Playing with Naomi

Design and implementation of a human-robot interaction game using the NAO robot platform

Author:
Bas Bootsma
0719080

Supervisor:
Dr. ir. Martijn van Otterlo
Second Reader:
Dr. Ida Sprinkhuizen-Kuyper

August 31, 2012

Bachelor Thesis
Artificial Intelligence
Faculty of Social Sciences

Radboud University Nijmegen

Contents

Chapter 1

Introduction

Robots have been present for over 25 years in professional environments, but are slowly beginning to enter domestic environments [1, 6]. While most of the domestic robots are confined to vacuum cleaning or lawn mowing other more social oriented robots, like the NAO robot, are becoming more common. In a social setting human-robot interaction is an important aspect for successful communication and cooperation between humans and robots. Even though interaction between humans and robots is still limited, research on this topic is being conducted yielding interesting insights into human-robot interaction [5, 2].

Developing robot applications using social robots is complex and provides various challenges. In order for social robots to be successful in the interaction with humans and the environment then need to possess many different skills. Social robots generally reside in environments which have a wide range of different properties and settings (like domestic environments). In order for robots to determine where they are and to where they should go to, skills are needed to build a map of the environment and keep track of their current location. Social robots are designed to communicate and cooperate with humans and thus need skills to facilitate interaction between robots and humans. Furthermore, unlike robots in professional environments, like factories or hospitals, controlled by expert users, robots in other environments are handled by a broad variety of people, who generally have no experience with or knowledge about robots. Thus, having natural interactions facilitates the usage by these people.

Many of these skills are fairly well researched and developed on an individual basis, however incorporating these skills into a single system still is very challenging. In order to get more insights into how individual skills can be incorporated into a single system and to be able to run experiments on the topic of human-robot interaction with the same system, the following goal has been devised:

> Design and implementation of a human-robot interaction game using the NAO robot platform.

Just like robot applications using social robots, games are challenging. Games

generally revolve around complex environments in which many different aspects, like the playing board and the players are present. In order for a robot and a human to successfully play a game with or against each other, the robot needs to possess many different skills to handle all the different aspects of the game. For example, the robot needs skills to perceive the environment, ways to interact with the playing board and the players and keep track of the game. However, when compared to other environments, like professional or domestic environments, games are generally smaller in scale while still requiring the incorporation of many different skills into a single system. Thus, games are a suitable environment to experiment with social robots.

The goal of the human-robot interaction game is for the NAO robot and a human to cooperatively build an object. The NAO robot will gesture towards several pieces which the human has to pick up, since the NAO robot is unable to pick up the pieces itself. When all the pieces have been picked by the human the NAO robot will gesture the human to create an object out of the picked pieces. When the human is done building an object the NAO robot will evaluate the object to see if it matches its expectations.

In order for a successful experiment a suitable experiment setting, as can be seen in figure 1.1, had to be found in which the game is played. In the next section a quick overview of most important aspects of the experiment and experiment setting is discussed.

1.1 Experiment Setting

The environment in which the game is being played by the NAO robot and a human subject is called the experiment setting. In the experiment setting the NAO robot is sitting on one side of the playing field and the subject on the opposite side. The playing field, which is in between the NAO robot and the subject, contains the pieces at the edges and an empty area named the center field. The pieces which are picked by the subject have to be placed on the center field, which is also the area in which the object is created. Using figure 1.2 certain key aspects of the experiment setting will be discussed.

1.1.1 Playing Field

The playing field (denoted 1) is the area which encompasses both the pieces and the center field. After having tried several surfaces covering the playing field, e.g. a plain table or a white surface, it became clear that a matt black surface returned the best results by the perception. This probably has to do with the fact that the color distortion at the edges of the pieces, due to the shade or the reflection of the surface, is minimized when using a mat black surface.

1.1.2 Center Field

The center field is the open area (denoted 3) on the playing field between the pieces. The picked pieces are to be placed on the center field as well as the object which is created from the pieces.

1.1.3 Position of the NAO Robot

It is important that the NAO robot is positioned in a sitting position perpendicular to and facing towards the playing field, as can be seen in figure 1.1. The feet of the NAO robot are positioned at the edges of the playing field (denoted 2), which is done for two reasons. The first reason is the fact that the NAO robot now sits at a fixed known position which is necessary because the gazing and pointing at the pieces is relative to the center field. This means that if the NAO robot is not positioned properly it will not gaze and point to the correct pieces. The second reason is the fact that it simply saves time, since it prevents having to initialize the locations of the pieces every time you want to run the experiment.

1.1.4 Pieces

The pieces are an important aspect of the experiment setting for several reasons. Not only do they have to be handled by the subject, and thus not look too artificial or be too fragile, more importantly they have to be recognized by the visual perception system. In order for the pieces to be recognized by the visual perception system they would need to have a decent size. This means that the pieces can not be too small, otherwise the visual perception system would not be able to detect the pieces, nor too large or not enough pieces would fit in the sight of the NAO robot (the view angle is limited) to make the game interesting. Furthermore, it is important that the colors are distinct enough from one another to prevent mismatches by visual perception system, and they have to be all distinct enough from the surface. In general, the more flat the pieces are the better they are recognized by the visual perception system, since there is less color distortion around the edges caused by shadows and the birds eye view from the NAO robot. Therefore the almost flat, brightly colored pieces were used as can be seen in figure 1.2.

1.1.5 Subject

The subject of the experiment will sit at the opposite side of the playing field. He or she will have to be close enough to the NAO robot in order to press its head to start the experiment.

In the following chapters first the hardware and software that were used are discussed. In subsequent chapters an overview of the system and the interaction between the individual components is discussed followed by the technical implementation of the individual components. Afterwards a detailed overview of

the experiment in action is discussed and the final chapter contains conclusions and future work, with many possible suggestions for improvements.

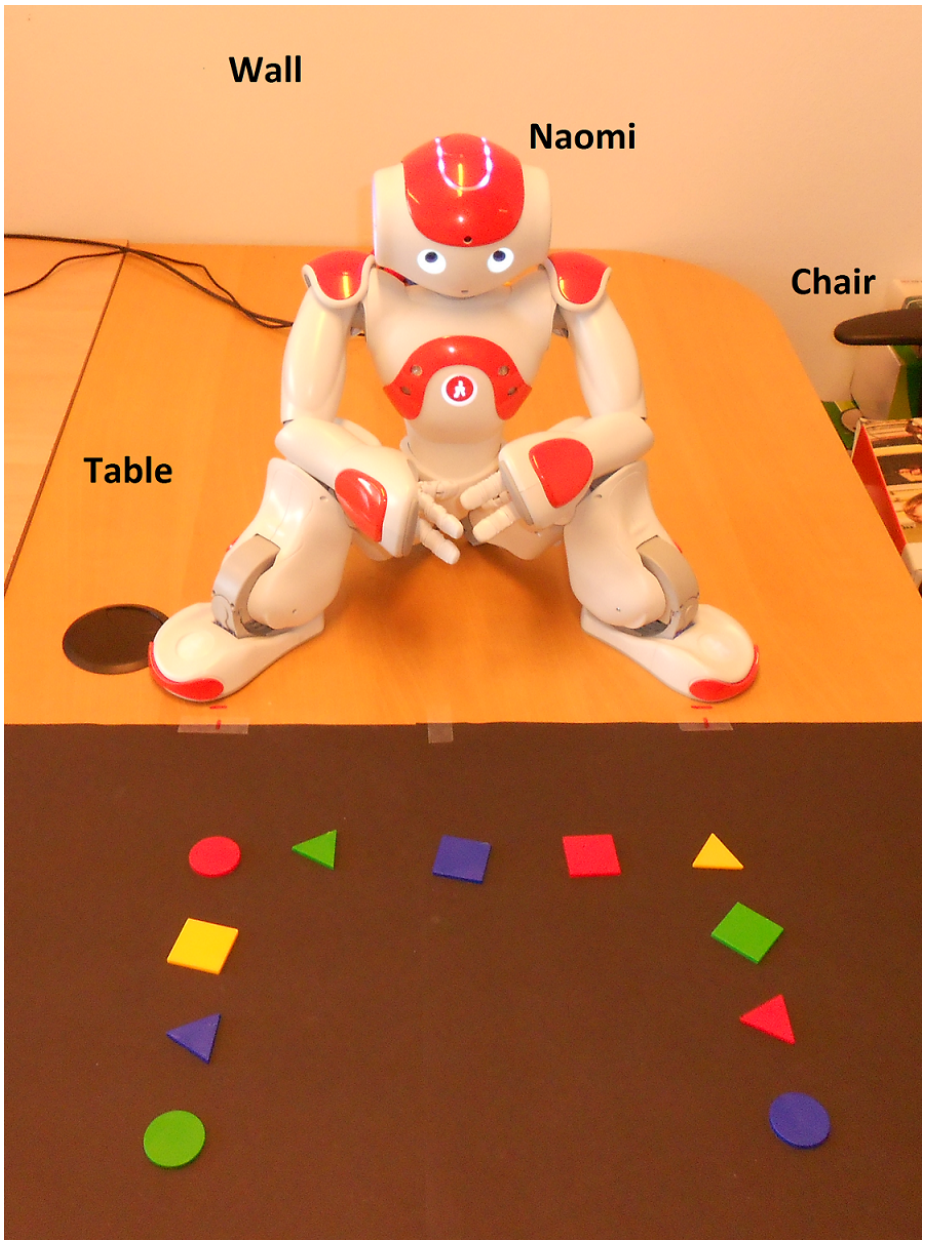

Figure 1.1: Birds's-eye view of the experiment setting.

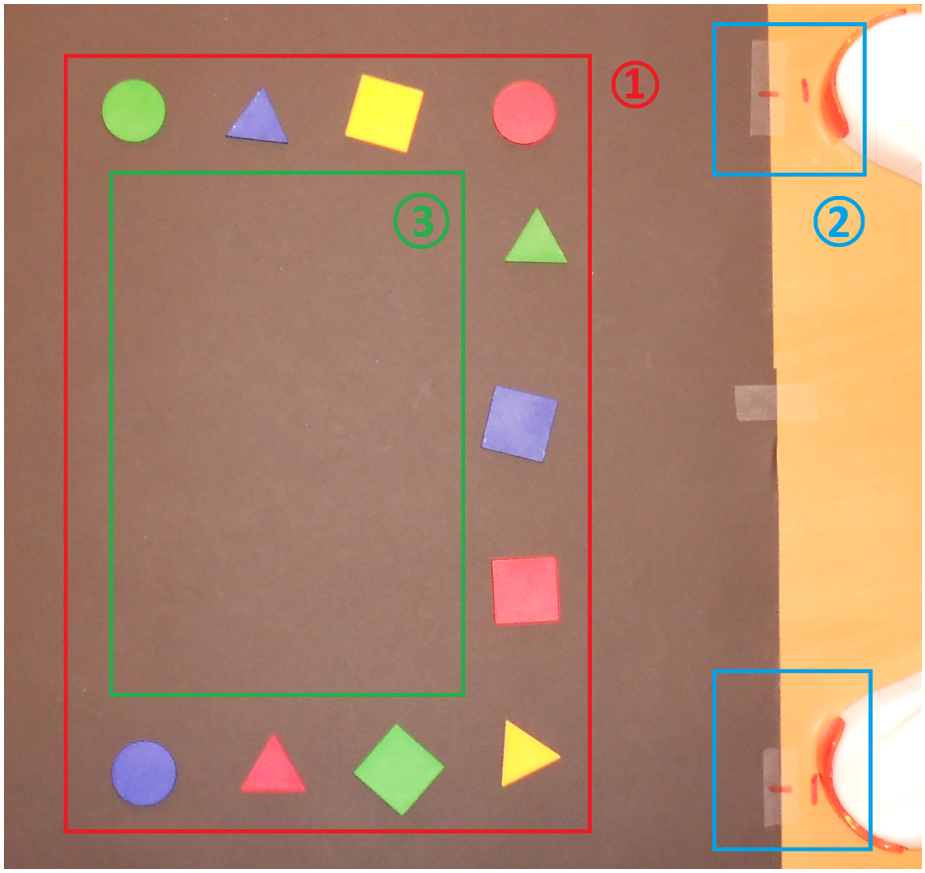

Figure 1.2: Detailed top down view of the experiment setting: 1. playing field, 2. position of the NAO robot and 3. center field.

Chapter 2

Hardware / Software

In the following sections an overview of the different hardware and software components is given.

2.1 Hardware

The hardware components consist of the NAO robot and a computer, which is used to run the software on. For specific details on the requirements of the computer see appendix A.

2.1.1 NAO Robot

The NAO Robot is a humanoid robot created by Aldebaran Robotics [8]. In total the NAO robot has 25 degrees of freedom of which the tension can be individually controlled via an electric motor and an actuator. It has various sensors, ranging from tactile sensors in its feet to two cameras and several microphones in its head. This is all driven by an embedded computer with Linux installed on it through which all the hardware components can be accessed. In order to more easily control the hardware components the company also provides an application programming interface (API), called the NAOqi, which is the basis of all the behaviors the NAO robot performs.

2.2 Software

The software components consist of two environments: the NAO robot platform, consisting of NAOqi, Choregraphe, NAOsim and monitor, and the Robot Operating System (ROS), a middleware system designed to be used by many different robots. Both environments will be discussed in the following sections.

2.2.1 NAO Robot Platform

The NAO robot platform [9] not only consists of the NAO robot, but also of an API, named NAOqi, and several tools, like Choregraphe, NAOsim and Monitor, which will be discussed in the following sections.

NAOqi

NAOqi is an API which is being used for all the versions of the NAO robot. It is a cross-platform library which contains high-level methods to communicate with the different hardware components of the NAO robot. NAOqi allows developers to more easily create robotic applications without having to understand to control hardware components directly. For example, there are methods to control each joint individually or multiple joints at once which allow you to set several goal joint angles after which NAOqi will automatically interpolate the correct movement. Several high-level modules are also available which have been built by the Aldebaran Robotics team already. These include an omni-directional walking module and a speech synthesis module.

It is important to know that NAOqi is designed as a library to be used for NAO robots only, which allows for homogenous communication between all the hardware components and modules. This means it does not have to deal with other kinds of robots or hardware components (like the Kinect) which makes the coupling between different hardware components and modules much tighter. Therefore it is possible to quickly develop robot applications for the NAO robot using only NAOqi. However, with only NAOqi it is impossible to built the full experiment setting, which becomes clear in the following chapter, and therefore another library, named the Robot Operating System (ROS), is used in conjunction.

Choregraphe

One of the tools provided by Aldebaran Robotics to aid developers in creating robot applications is Choregraphe. It provides several useful modules, like a module to create poses, a module which gives complete visualization of the NAO robot by providing real-time feedback of the current joint states and a graphical programming module, as can be seen in figure 2.1. The graphical programming module allows users to create robot applications by inserting boxes via an drag-and-drop interface. These boxes can be connected to each other by connecting the output(s) of one box to the input(s) of another box. With a lot of preset boxes available users can quickly create quite robust and advanced applications for the NAO robot. Since most of the boxes contain pure Python code it is even possible to edit the boxes to create even more advanced applications.

NAOsim

The NAOsim is a simulator with a physics engine designed for the NAO robot. It contains a detailed mesh model of the NAO robot and several prebuilt en-

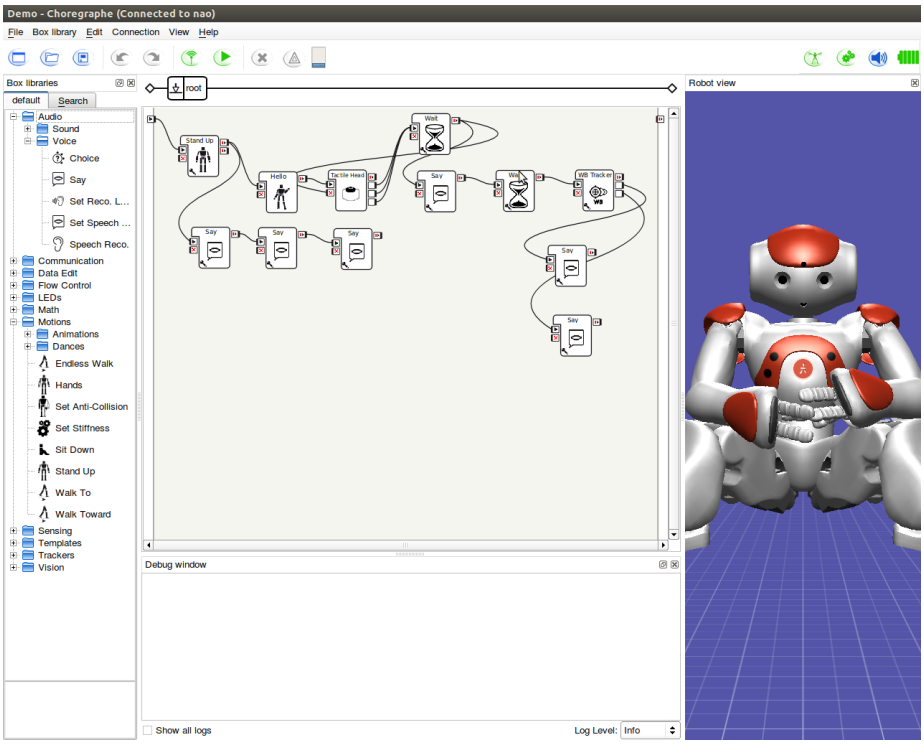

Figure 2.1: Choregraphe with the graphical programming module.

vironments with a number of objects. With the NAOsim it is possible to test applications developed with Choregraphe or NAOqi. However, only a subset of the sensors and actuators is available which means certain applications cannot be tested within the NAOsim.

Monitor

Monitor is a tool with two separate modules, namely a *camera* and a *memory* module. With the camera module it is possible to display the video stream from the video camera of the NAO robot and perform several actions with regard to the video stream. Examples of actions are, recording the video stream, adjusting brightness/contrast/saturation/etc. levels, or displaying detected faces in the video stream. With the memory module it is possible to display data from the robot's sensors in a user friendly environment.

2.2.2 Robot Operating System

The Robot Operating System [4] is a middleware system which is designed to help developers with the development of robot applications. ROS provides libraries and tools such that individual components can communicate with each other. In this sense NAOqi and ROS have a similar goal, namely making the

development of robotic applications more easily. There is however an important distinction; ROS focuses on different components from *different* robots to communicate with each other, while NAOqi only allows different components from the *same* robot, namely the NAO robot, to communicate with each other. Therefore it is necessary for ROS to have a broad perspective on robots (and components in general).

This broad perspective from ROS can be seen in how it is structured into different components of which important components, like nodes, communication (topics and services) and the parameter server, will be discussed in the following sections.

Nodes

Each process which performs any sort of computation is seen as a node. Nodes are the most basic concept and provide ROS with a great level of modularity. The robot application is generally built from multiple nodes, for example: a node to control the wheels, a node to provide localization and a node to provide path planning. The benefit of having such a modular system is that once a node has been developed it can be used by other robots as well.

Communication

ROS provides two ways for nodes to communicate with each other, namely via topics and via services.

Topics Topics are an asynchronous many-to-many way of communication. Each node has the ability to publish to and subscribe to as many topics as needed. Each publisher is able to publish messages to a topic and each subscriber (if subscribed to the same topic) is able to retrieve them. A message is either from a predefined basic type, like a string or an integer or from a composite type, which is built from basic types. To give an example of how it could work in practice: suppose there are two nodes, one node which reads the video stream from a robot and another node which performs face recognition. The video stream reading node publishes images from the camera to a topic and the face recognition node subscribes to the same topic to be able to retrieve the video stream. Every time the video stream publishing node publishes a new camera image the face recognition node retrieves the data and is able to perform face recognition on the camera image. The results from the face recognition node can then be published to a different topic which other nodes can use.

Services Services are a synchronous one-to-one way of communication. It allows nodes to exchange information with each other via a request-reply protocol. For example, services can be used when a node A needs a node B to perform computations on an image, yet node A cannot continue before it has received the results of the computations. In such a case node B can be set up as a service provider to which node A, as a client, can connect and make a

request containing the image data. Node B will then respond with the results of the computations after which node A can continue. It is even possible to set up a persistent connection between clients and the service provider allowing for increased performance.

Parameter Server

Since ROS is set up as a modular system it is not uncommon that nodes need to be configured before they are able to work properly. Therefore ROS has already implemented a way to handle this, namely via the parameter server. The parameter server is basically a large dictionary through which nodes are able to store and retrieve parameters at runtime. Nodes which make use of the parameter server can be easily configured by other nodes.

Organization

In order to keep track of all the software, e.g. nodes, messages or third-party libraries, ROS provides a way to organize the software. Software which shares enough common characteristics can be packed together in *packages*. Packages themselves can be packed together in *stacks*. The organization of software in packages and stacks make it easier to reuse and share the software. Furthermore both packages and stacks can define dependencies on other packages and stacks which makes it possible for ROS to automatically install the dependencies, if available. More information on the different stacks that were used can be found in appendix A.

Chapter 3

System

The system is a collection of components, like perception, gestures and the world model, which are used in order to run the experiment. A high-level overview of the system and the interaction between the individual components is given in figure 3.1. In short the system works as follows: the system uses perception to perceive the environment which is used to update the world model. With the updated world model the system then is able to decide whether the goal has to be updated. Based on the goal, actions are necessary to reach the goal. These actions translate into intentions which are conveyed by the system to the subject using gestures. The subject manipulates the environment which allows the perception to perceive the changes, which closes the loop. In the following sections all the individual components are discussed from an AI point-of-view, while the next chapter focuses on the technical aspects.

3.1 Perception

Perception is used by the system to perceive the environment, i.e. what the world "looks" like. For perception only vision is used, however it is possible to extend perception by making use of various other sensors, like touch or sonar sensors, available on the NAO robot. In order for the world model to update itself it expects a list containing information on the location of the pieces. In the current system pieces are detected as blobs, with each blob containing a bounding rectangle of the piece it is surrounding, using blob detection. Thus, perception passes a list of blobs onto the world model.

In order for the blob detection to work it was necessary to get the video stream from the NAO robot. Since the NAO robot has two cameras, namely a top and bottom camera, it had to be decided which camera had to be used. The top camera was chosen since in the bottom camera the NAO robots arms would block the bottom part of the playing field. There were however a number of problems with the video stream and the blob detection which will be discussed in the following sections.

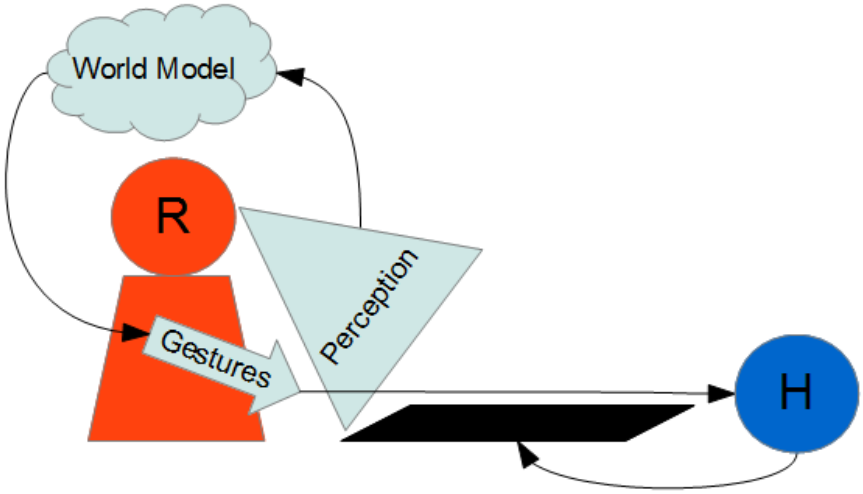

Figure 3.1: High-level overview of the system and the interaction between individual components.

3.1.1 Video Camera Angles

When looking down the video stream would not only consist of the playing field, but also of large areas at the east, north and west edges. This meant that the blob detection would not only detect the pieces in the playing field, but sometimes other non-relevant blobs. While in general it would not be much of a problem since the position of pieces are manually marked during the setup, but if the blob detection detects too many blobs it would slow down the system and on occasion even crash the system. Since the video stream is only used for perceiving the game field, and not for example for face detection or gaze tracking, a practical solution was to crop the image to the boundaries of the playing field.

3.1.2 Noise of the Video Stream

The video stream itself contains a fair amount of noise which primarily has to do with the quality of the camera. The blob detection was not able to handle the noisy video stream and would often only partially detected pieces or return multiple blobs for the same piece. In order to solve the problem an image mean shift filter was used, which homogenizes the colors in the video stream and also reducing the noise in the video stream.

3.1.3 Color Distortion

With the NAO robot in a sitting position and looking down at the pieces the camera provides images in a birds-eye view. This causes the colors of the pieces

getting more distorted the further they are physically away from the NAO robot. This means that a red piece at the bottom left would have a significantly different color than when it would have been placed at the top right. The blob detection would therefore sometimes not be able to detect the same piece at different locations in the playing field. In order to solve this problem the lower- and upper bound of each class were extended by a certain offset. This was possible because the individual colors of the pieces were distinct enough in order to prevent the overlapping of colors.

3.1.4 Perspective Distortion

Another problem which occurs from the birds-eye view is perspective distortion. Even though the pieces are placed around the edges of an imaginary rectangle, as can be seen in figure 1.2, in the video stream the pieces appear to be placed around the edges of a trapezoid. This causes problems for the gesturing component and therefor the solution will be discussed in section 4.2.

With all the problems solved the system had a very robust way of getting the pieces detected as blobs using blob detection.

3.2 Gestures

Gestures are used by the system to convey the intentions from the NAO robot to the subject. The goal of the gestures is to interact with the subject such that the subject will manipulate the environment, preferably in a way that matches the intentions. To aid in designing gestures for successful human-robot interaction we looked at existing research in this field. Research by C. Sidner et al. [11] suggests that humans respond to changes in gaze of the robot by changing their own gaze. Furthermore robots which gesture and speak are able to capture a human's attention more often. Other research by M. Salem et al. [10] reveal that successful interaction between robots and humans is facilitated if hand and arm gestures are displayed along with speech, even if they do not semantically match.

Even though the NAO robot has far less facial expressive power than the robot Kismet, research by S. Breazeal on the robot Kismet [2] does provide some interesting insights, namely to offer engaging and compelling interaction with humans it is important robots have to do the right thing at the right time in the right manner. An example which perfectly illustrates these insights and has been a large inspiration is a promotional video named named Developing Robots that can Teach Humans [7]. In the video a robot directed sorting task is shown in which a human has to put Duplo-like blocks into one of two boxes based on the instructions of the robot. Humans appear to do better at the sorting task when both speech and corresponding gazing behavior is combined when compared to just speech.

Using the existing results it was decided to let each gesture consist of both a verbal and non-verbal components. To make gesture look as natural as possible it

was decided to execute the verbal component first, directly followed by the non-verbal components. The system uses different kind of gestures, namely general gestures and pointing gestures. General gestures consist of speech, gazing and supportive arm movements, while pointing gestures consist of speech, gazing and pointing. Pointing gestures are used during the stage in which the subject has to collect the pieces and general gestures are used in all other stages.

The verbal component of each gesture was easy to execute since NAOqi made it possible to enter an English sentence which would automatically be converted into a spoken message. In order to make the executing of the non-verbal components easier, they are combined into a pose. Each pose consists of several motor joints angles which have to be executed in a three dimensional space. Using NAOqi it is possible to define the final motor joint angles and let NAOqi automatically interpolate the trajectory from the current pose. For each gesture the NAO robot has been manually molded into a natural looking pose which is stored and to be recalled any time the gesture is executed. This process of manually molding a robot into a pose can be seen as an basic example of imitation learning. Most forms of imitation learning use an algorithm which needs to be trained after which it is able to approximate the correct pose. However, in our case, the exact motor joint angels are stored and can be recalled at any time without the need for training.

3.3 World Model

In order for the system to be able to decide when, how and what to do it needs to have model of the world. The world model incorporates both a priori knowledge and knowledge received by perception into a single model which is used by the system to determine its beliefs about the world. Using these beliefs the system is able to update its goal and decide what actions needs to be performed in order to reach the goal. In order to form proper beliefs about the world, the world model needs to consist of several components:

- **State**: The experiment consists of several stages in which each stage has specific actions to reach the next state;
- **Blobs**: The latest received set of blobs from perception is stored and used to update the world model;
- **Pieces**: In the beginning it is assumed all the pieces are available and at their specified location, which is known through a priori knowledge. Each piece has one or more flags indicating whether it has been detected, whether it should be picked and whether it has either successfully or unsuccessfully been picked. Using the blobs the world model is able to update the correct flags on each piece.

Chapter 4

Individual Components

In the previous chapter a high-level overview of the system and the individual components is given. In this chapter the technical aspects of the individual components are discussed.

4.1 Perception

In order to perceive the environment perception only uses blob detection. The goal of blob detection in perception is to find clusters of pixels of the *same* color. In this case that means that each color defined as a set of thresholds, one for each component of the color. For example, if the colors are in the YUV color space the blob detection will have for each class to be detected a set of thresholds which consist of $\{(\text{minY}, \text{maxY}), (\text{minU}, \text{maxU}), (\text{minV}, \text{maxV})\}$. After each pixel has been visited and possibly been assigned a class, connected pixels are assigned the same cluster. Furthermore if clusters of the same class which are within a certain defined range of each other are found, it is likely these clusters actually cluster the same object. Thus, such clusters are merged together to one cluster. After all the clusters are found the blob detection will calculate a rectangular bounding box around the cluster.

Listing 4.1: Example blob detection configuration file

```
[colors]
(255, 255, 0) 0.000000 1 Yellow
(0, 255, 0) 0.000000 1 Green
(0, 0, 255) 0.000000 1 Blue
(255, 0, 0) 0.000000 1 Red

[thresholds]
(189:242, 21:71, 116:166)
(69:141, 73:136, 63:111)
(10:98, 162:216, 96:144)
(67:167, 69:98, 155:209)
```

In the current system blob detection has been implemented using a blob detection algorithm by Color Machine Vision Project [3], which has been made available to ROS. The algorithm uses a configuration file, as seen in listing 4.1, containing the information for the classes of blobs which have to be detected. The configuration file consists of two sections, namely *colors* and *thresholds*, in which an individual class consists of both a line from the section colors and a corresponding line from the section thresholds. For example the class *Yellow* consists of *(255, 255, 0) 0.000000 1 Yellow* and *(189:242, 21:71, 116:166)*. Each line from the section colors consists of the following elements:

- **Color**: Returned via the Blob message in ROS which can be used to determine to which class the blob belongs to. The color is given in the RGB color space;
- **Merge threshold**: Used by the algorithm to determine whether regions have to be merged;
- **Expected blobs**: The amount of expected blobs of this class;
- **Name**: Name of the class.

A line from the section thresholds corresponds to the minimum and maximum thresholds for a blob in the YUV color space, specified as follows:
$(minY : maxY,\ minU : maxU,\ minV : maxV)$.

Using a configuration file the blob detection algorithm is able to extract blobs from a given image using several stages, which are briefly discussed in the following sections.

4.1.1 Thresholding

At the first stage for each pixel in the image is determined to which class(es) it belongs. It basically determines whether the Y, U and V values are in between their respective lower and upper thresholds. Since it is quite expensive to perform this real time the algorithm builds a boolean valued decomposition of the thresholds from the configuration file. This allows the algorithm to simultaneously determine to which class(es) a pixel belongs which increases the performance.

4.1.2 Connecting Regions

At first a run-length encoded (RLE) version of the classified image is created. This means that horizontally connected pixels are merged together in a single run. Using a tree-based union find, overlapping horizontal runs of the same class are connected. Each connected run is assigned the class of the upper leftmost run.

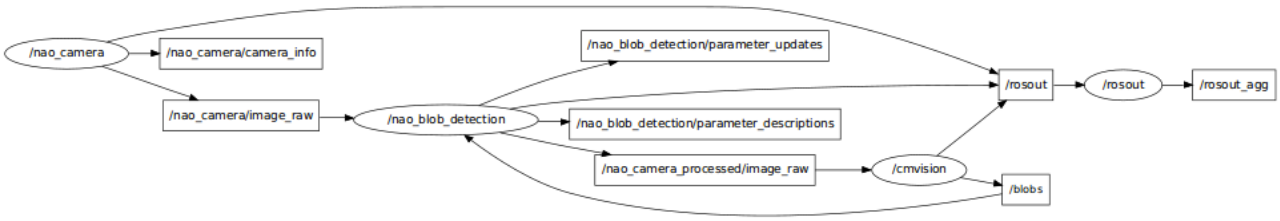

Figure 4.1: Nodes and topics for blob detection in ROS.

4.1.3 Extracting Region Information

During this stage region information, like the bounding box, centroid, etc., are extracted incrementally from the merged RLE map.

4.1.4 Density-based Region Merging

At the final stage regions which are similar and close to each other are merged if their combined density is above a certain *merge threshold* which is defined in the configuration file.

In order to get an idea of how the blob detection has been implemented in ROS, an overview of the different nodes, as seen in figure 4.1, is discussed:

- **nao_camera**: The purpose of the node nao_camera is to publish a stream of images from the video camera of the NAO robot. Using NAOqi images are captured from the video camera of the NAO robot at a certain frequency. These images are published to the topic */nao_camera/image_raw*, while details of the video camera are published to the topic */nao_camera/camera_info*.
- **nao_blob_detection**: The purpose of the node nao_blob_detection is to process the images to make them more suited for the blob detection algorithm. The node nao_blob_detection subscribes to the topic */nao_camera/image_raw* and processes the images, namely cropping the image and applying a mean shift filter. Afterwards the processed images are published to the topic */nao_camera_processed/image_raw*.
- **cmvision**: The purpose of the node cmvision is to apply the blob detection algorithm to the images and publish the retrieved blobs. The node cmvision subscribes to the topic */nao_camera_processed/image_raw* and retrieves the blobs using the blob detection algorithm. These blobs are published to the topic */blobs*.

4.2 Gestures

As discussed in the previous chapter gestures are used by the system to convey the intentions from the NAO robot to the subject. In this section only pointing

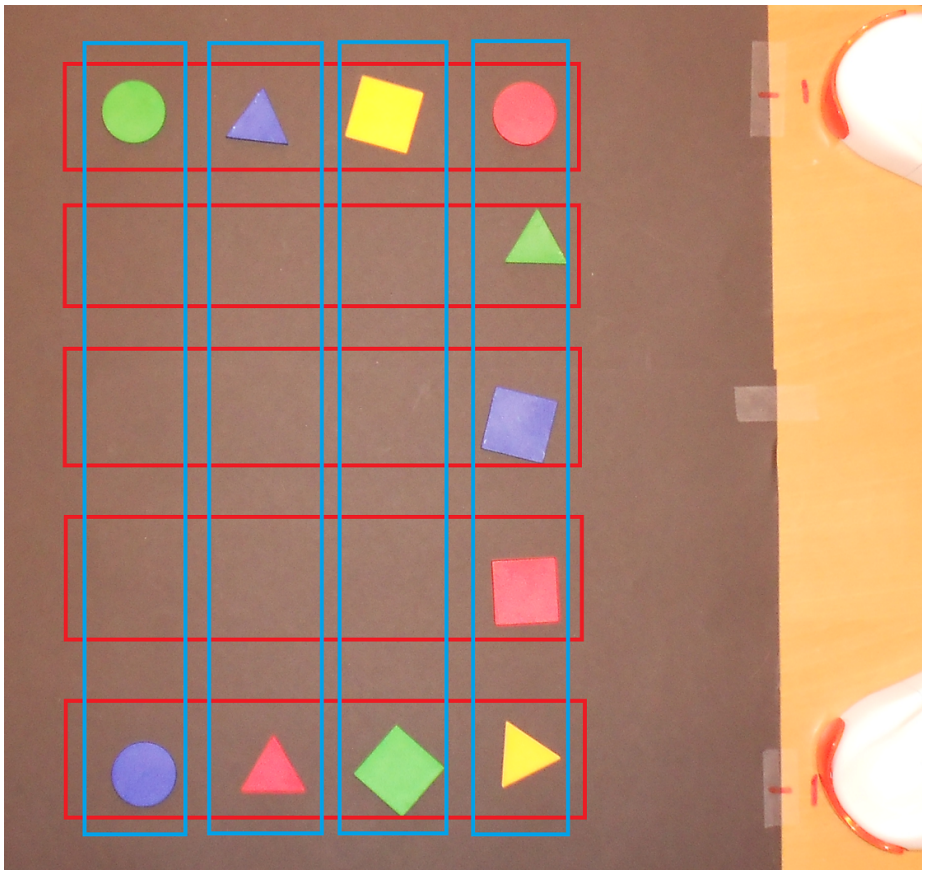

Figure 4.2: Playing field divided into vertical (red) and horizontal (blue) lanes, based on the point of view of the NAO robot.

gestures, which consist of speech, gazing and pointing, are discussed. In the ideal case it is possible for the NAO robot to point to any piece on the playing field. However, implementing such a solution would have taken up more time than there was available, thus a simple workaround was implemented. The playing field is divided, from the NAO robots point-of-view, into several vertical (red) lanes and horizontal (blue) lanes as can be seen in figure 4.2. For each vertical (red) lane the NAO robot has been molded into a pose in which it looks like the NAO robot is gesturing towards the piece(s) in the lane. The pose forms the non-verbal component of the gesture and the verbal component of the gesture is created when needed based on the information, like shape and color, in the world model. Thus, for each lane available a different gesture was created.

With a pose for each lane available, how is determined which pose has to be executed? As can be seen in figure 4.2 the playing is divided into several vertical (red) lanes and horizontal (blue) lanes which form a grid together. Based on the original position of the piece it is determined in which cell of the grid the piece belongs to. However, the original position of the pieces is determined based on the video stream which is subjected to perspective distortion. This means that

pieces which are physically further away from the NAO robot appear to be in a different vertical (red) lane. In order to solve the problem any piece which is not in the bottom horizontal (blue) lane will be placed in either the far left or far right vertical (red) lane depending whether they located in the left or right half of the video stream. Algorithm 1 shows pseudo code to determine to which lane the NAO robot has to gaze and point to.

Algorithm 1 Determining NAO robots gaze and point to lane.

```
function DETERMINELANETOPOINTTO(piece)
    Upon dividing the image in a grid, based on the number of pieces horizontally and vertically, determine to which cell the piece belongs to.
    cell ← INCELL(piece.X, piece.Y)
    if cell.Y > 0 then
        if cell.X lies within the first half of the image then
            LaneTo ← Bottom left cell x-coordinate
        else
            LaneTo ← Bottom right cell x-coordinate
        end if
    else
        LaneTo ← Cell x-coordinate
    end if
    return LaneTo
end function
```

4.3 World Model

The world model is used to keep track of the world and the state of the experiment. Using a priori knowledge, namely the original position of the pieces, and the perception knowledge the world model is able to update its view of the world. Each time the world model receives perception knowledge it updates the detected-state of the pieces as can be seen in algorithm 2.

Algorithm 2 Update pieces detected state using blobs received from perception.

```
function DETECTPIECES(pieces, blobs)
    for all piece ∈ pieces do
        isDetected ← False
        for all blob ∈ blobs do
            rectangle ← Create rectangle from the data in the blob
            if piece intersects with rectangle then
                isDetected ← True
            end if
        end for
        Use isDetected to set piece detected state
    end for
    return pieces
end function
```

4.4 Graphical User Interfaces

To aid in setting up and running the experiment, two interfaces were built which will be discussed in the following sections.

4.4.1 Setup

The setup GUI, as can be seen in figure 4.3, allows certain components, like the cropping of the video stream, the blob detection algorithm and the piece marking, to be set up.

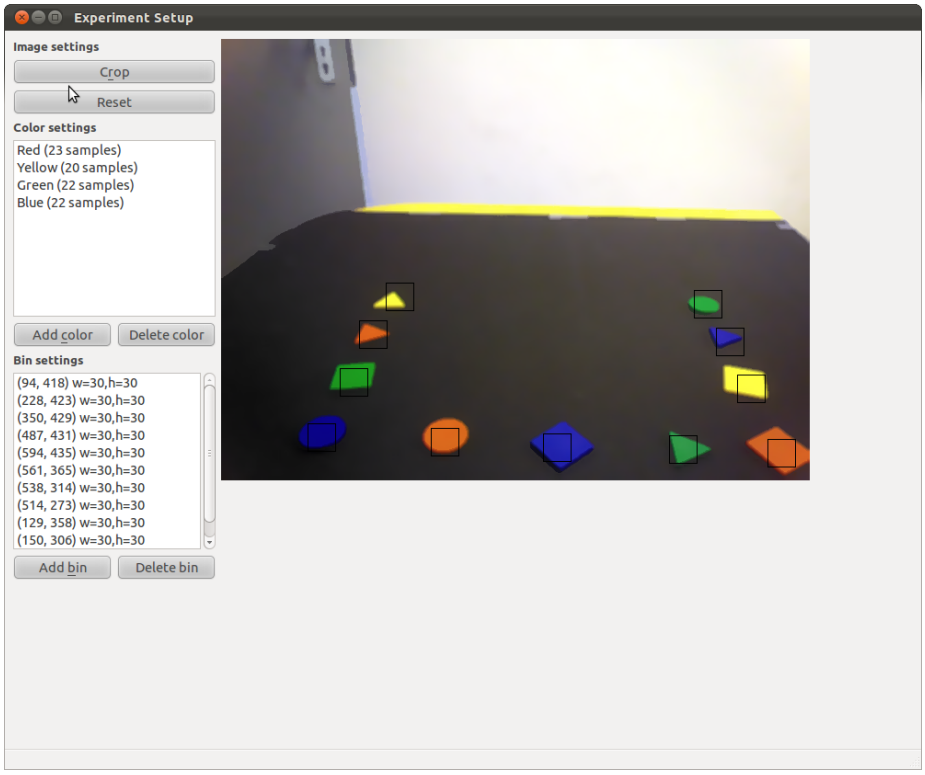

Figure 4.3: Screenshot of the setup GUI.

Video Stream Cropping

As was discussed in section 3.1.1 the video stream would record more than just the playing field. This causes problems for the blob detection of perception, i.e. it would detect unnecessary blobs. Even though the blob detection is able to detect many blobs in real time, having a larger image would could cause it to detect so many blobs that would slow down the system and on occasion even crash the system. Therefore it is possible to use the setup GUI to draw a bounding rectangle to which the video stream is cropped. For the experiment setup the bounding rectangle is drawn around the playing field.

Blob Detection Algorithm Colors

The blob detection algorithm by the Color Machine Vision Project uses a configuration file, as can be seen in listing 4.1, to determine which blobs to detect. Since it is time consuming to create a configuration file manually, the setup GUI has a section with which this can be done much faster. In order to create a new class for the configuration file the experimenter has to add a new color, fill out the basic information (all the elements from the section colors) and click at least once in the video stream on the desired color. Each time the experimenter clicks in the video stream the RGB value at the location of the mouse is saved in a list. When the experimenter is done adding a new color the setup GUI using a algorithm, as seen in algorithm 3, to determine the thresholds based on the list with RGB values. After the thresholds have been determined the setup GUI applies a predefined offset to each component of the thresholds. This increases the range of the color the blob detection has to detect making the blob detection more robust, negating several external effects, like lighting or noise in the video stream. For example, if the algorithm to determine the thresholds returns the following threshold values: $yuvMin = (59, 36, 127)$ $yuvMax = (78, 52, 146)$, after applying predefined offset of for example 10 the thresholds values are: $yuvMin = (49, 26, 117)$ $yuvMax = (88, 62, 156)$. Afterwards the new class is saved to the configuration file.

Algorithm 3 Determining blob detection configuration thresholds.

function DETERMINETHRESHOLDS($colors$)
  $minR, minG, minB \leftarrow 255$
  $maxR, maxG, maxB \leftarrow 255$
  for all $color \in colors$ **do**
    $r, g, b \leftarrow color$

    $minR \leftarrow \min(minR, r)$
    $maxR \leftarrow \max(maxR, r)$
    $minG \leftarrow \min(minG, g)$
    $maxG \leftarrow \max(maxG, g)$
    $minB \leftarrow \min(minB, b)$
    $maxB \leftarrow \max(maxB, b)$
  end for
  $yuvMin \leftarrow$ TOYUV($minR, minG, minB$)
  $yuvMax \leftarrow$ TOYUV($maxR, maxG, maxB$)
  return $yuvMin, yuvMax$
end function

Piece Marking

The world model uses the original position of the pieces to determine whether pieces have been picked. With the setup GUI it is possible to mark all the pieces, which consists of a small rectangle (which is used by the world model to detect whether pieces and blobs intersect).

4.4.2 Monitoring

A monitoring GUI was created, as can be seen in figure 4.4, which makes it possible to run the experiment and allows the experimenter to keep track of the experiment.

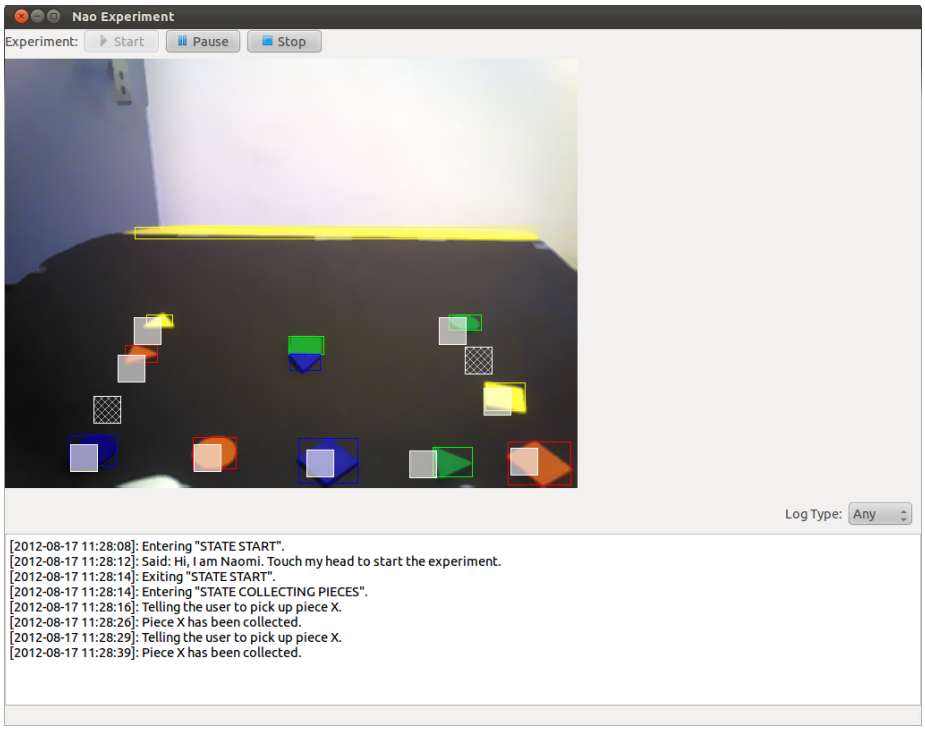

Figure 4.4: Screenshot of the monitoring GUI.

Video Stream

The video stream module display in real time the video stream from the NAO robot. It uses the world model display information about the original position of the pieces and the current state of each piece. In the video stream module the position of each pieces is made visible using rectangles and the filling of the rectangle determines the state of the piece. Each filling has a different meaning, as can be seen in figure 4.4, namely:

- **No filling**: The piece is not detected which in most cases is caused when the NAO robot is not looking straight down, so either when it is looking towards the user or when it is gazing.
- **Full filling**: The piece is detected.
- **Dense diagonal pattern**: The piece which should currently being picked by the subject.

- **Sparse diagonal pattern**: A piece which has been taken successfully by the subject.

Log

The log module displays a stream of messages which have been logged by the experiment. It allows the experimenter to view the current state of the experiment, which actions have been performed by the NAO robot and what instructions have been given by the NAO robot to the subject.

Chapter 5

How "The Experiment" Works

In the previous chapters an overview is given of the individual components of the system. In this chapter a in depth overview of the experiment and the stages of the experiment, namely *instructions*, *collecting the pieces* and *evaluating the object*, is discussed.

5.1 Stages

5.1.1 Instructions

During the instructions stage the NAO robot will instruct the subject on how the experiment works and what is expected from the subject.

5.1.2 Collecting the Pieces

After the instructions stage the NAO robot will randomly choose N pieces ($0 < N < \#pieces$) from the list of known pieces. The NAO robot will proceed to gesture the subject to pick one of the remaining pieces. The verbal component of the gesture is structured as follows: "Could you pick the *color shape*?", which for example results into: "Could you pick the red square?". The subject is given several seconds to pick up the piece and place it in the center field. When the subject picks up the piece the system detects that the piece has been taken and the NAO robot will respond positively saying: "Well done.". If the subject fails to pick up the piece within the given time, it will respond negatively saying: "You did not pick up the piece." Afterwards it will instruct the subject to pick the same piece again. This process continues until all the chosen pieces have been picked and placed in the center field.

Object Creation

Once all the pieces have been picked and placed in the center field the NAO robot will instruct the subject to create an object of the pieces by saying: "Make something you think I would like out of the pieces you just picked." The subject is given predefined amount of time to create an object after which the NAO robot will continue to evaluate the object.

5.1.3 Evaluating the Object

The NAO robot evaluates the created the object and returns an absolute evaluation at first and after the user is given some time to rearrange the pieces, based on the absolute evaluation, it will return a relative evaluation.

Absolute Evaluation

Based on the created object the NAO robot will return an absolute evaluation. In the current implementation the NAO robot does not process the created object at all and will randomly return one of the following evaluations: "It looks interesting", "I don't know it yet" and "I don't like it". After the given evaluation it will instruct the subject to try to make it (even) better by saying: "I think you can do better".

Relative Evaluation

After a predefined amount of time the NAO robot will return a relative evaluation based on the previous and current object. In the current evaluation it will simply return: "It looks better."

5.2 Experiment in Action

In this section an example of the experiment is visualized using several images.

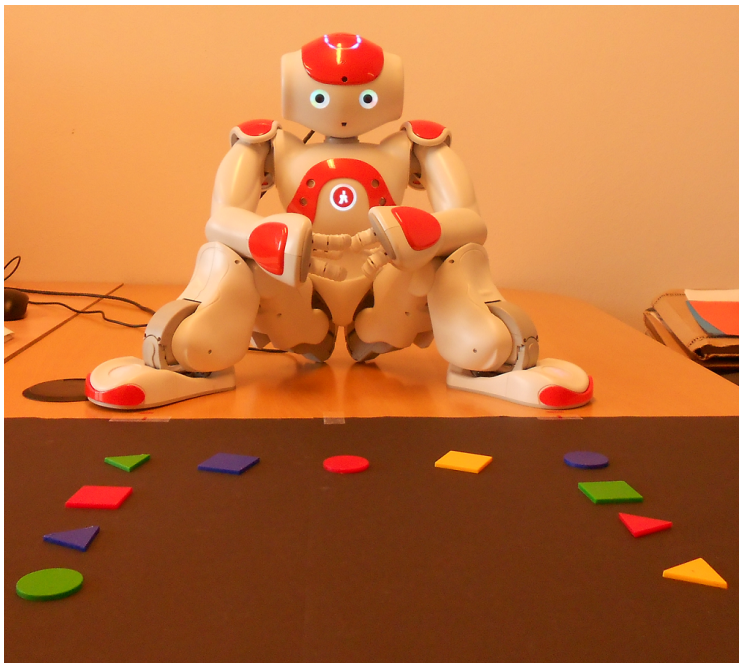

Figure 5.1: Initial state after the NAO robot has given instructions to the subject.

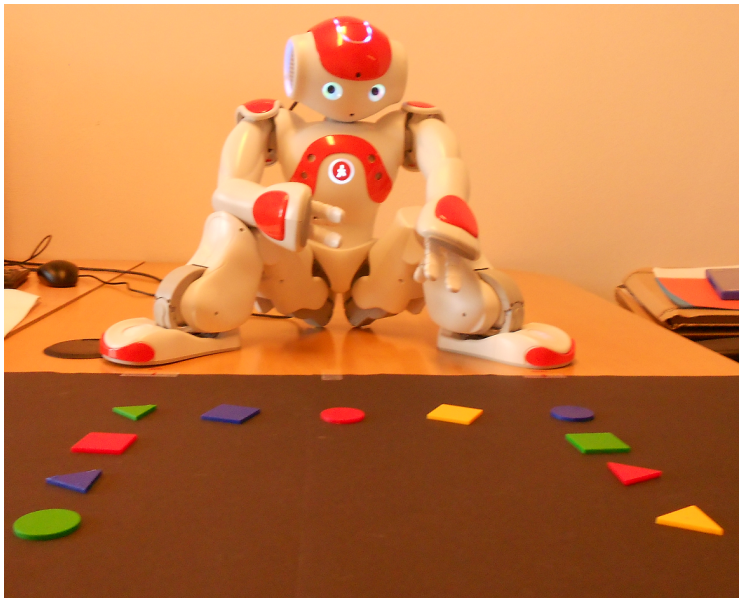

Figure 5.2: The NAO robot is gesturing towards the yellow square.

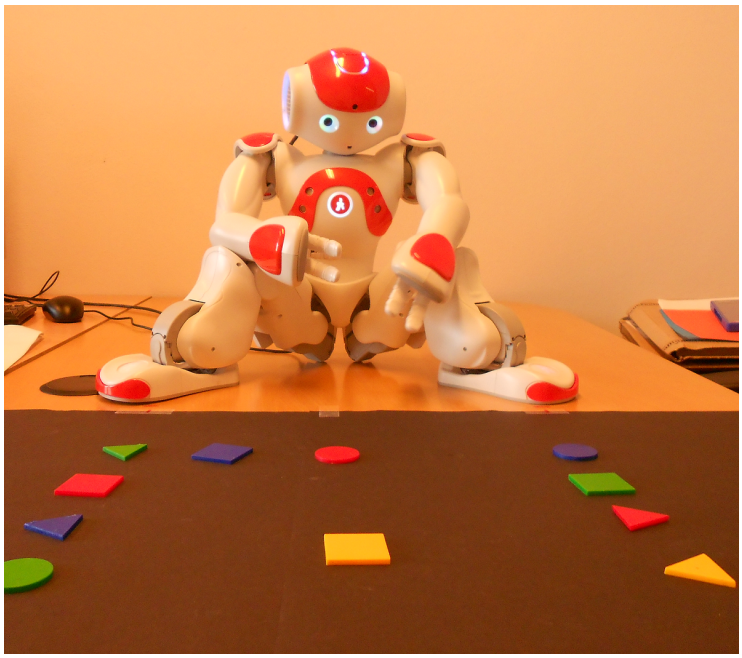

Figure 5.3: The NAO robot is gesturing towards the red triangle. (Note: the picture was taken too late which results in the NAO robot not pointing towards the red triangle.)

Figure 5.4: The NAO robot is gesturing towards the green circle.

Figure 5.5: The NAO robot is gesturing towards the red circle.

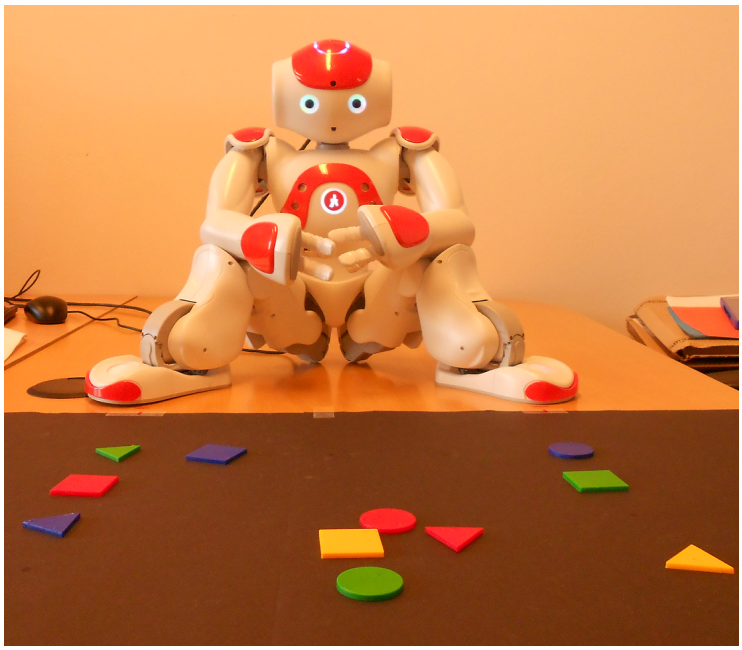

Figure 5.6: The NAO robot is instructing the subject to make an object out of the collected pieces.

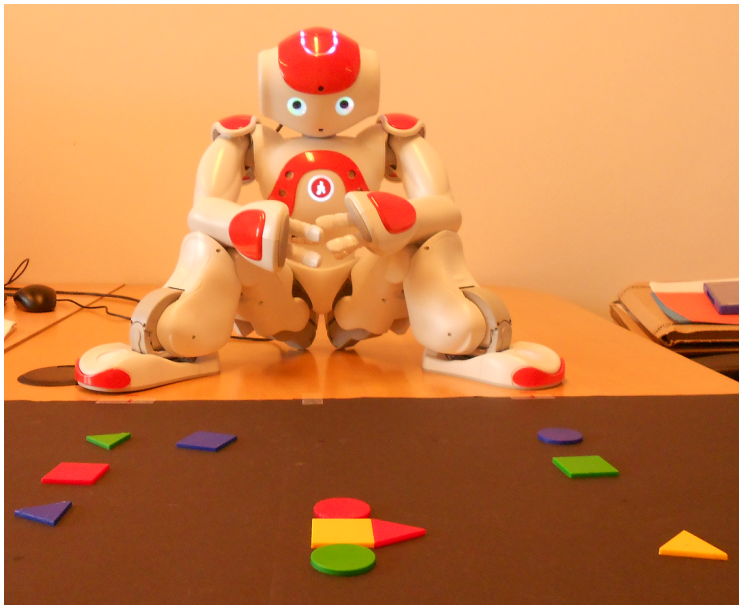

Figure 5.7: The NAO robot is returning an evaluation of the created object.

Chapter 6

Conclusions and Future Work

Developing robot applications using social robots is complex and challenging, since many individual robot skills have to be incorporated into a single system. In order to get more insight into how such a system can be developed and to actually build a system which can be used in human-robot interaction experiments a human-robot interaction game using the NAO robot platform was designed and implemented. Building a human-robot interaction game has been both interesting and challenging requiring many different tools. Since other universities had already been working with the NAO robot platform and ROS it seemed like a good idea to build the experiment using a combination of the two libraries, namely NAOqi and ROS. In retrospect, there are several reasons why it would have been better to have focused more on NAOqi instead of ROS.

First, NAOqi, on the hand one, is an extensive library designed specific for NAO robots. It comes bundled with several tools, like Choregraphe and NAOsim, which make the developing of robot applications for the NAO robot even more easily. ROS, on the other hand, is a library designed to be used by many different components and robots. This makes developing for the NAO robot using NAOqi more easily and faster in general.

Second, all the ROS stacks for the NAO robot, including the nao-stack by Albert-Ludwigs-Unversitaet, are in most cases wrappers around NAOqi. This means the stacks, in general, provide not more, but even less functionality than what NAOqi is able to provide. The only reason to use ROS with regards to the NAO robot is if you want to make use of stacks which provide functionality not present in NAOqi, for example blob detection algorithms, SLAM algorithms or RoboEarth.

6.1 Improvemens

The experiment can be improved on different aspects, like the experiment setting, gestures and evaluation.

6.1.1 Experiment Setting

In its current stage the experiment setting is quite static; the NAO robot is sitting still and has to be positioned at a specific location in order to gesture to the pieces properly and the pieces have to be laid out in a grid-like map. In order to make the experiment setting more dynamic it would help if the NAO robot is not tied to a specific location, but would walk around for example. Furthermore if the perception aspect of the system is improved the pieces do not only have to be in specific locations, but also different types of pieces, like Lego or Duplo blocks, could be used. One way to implement all these suggestions is by using a Kinect camera, since it would give the system an overview of the experiment setting. This means that is is possible to keep track of the NAO robot at all times, but also to detect more advanced pieces.

6.1.2 Gestures

Multimodal, i.e. both verbal and non-verbal components, gestures are currently only used to direct the subjects attention towards a piece. During the introduction and evaluation stages only verbal components of gestures are used. To make the behavior more natural non-verbal components can be added during those stages. Furthermore, it would also be interesting to investigate whether the color of the LEDs in eyes of the NAO robot have any effect on the gestures. For example, when the NAO robot is happy the LEDS could have a more bluish shade and when it is unhappy the LEDs could have a more reddish shade.

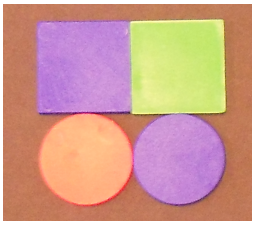

Figure 6.1: A car.

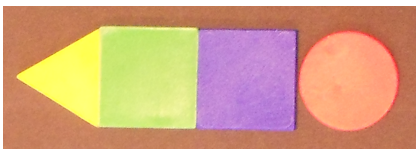

Figure 6.2: A rocket.

6.1.3 Evaluation

The evaluation of the pieces can be greatly improved. Instead of giving random evaluations a basic shape matching algorithm could be used. An example of possible shapes which can be made using the pieces can be seen in figure 6.1 and figure 6.2. A content-based image retrieval (CBIR) service could also be used

to be able to supply even more advanced evaluations. Another improvement to the evaluation phase is to have the NAO robot return feedback cues while the subject is making the object. This however means that the NAO robot needs to have a preferred final object. For example, if the NAO robot wants a house being made, it could first supply specific pieces. When the subject is creating the object it could give friendly nods when the subject is on the right track or shake his head and have its eyes turn red when the subject is not.

With the current experiment setting, the lessons learned and the list of improvements it is now possible to built new and interesting human-robot interaction experiments using the NAO robot.

Chapter 7

Testimonial

At February 3 the Artificial Intelligence Department of the Radboud University received a package which contained the latest model of the NAO robot series. Even though its head was not fully functional it was able to impress us fairly quickly with its Tai Chi demo. After the head had been taken off and shipped to Paris where it was repaired, it was received a couple of weeks later after which it was reassembled onto the body again. Up to that point it was not exactly clear what the NAO robot could do, but more importantly how it could be done. Luckily, at that moment I did not have a topic for a bachelor thesis yet and with some dumb luck I was fortunate enough to receive the opportunity to be the first to work with the NAO robot. Even though there was no end goal or any well-defined tasks at that point I took the opportunity. After a couple of weeks of reading manuals and tutorials, but even more experimenting with the actual robot I was able to do all sorts of things with the NAO robot, from basic things like saying a sentence or detecting a red ball to some more advance things like teleoperation via a PS3-controller.

Luckily enough other universities were already experimenting with the NAO robot and ROS which meant several stacks were available. This meant that the learning curve was less and made sure it was possible to complete my bachelor thesis in time.

Bibliography

[1] Robots go home. *Professional Engineering*, 17(19):28 – 29, 2004.

[2] Cynthia Breazeal. Toward sociable robots. *Robotics and Autonomous Systems*, 42(34):167 – 175, 2003. Socially Interactive Robots.

[3] J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 2061 –2066 vol.3, 2000.

[4] Open Source Community. Robot operating system. `http://www.ros.org/`, 2012. [Online; last accessed 01-September-2012].

[5] Kerstin Dautenhahn. Getting to know each other - artificial social intelligence for autonomous robots. *Robotics and Autonomous Systems*, 16(24):333 – 356, 1995.

[6] Bill Gates. A robot in every home. (cover story). *Scientific American*, 296(1):58 – 65, 2007.

[7] Bilge Mutlu. Developing robots that can teach humans. `https://www.youtube.com/watch?v=_z5o0e2Q67g`, 2012. [Online; last accessed 01-September-2012].

[8] Aldebaran Robotics. Aldebaran robotics. `http://www.aldebaran-robotics.com/`, 2012. [Online; last accessed 01-September-2012].

[9] Aldebaran Robotics. Nao software 1.12.5 documentation. `http://www.aldebaran-robotics.com/documentation/index.html`, 2012. [Online; last accessed 01-September-2012].

[10] Maha Salem, Stefan Kopp, Ipke Wachsmuth, Katharina Rohlfing, and Frank Joublin. Generation and evaluation of communicative robot gesture. *International Journal of Social Robotics*, 4:201–217, 2012. 10.1007/s12369-011-0124-9.

[11] Candace L. Sidner, Cory D. Kidd, Christopher Lee, and Neal Lesh. Where to look: a study of human-robot engagement. In *Proceedings of the 9th international conference on Intelligent user interfaces*, IUI '04, pages 78–84, New York, NY, USA, 2004. ACM.

Appendix A

Technical Documentation

In this chapter an overview of the different technical aspects, like requirements, stacks from ROS used and the organization of the code will be discussed.

A.1 Requirements

The system makes use of both the NAO robot platform and ROS. Each environment has different requirements on both hardware and software. For the development of the system a single system was used which will be described in the following sections.

A.1.1 Hardware

The computer was a standard Dell provided by the *Gebruikersdienst ICT (GDI*, which provides ICT support for all departments of the university, having the following system properties:

- **Processor**: Intel Core 2 DUO (2.33 GHz)
- **Memory**: 4 Gb
- **Graphics Card**: nVidia GeForce 6600GT

In general the hardware requirements are low, however it is recommend to have a decent graphics card since Choregraphe, NAOsim and especially rviz (3D visualization environment in ROS) do require a fair amount of graphical processing power.

A.1.2 Software

From the NAO robot platform all tools, except NAOsim which is Windows only, are able to run on Windows, Linux and OS X. However, ROS is only fully

supported for 32-bits version of Ubuntu, making Ubuntu the obvious operating system. Below a list with all the specific version of the operating system, libraries and tools which were used during development:

- **Operating System**: Ubuntu 11.10 (Oneiric) 32-bits
- **Libraries**: ROS Electric, NAOqi 1.12.5
- **Tools**: Choregraphe Suit 1.12, containing both Choregraphe and Monitor
- **Integrated Development Environment (IDE)**: Eclipse Indigo 3.7.0

A.2 ROS

The latest and recommended version of ROS available at the time was ROS Electric. The *desktop-full* configuration was installed using the Advanced Packaging Tool (APT). The installation guide at `http://www.ros.org/wiki/electric/Installation/Ubuntu` was used which can be consulted for other details.

A.2.1 Stacks

The system uses two stacks not available by default in ROS, namely a stack to control the NAO robot and a stack to be able to detect blobs.

NAO robot

Since we were not the first university to experiment with a NAO robot and ROS other stacks were already available. The Albert-Ludwigs-Unversitaet in Freiburg has been working on the NAO robot and ROS since 2010 and made several stacks available (hereafter called the Freiburg-stacks), which are recommended by ROS. Brown University has made another stack available for the NAO robot[1], however development on the stack has been ceased.

The Freiburg-stacks consist of several stacks, namely *humanoid_msgs*, *nao_robot* and *nao_common*. To install the Freiburg-stacks the installation guide at `http://www.ros.org/wiki/nao/Installation` was used. The individual stacks and how they are used by the system are discussed in the following paragraphs.

humanoid_msgs The humanoid_msgs-stack can be found at `http://www.ros.org/wiki/humanoid_msgs` and contains generic messages and services for humanoid robots. While the humanoid_msgs-stack is not directly used by the system it is a prequisite to be able to use the other stacks.

[1] `code.google.com/p/brown-ros-pkg/downloads`

nao_robot The nao_robot-stack can be found at `http://www.ros.org/wiki/nao_robot` and contains several useful nodes, like a camera node to publish images from the video camera or a driver node to provide access to walking commands and joint control. The system uses the nao_robot-stack to sent poses of the gestures to the *body pose*-server which are then executed by the NAO robot.

nao_common The nao_common-stack can be found at `http://www.ros.org/wiki/nao_common` and contains common tools in order to remote operate the NAO robot. The stack was used during testing to see whether the NAO robot could be teleoperated using a PS3-controller, however the system does not use the stack at all.

Blob Detection

The system uses the *cmvision*-stack from the Color Machine Vision Project which can be found at `http://www.ros.org/wiki/cmvision`. It is the implementation of the algorithm as described in their paper Fast and Inexpensive Color Image Segmentation for Interactive Robots [3].

Self-created stack

The system does not only consist of third party stacks, but also an extensive self-created stack. The code of the self-created stack can be found at `https://github.com/bootsman123/nao_utils`. The stack contains several packages which will be discussed in the following paragraphs.

nao_core The nao_core-package contains providing basic functionality to connect to the broker in the NAO robot and set up proxies. Thus, every other node which wants to make use of NAOqi will have to extend the NaoNode-class.

nao_sensors The nao_sensors-package contains functionality which is very similar to functionality concerning sensors available in the nao_driver-package from the Freiburg-stack. It was primarily created to get familiar developing nodes in ROS.

nao_landmark_detection The nao_landmark_detection-package contains functionality to detect NAO marks[2]. NAO marks are similar to barcodes and can be natively and quite robustly be detected using NAOqi. The system does not use the nao_landmark_detection-package, however during testing and familiarizing with the NAO robot the package was used.

[2] `http://www.aldebaran-robotics.com/documentation/naoqi/vision/allandmarkdetection.html`

nao_speech The nao_speech-package contains functionality which to let the NAO robot speak. The system uses the nao_speech-package whenever the verbal component of the gesture has to be executed. The nao_speech-package listens to a specific topic and using NAOqi to say the messages.

nao_camera The nao_camera-package contains functionality to publish images from the video camera of the NAO robot at a certain frequency. The system does not use the nao_camera-package directly, however the nao_blob_detection-package does.

nao_blob_detection The nao_blob_detectionn contains functionality to process the image, according to the settings of the setup GUI, and to detect blobs of the processed images. The system uses the detected blobs as part of the visual perception system.

nao_experiment The nao_experiment contains all the functionality needed for the experiment. It contains both the nodes for the setup GUI and the monitoring GUI.