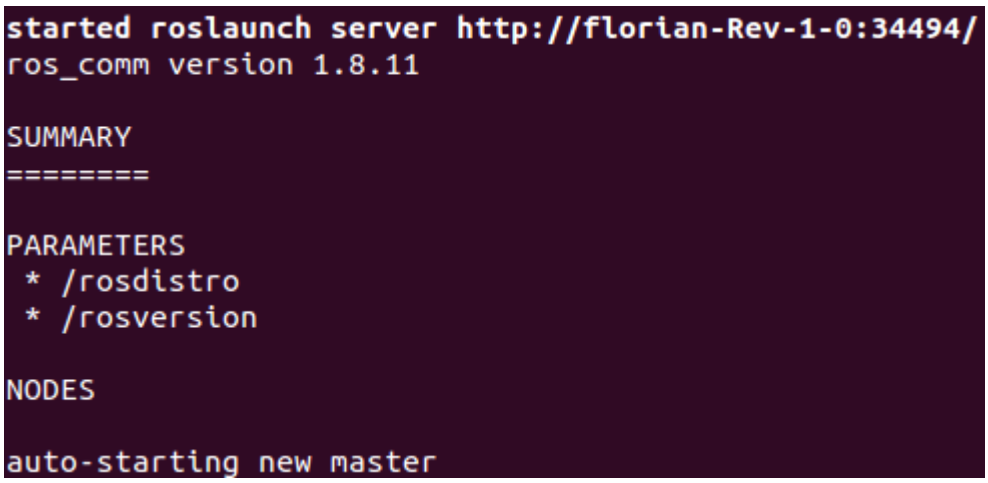

Figure 5: Roscore

The roscore is a standard process. It is launched by using the roscore command and includes a collection of nodes and programs which are required for a ROS-based system. You have to start the roscore before you execute the other stacks.

3.1 Nao controlling via ROS-Wrapper

The first experiment aims to connect and control a simulated Nao robot with the Nao-ROS-Wrapper. Before we connect the Nao-Wrapper with the NaoQi broker, we have to start the NaoQi process in figure 6.

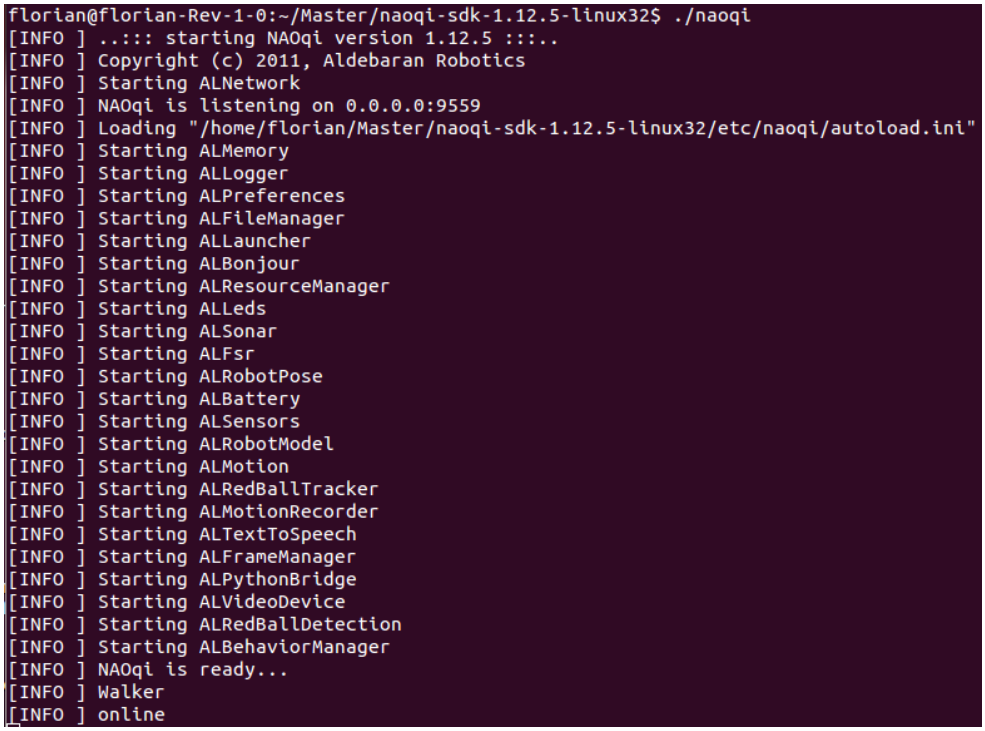

Figure 6: NaoQi-Broker

When the main broker is started on port 9559, it also starts several modules which are listed in figure 6. The next figure shows the output after starting the process Nao-Wrapper which joins the main broker of the NaoQi framework.

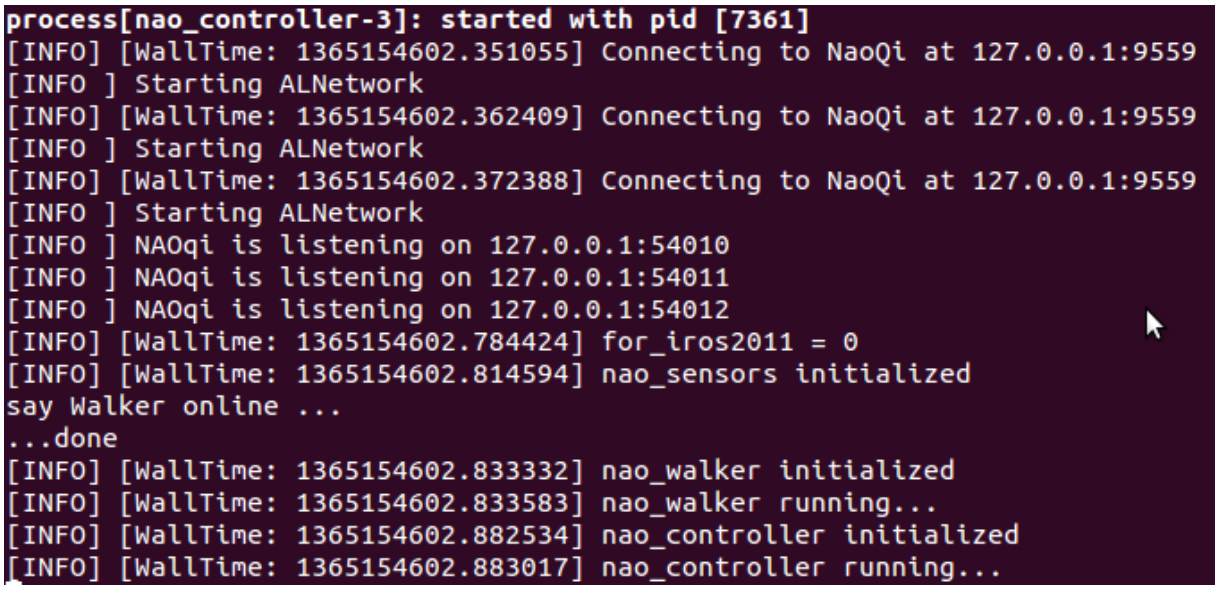

Figure 7: Nao-Wrapper

After starting the Nao-Wrapper we are able to control the Nao robot with ROS commands. We can move the Nao robot in the x-direction with the next command.

```
$ rostopic pub -1 /cmd_vel geometry_msgs/Twist
'{linear: {x: 1.0, y: 0.0, z: 0.0}'
```

The command `rostopic pub` is defined with the syntax: `<topic-name> <topic-type> data`. So, the coordinates will be sent to the topic cmd_vel. Topics represents named channels over which ROS-processes exchanging messages. You can monitor the values of the torso and the joints with the echo package of the rostopic command.

```
$ rostopic echo /torso_odometry
$ rostopic echo /joint_states
```

3.2 Definition of abstract plans with TurtleSim

The second experiment deals with the definition of abstract plans in the CRAM plan language for the simple robot simulator TurtleSim[11]. TurtleSim provides a simple simulator for teaching ROS concepts. We have chosen rosemacs as REPL[12] to implement CRAM plans. This is an emacs extension for the ROS-middleware. So, it is possible to execute ROS-commands like `roscore` or `rosrun` directly from emacs. We can start rosemacs with the ROS-command `rosrun roslisp_repl repl`.

Firstly, I have created an ASDF[13] project. ASDF provides mechanisms to organize and to build a Lisp project. This project imports the packages roslisp, cram_language and turtlesim_msg. The following command can be used to compile the project in the REPL rosemacs.

```
(ros-load:load-system "cram_tutorial" :cram-tutorial)
```

Before we start the TurtleSim simulator, we have to start the roscore again. With aid of rosemacs, we can start the TurtleSim ROS-process with the following command.

```
(start-ros-node "cram_tutorial_client")
```

The next step is to define a simple plan for moving the Turtlesim. This is implemented directly in rosemacs as a top level plan which shows the next code lines.

```
(def-plan move-to (goal &optional)
 ...
```

Now, we are able to move the simulated robot TurtleSim with the CRAM Plan Language.

[11] http://www.ros.org/wiki/turtlesim
[12] Read-Eval-Print-Loop
[13] Another System Definition Facility

4 Evaluation

The performed experiments generated first results to provide a cloud-enabled Nao. The first test was designed for controlling the NaoQi SDK with ROS commands. The access of the Nao specific programming interface was very comfortable with the abstract ROS commands of the Nao-Wrapper. I was able to control the Nao to an exact position and I received the coordinates of the torso and joints during the walk. On the other hand, with the added middleware some latitude will be lost to program the Nao robot. Finally, the Nao-Wrapper is well integrated in ROS and satisfies my requirements to provide a knowledge sharing.

The second experiment tested the compatibility of the CRAM Plan Language with the ROS middleware. With the aid of the REPL rosemacs, it's very easy to adapt top level plans to the ROS abstraction layer.

5 Outlook

The experiments of this project have shown that it is possible to program abstract CRAM plan for the ROS middleware. The next part of my work will include the implementation of an interface between the Nao-Wrapper and the RoboEarth cloud service. In the next project [4] I will implement a process module which provides an interface to receive CRAM plans from RoboEarth and map these instructions to the Nao specific ROS commands. Besides, some experiments will show the usability and correctness of the process module.

6 Conclusion

This work presented the first practical experiences with the topic "Knowledge sharing for robots via cloud services". The target of the master thesis is to realize this approach with the aid of the humanoid robot Nao and the cloud service RoboEarth. This project introduced an architecture which provides a cloud-enabled Nao robot. It described the robotic middleware ROS which represents an abstract layer over the Nao specific hardware. Besides, it introduced the required ROS-Stacks Nao-Wrapper, RoboEarth, CRAM-Core and KnowRob which are needed to connect the Nao robot with RoboEarth. Furthermore, it included two experiments which tested the compatibility of the Nao-ROS-Wrapper with the CRAM Plan Language. This language will be used for the next project to download and to execute abstract plans from RoboEarth on the Nao robot. Finally, it showed the next working steps in an outlook.

List of figures

List of tables

References

1. J. Kuffner. *Robots with their Heads in the cloud.* 2011

2. J. Quintas, P. Menezes, J. Dias. *Cloud Robotics: Towards context aware Robotic Network.* 2011

3. M. Inaba. *Remote Brained Robots.* Tokio. 1993

4. F. Johannßen. Knowledge sharing for robots. 2013

5. https://kforge.ros.org/turtlebot/trac/chrome/site/turtlebot320.png (Access on 10.05.2013)

6. http://ftp.isr.ist.utl.pt/pub/roswiki/attachments/Robots(2f)Husky/husky-a200-unmanned-ground-vehicle-render.jpg (Access on 10.05.2013)

7. http://asep-championship.com/wp-content/uploads/2011/11/NAO-4_cutout.png (Access on 10.05.2013)

8. http://www.aldebaran-robotics.com/documentation (Access on 10.05.2013)

9. http://www.ros.org/wiki/Robots/Nao (Access on 10.05.2013)

10. O. Zweigle, R. Molengraft, R. d´Andrea, K. Häussermann. RoboEarth – connecting Robots worldwide

11. http://www.ros.org/wiki/roboearth_stack (Access on 10.05.2013)

12. http://www.ros.org/wiki/knowrob (Access on 10.05.2013)

13. M. Beetz, L. Mösenlechner, M. Tenorth. CRAM - A Cognitive Robot Abstract Machine for Everday Manipulaiton in Human Enviroments. 2010.

14. http://www.ros.org/wiki/cram_core (Access on 10.05.2013)

15. M. Quigley et al. ROS: an open-source Robot Operating System. 2010