Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Project paper 2

WeSe2013

Florian Johannßen

NAO Robots in the Cloud

An Interface to Execute Abstract Plans

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 Robot in the Cloud – An Interface to Execute Abstract Plans

Stichworte

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

Kurzzusammenfassung

Diese Projektausarbeitung ist Teil einer Masterarbeit, die sich mit dem Wissensaustausch zwischen Robotern beschäftigt, um deren Lernmechanismen zu verbessern. Das Projekt realisiert eine abstrakte Schnittstelle, sogenannte Prozess-Module, um auf dem humanoiden Roboter Nao abstrakte Pläne auszuführen. Es werden drei Prozess-Module präsentiert, um den Nao Roboter auf einer abstrakten Ebene zu manipulieren, zu navigieren und dessen Sprachkomponente zu steuern. Dazu wurden mehrere Experimente durchgeführt, welche die Benutzbarkeit und Flexibilität dieser abstrakten Schnittstelle demonstrieren. Mithilfe dieser Abstraktionsschicht soll es möglich sein, abstrakte Pläne vom RoboEarth Webservice herunterzuladen und auszuführen.

Florian Johannßen

Title of the paper

Nao Robot in the Cloud – An Interface to Execute Abstract Plans

Keywords

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

Abstract

This project is part of the master thesis which investigates knowledge sharing among robots to improve their learning mechanism. This work creates an abstraction layer, called process modules, to execute high-level plans on the humanoid Nao robots. It implements process modules to manipulate the actuators of the Nao as well as to navigate the robot to a specific position. Besides, it develops an interface to control the speech module of the Nao robot via abstract commands. Through this approach it should be possible to download and execute high-level plans from the RoboEarth web service on the Nao platform.

Table of contents

List of figures

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. This area of research has been most recently related to robotics, enabling researchers 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 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 Objective Target

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. Figure 1 delineates the approach of cloud robotics. The robots communicate with the cloud by accessing a remote laptop which provides the needed components and middlewares. It is possible to install all components directly on the robot as well to improve the performance, but the remote connected approach is more secure.

[1] http://www.aldebaran-robotics.com
[2] http://www.willowgarage.com

Figure 1 presents an architecture to share knowledge among different robot platforms with the aid of cloud services.

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

Firstly, the practical part of the thesis will handle this topic with two homogenous robots which are connected with a cloud service to download and execute abstract plans. The Nao robot and the cloud service RoboEarth will be used for the implementation. The research hypothesis of this work describes the problem of how a Nao robot is able to transform and execute high-level plans from the cloud service RoboEarth. This includes the challenge of how it is possible to control a robot on an abstract level without thinking about low-level tasks. Besides, the master thesis introduces another way to give robots the possibility to share information among each other.

This thesis will include the implementation of an interface between the humanoid robot Nao and the RoboEarth[3] cloud service as well as the execution of a scenario in which several Nao robots download and execute information from the cloud service.

The project carried out by Johannßen [8] realized the preconditions to connect the Nao robot with the cloud. It introduces an architecture which provides a cloud-enabled Nao robot and evaluates the main components ROS[4], RoboEarth[5], CRAM[6] plan language, Nao-Wrapper[7] and NaoQi[8]. With the aid of this architecture a Nao robot is able to be controlled in an abstract way. This project builds upon [8] and aims to implement an interface between the Nao robot and the RoboEarth cloud service. This interface which is called process module will be used to transform abstract plans from RoboEarth to low-level Nao specific commands.

---

[4] Robotic Operating System [9]
[5] www.roboearth.org
[6] Cognitive Robotic Abstract Machine [10]
[7] ROS stack for the Nao robot [11]
[8] Software Developer Kit for the Nao robot

Every robot which wants to process information from RoboEarth needs to provide process modules for the robot specific low-level tasks, like manipulation, navigation, perception and speech processing.

Through process modules it is possible to develop abstract plans, which are robot independent. Process modules offer actions to grasp an object, to move to a position or to pronounce a sentence. If someone develops an abstract plan, it is insignificant to know how a process module navigates the robot as well as it is unimportant to know which path planning algorithm or face detection policy will be used. Abstract plans can be executed on heterogeneous robot platforms by substitution of the process modules. The project [8] described that it is possible to download plans which are implemented in the CRAM plan language from RoboEarth. This project implements the needed process modules for the Nao robot to execute such plans.

2 Design

This section introduces a design of the process modules for the Nao robot to execute abstract plans. It describes the functionality as well as the internal mechanisms of the process modules.

Process Module

This component presents an abstraction layer over the robot-specific hardware. Generally, this approach aims to execute the same high level plan, like serve-a-drink on heterogeneous robot platforms. A process module is part of the CRAM system which is described in the project by Johannßen [8]. The Cognitive Robotic Abstract Machine Plan Language, developed by Lorenz Mösenlechner [10], extends the functional programming language Common Lisp with the help of domain specific functions and will be used to describe abstract control programs for robots. This language is based on the concepts of the reactive plan language by Drew McDermott [12].

A process module triggers the required hardware actions to control the robot if an abstract task should be executed. The number of process modules depends on the robot and its application domain. A household robot has to provide at least process modules for manipulation, navigation, perception and speech output. A high-level plan interacts only with the robot through these modules. This project implements process modules to manipulate and navigate the Nao robot as well to process text to speech. A perception process module could be integrated as well during the master thesis. With the aid of this interface a high-level plan would be able to access face detection and speech recognition. The main requirement of a process module is to map high-level commands to low-level instructions as well as the encapsulation of the robot specific hardware components.

Figure 2 shows the integration of the process modules into the architecture of the project [8]. They are built on top of the Nao-CRAM-ROS environment which are already implemented and explained in the project [8].

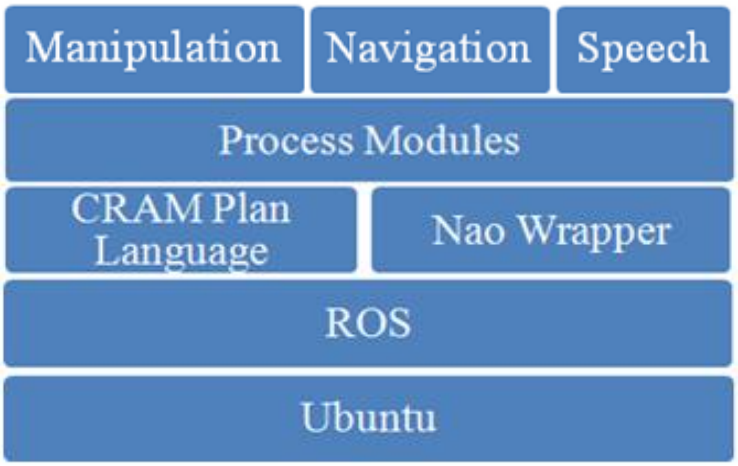


Figure 2: Cloud-enabled Nao architecture

This project uses the CRAM plan language which provides features to implement process modules. A process module acts as a black box as shown in figure 3. The input is always a designator[9] which contains the meta information from the environment like a target position to navigate the robot or the location of an object which should be grasped via the manipulation interface. The designators concept will be used for dynamic plan parameterization. There are three kinds of designators in the CRAM system, especially for objects, locations and actions. An object designator specifies the physical information of a cup, table or chair. A location designator describes positions of objects. Action designators contain all needed information to execute a task successfully.

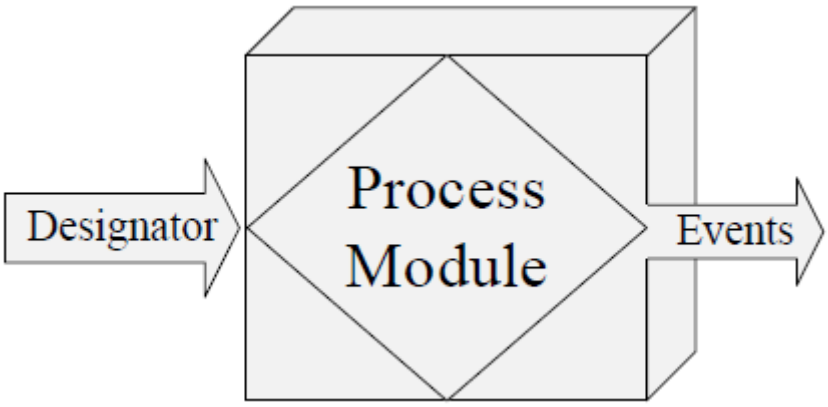


Figure 3: Process Module as black box

With the aid of the designator concept robots are able to handle dynamic and flexible plans. The execution of a plan can be changed at runtime without restarting the plan. A process module receives a designator as parameter and resolves these high-level descriptions to numeric control values. Furthermore, these resolved values will be forwarded to the

[9] http://wiki.ros.org/cram_designators

specialized low-level routines for execution. These low-level routines are centered in ROS-Nodes, which represent own processes. The Nao-Wrapper created by the University of Freiburg [11] provides several ROS-Nodes, like nao_controller, nao_walker, nao_sensors, nao_speech and nao_path_follower. More information about the Nao-Wrapper can be found in [8]. These nodes communicate among each other via ROS-Topics. These message streams represent a URL[10]-like approach to transfer information among nodes. Nodes are able to send messages via the publish/subscribe method, ROS-Services or via the ActionLib[11] interface. The process modules use these three communication approaches to call the low-level controllers of the Nao robot.

After executing the action, the process module creates events to influence the belief state of the robot. The belief state of the robot includes all information about the execution environment which the robot assumes they are correct. This internal state contains also the set of known object as well as its locations and properties. A successfully executed action of the navigation module creates a event like moveTo(position) and will be archived.

Manipulation Process Module

This interface should provide the following actions to manipulate the low-level components of the robot:

- Moving the arms
- Ranging the torso
- Turning the head
- Opening and closing the grippers

Firstly, the process module should be able to receive an action designator which describes what kind of action can be executed as well as how this action should be performed. Moreover, prolog rules have to be defined to resolve the meta information to numeric values. The next step describes to access the corresponding Nao actuator by calling a ROS-Action including the resolved plan parameter. To solve this challenge the ActionLib interface will be used which is part of the ROS middleware. Every ROS-enabled robot has to implement the ActionLib interface to map abstract commands to low-level instructions. The manipulation process module is using the nao_controller ROS-Node, which is part of the Nao-Wrapper. This node provides a Joint-Trajectory-Action-Server to control the Nao joints.

The resolved meta information will be wrapped into a ROS-Action-Goal. Action-Client and Action-Server communicate via the ROS Action Protocol with each other, which specifies the messages: goal, result, feedback, cancel and status. The nao_manipulation_node acts as an Action-Client by sending an Action-Goal to the Action-Server. The nao_controller receives the goal and executes the requested task like moving the arm or opening the gripper.

Note that the ActionLib calls are working asynchronously. This is well-founded, because many computations are long-running. Hence, it is advantageous to control another component of the robot while the Action-Server is calculating the result.

[10] Uniform Resource Locator
[11] http://wiki.ros.org/actionlib

The next figure provides an overview of the communication between the nao_manipulation_node and the nao_controller via ROS-Action-Messages. During the execution a server is able to send feedback messages to the client. After the execution the client receives the result from the server which contains the target positions of the Nao joints. Furthermore, the client is able to subscribe status information related to the execution.

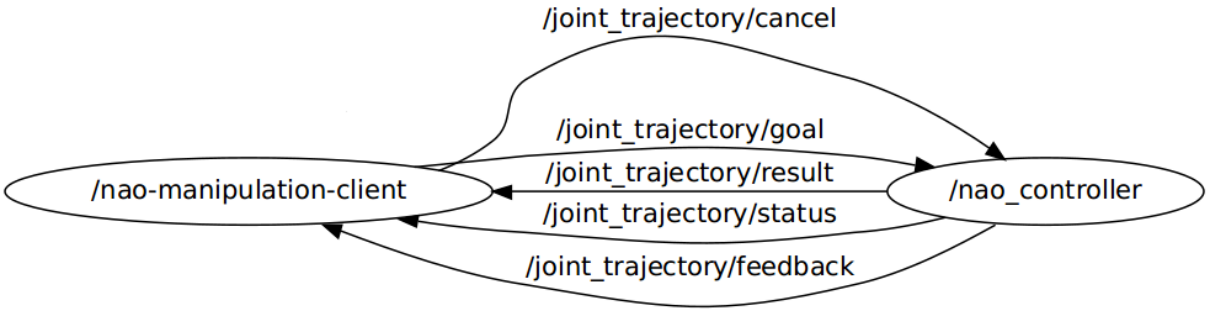


Figure 4: ROS-Nodes for manipulation

Using this concept it is possible to create Action-Clients in LISP or Java and on the other side Action-Severs in a different programming language like C++ or Python. This process module includes one corresponding Action-Client which delegates the resolved meta information to the Joint-Trajectory-Action-Server by sending an Action-Goal for every manipulation task.

Navigation Process Module

With this interface it is possible to move the Nao robot to an explicit position via high-level functions. It receives a localization designator which contains all needed meta information for the target position and resolves it to numeric values. Besides, this module delegates these values to the Nao control layer for their execution. The ROS-Node nao_path_follower is used, which implements an Action-Server to navigate the robot. This process module contains only one Action-Client. The client wraps the resolved location designator to an Action-Goal and sends this message to the Action-Server which is provided by the nao_path_follower. Furthermore, this ROS-Node commands the Nao to walk to the target position. The following figure shows the communication via ROS-messages between the ROS-Nodes nao_path_follower and nao_navigation_client.

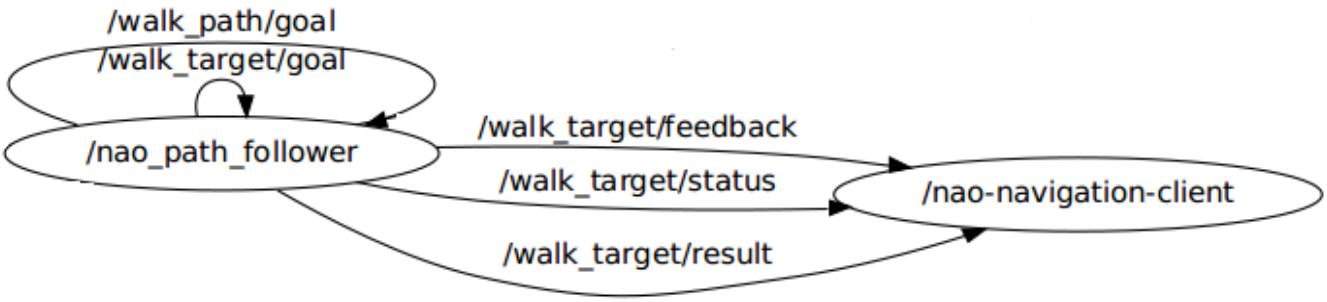


Figure 5: ROS-Nodes for navigation

The navigation process module creates a nao_navigation_client. This ROS-Node receives the temporary position through the /walk_target/feedback topic. After reaching the target position the nao_path_follower sends the result back to the nao_navigation_client.

Speech Process Module

This module has the responsibility to translate text to speech. The next ROS graph shows how a nao_speech_client sends a sentence, which is wrapped in a ROS-Goal to the nao_speech node. This process module uses the publish/subscribe pattern to realize the communication between the ROS-Nodes. The nao_speech_client publishes the sentence to the nao_speech node which has already a subscription on this topic. Finally, the nao_speech node delegates the message to the speech component of the NaoQi SDK.

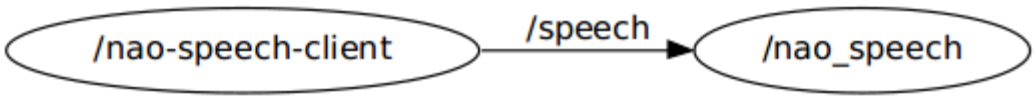


Figure 6: ROS-Nodes for text to speech

3 Experiments

This part of the paper shows some practical experiences with the implemented process modules for the Nao robot. All experiments are performed on the infrastructure described in figure 2. This chapter shows practice tests for the implemented process modules. The simulation tool Choregraphe by Aldebaran Robotics [13] is used to display the behavior of the Nao robot. The following trials use the Emacs[12] environment to develop and execute abstract plans in the CRAM plan language. The experiments aim to investigate the possibility to access the Nao process modules (:manipulation, :navigation, : speech) by high-level plans. These experiments are used to analyze the internal steps of the process modules and to show a detailed understanding how a process module works.

Setup

A special setup is needed to perform the experiments. Before we are able to access the process modules, we need to start these separately in an own thread.

```
(par
  (cpm:pm-run 'nao-manipulation-process-module :manipulation)
  (cpm:pm-run 'nao-navigation-process-module :navigation)
  (cpm:pm-run 'nao-speech-process-module :speech))
```

Therefore, we are using the utility methods which are implemented in the ROS-Package cram-process-modules by the University of Munich [14]. Now, we can use the process modules known as manipulation, navigation and speech.

[12] http://www.gnu.org/software/emacs/

3.1 Manipulation Process Module

This experiment shows some practical experiments to control the Nao robot with the aid of the process module. The idea of the process module is explained in the design chapter before. High level plans should be able to call actions like moving the arm or opening the gripper by accessing this interface. This experiment tests the usability of the manipulation process by execution of some actions which are specified in the design chapter.

Meta information

Note that the meta information about where to turn the head has to be provided by the environment and will be forwarded to the process module. The following variable defines the input for the action turning the head. Both values are radians. The first one describes a rotation around the y-axis about 57.29°. The second value represents a rotation around the x-axis about -114.59°.

```
(defvar head-position vector (1 -2))
```

The target position of the Nao left arm defines a 6D vector of radians to control the joints (LshoulderPitch, LshoulderRoll, LelbowYaw, LelbowRoll, LwristYaw, LHand) separately.

```
(defvar arm-position vector (0.9 -0.5 -0.5 -1.4 -0.01 0.8))
```

These definitions will be wrapped in an action designator which represents the input of the manipulation process module. After resolving the designator these values will be forwarded to parameterize the low-level actuators of the Nao robot.

Abstract plan

Now, it's possible to implement abstract control programs. The following code lines demonstrate how to manipulate the Nao on an abstract way. The CRAM plan language is detailed documented in [14] and [15] and provides macros and functions, like def-top-level-cram-function to define abstract plans. The manipulation-test() function represents a high-level plan which executes the tasks open-gripper, move-arm and turn-head.

```
(def-top-level-cram-function manipulation-test ()
        (par (achieve `(open-gripper :left))
             (achieve `(open-gripper :right))
             (achieve `(move-arm :left arm-position))
             (achieve `(turn-head head-position))))
```

The achieve keyword is conduced to execute the actions. The CRAM plan language also provides support for parallel execution.

The following picture presents the output of the Nao manipulation. It shows how the head is rotated to the defined target position. Besides, this picture demonstrates the result after moving the left arm and opening both grippers.

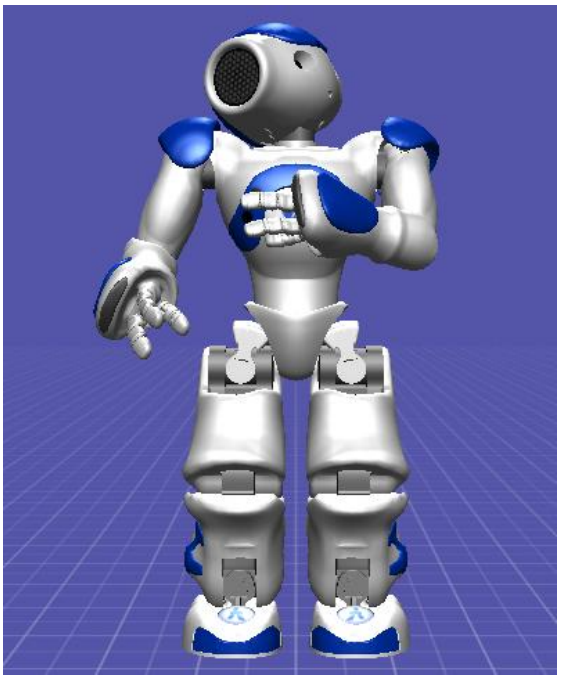


Figure 7: High-level control of the manipulation process module

3.2 Navigation Process Module

This test aims to send abstract commands to the Nao like walking to a specific position. The next code listing initializes a setup.

Meta information

```
(defvar point (make-msg "Point" x 2.0 y 1.5 z 0.8))
(defvar quaternion (make-msg "Quaternion" x 2.2 y 2.1 z 1.0 w 0.5))
(defvar p (make-msg "Pose" position p orientation q))
(defvar h (make-msg "Header" frame_id "base_link"))
(defvar posestamp (make-msg "PoseStamped" header h pose p))
```

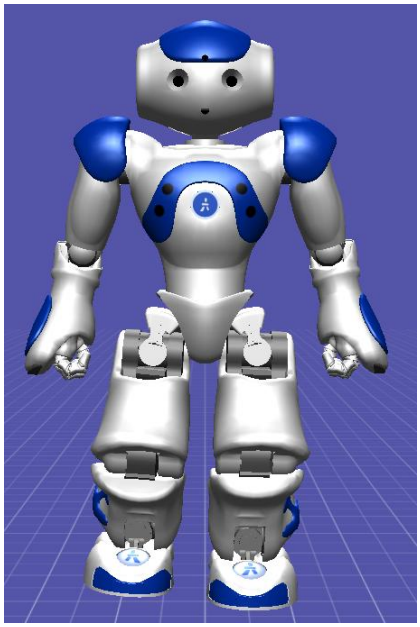
The posestamp represents the target position and contains the location as well as the frame_id. This identifier describes that the target position is relative to the coordination system of the torso. This information is important because the several components are equipped with its own coordination system.

Abstract plan

The following plan demonstrates how to navigate the Nao robot to a special position.

```
(def-top-level-cram-function navigation-test () (achieve `(walk-to posestamp)))
```

The result of the execution is shown in the next figure. It commands the Nao robot to walk to the defined location.



```
[MOVE_BASE_MSGS-MSG:MOVEBASEFEEDBACK
   BASE_POSITION:
     (GEOMETRY_MSGS-MSG:POSESTAMPED
      (:HEADER
       . [STD_MSGS-MSG:HEADER
          SEQ:
            0
          STAMP:
            1.382708949145007d9
          FRAME_ID:
            "/base_link"])
      (:POSE
       . [GEOMETRY_MSGS-MSG:POSE
          POSITION:
            (GEOMETRY_MSGS-MSG:POINT
             (:X . -0.020750809392064622d0)
             (:Y . -4.098065350820906d-5) (:Z . 0.0d0))
          ORIENTATION:
            (GEOMETRY_MSGS-MSG:QUATERNION (:X . 0.0d0)
                                          (:Y . 0.0d0)
                                          (:Z
                                           . 9.137137474409557d-20)
                                          (:W . 1.0d0))])))].
```

Figure 8: Output after navigating the Nao robot

This test client receives the MoveBaseFeedback message from the Action-Server which contains the current location of the Nao robot.

3.3 Speech Process Module

The next test sends in English formulated sentences to the speech process module. The output is displayed by the Choregraphe simulator.

Meta information

```
(defvar sentence ”Hi I am Nao Let’s Rock `n Roll”)
```

This sentence is used to test the speech process module of the Nao robot.

Abstract plan

The following high-level plan commands the Nao robot to translate the defined variable sentence to speech.

```
(def-top-level-cram-function speech-test ()
        (achieve `(speak sentence)))
```

The output of the execution is shown in the next figure. The Choregraphe simulator receives the sentence from the speech module and displays the result.

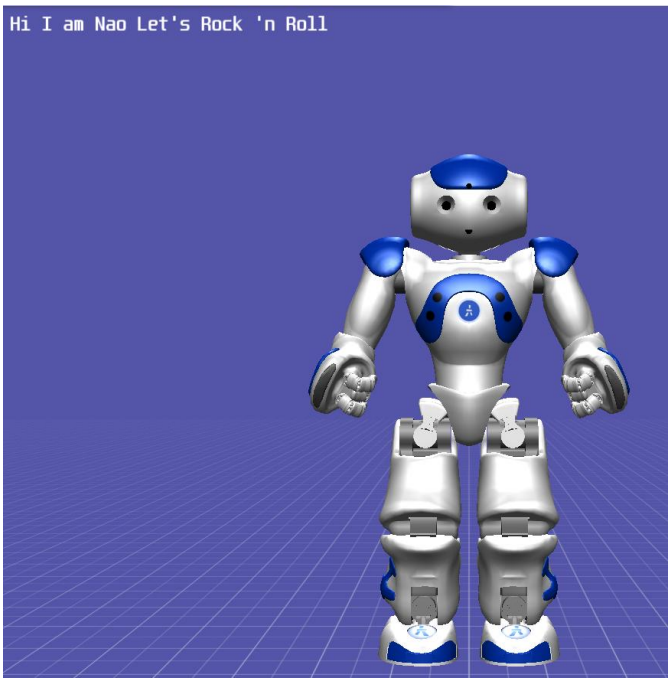


Figure 9: High-level control of the speech process module

4 Evaluation

The processed tests have shown first experiences how to control the Nao robot with high-level plans via process modules. Firstly, they demonstrated how to start the three process modules manipulation, navigation and speech separately in their own thread. Thus, it's possible to access these modules in parallel. Some of the specified actions in section 2 were tested. The experiments investigated the possibility to manipulate the actuators of the Nao as well as to navigate the robot to a specific position. Furthermore, another trial tested the speech process module to translate text to speech. The power of this approach lies definitely on the flexibility of the robot programs. The trials showed the possibility to execute abstract commands which aren´t Nao specific. So, it's potential to execute the same abstract plan on a different robot platform by exchanging the process modules. Through the Choregraphe simulator, the results of the experiments could be displayed in a comfortable way.

5 Outlook

This study developed process modules for the Nao robot to hide the robot specific hardware components. So, it´s possible to define high-level plans in the CRAM plan language. One of the following steps is to implement a mechanism to download abstract plans from RoboEarth. The preconditions are already done to execute such plans on the robot. It should be possible to request the RoboEarth web service for an abstract plan. The process module delegates the request to the KnowRob component which is described in the last project [8]. KnowRob explores the RoboEarth database for an existing plan. After matching the capabilities of the robot against the requirements of the requested plan KnowRob generates also a CRAM plan which should be executed on the Nao platform with the aid of the process modules. Besides, the implementation of a perception interface will provide face detection and speech recognition to extend the functionality of the Nao interface. Finally, some scenarios will show how Nao robots download and perform high-level plans, like grasping an object, from RoboEarth.

6 Conclusion

This paper described the implementation of the process module approach for the Nao robot platform. The design section introduced an interface to manipulate, navigate as well as to access the speech module of the Nao on an abstract way without thinking about how the robot solves these tasks. This project demonstrated how to map high-level plans to low-level commands via the ROS middleware. Furthermore, it showed the communication between the process modules and the Nao-Wrapper via ROS-Nodes. It also described which ROS-Nodes were used from the Nao-Wrapper to forward commands to the low-level control layer of the robot. Besides, some experiments presented how to use the three process modules for the Nao platform by abstract plans. The evaluation of these trials showed that using the process module approach for the robot provides definitely the possibility to execute programs which aren´t Nao-specific. Finally, the outlook described the following next steps.

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. F. Johannßen. Nao in the Cloud. 2013

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

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

11. http://wiki.ros.org/nao

12. Drew McDermott. A Reactive Plan Language. 1991

13. http://www.aldebaran-robotics.com/en/Discover-NAO/Software/choregraphe.html

14. L. Mösenlechner, N. Demmel, M. Beetz. Becoming Action-aware through

Reasoning about Logged Plan Execution Traces. 2010

15. T. Rittweiler. CRAM Design & Implementation of a Reactive Plan Language. 2010