



Technische Universität Berlin



# **Gesture Recognition for Human-Robot Interaction: An approach based on skeletal points tracking using depth camera**

## **Masterarbeit**

am Fachgebiet Agententechnologien in betrieblichen Anwendungen und der  
Telekommunikation (AOT)

Prof. Dr.-Ing. habil. Sahin Albayrak  
Fakultät IV Elektrotechnik und Informatik  
Technische Universität Berlin

vorgelegt von

**Sivalingam Panchadcharam Aravinth**

Betreuer: Prof. Dr.-Ing. habil. Sahin Albayrak,  
Dr.-Ing. Yuan Xu

Sivalingam Panchadcharam Aravinth  
Matrikelnummer: 342899  
Sparrstr. 9  
13353 Berlin

# Statement of Authorship

I declare that I have used no other sources and aids other than those indicated. All passages quoted from publications or paraphrased from these sources are indicated as such, i.e. cited and/or attributed. This thesis was not submitted in any form for another degree or diploma at any university or other institution of tertiary education

Place, Date

Signature

# Abstract

Human-robot interaction (HRI) has been a topic of both science fiction and academic speculation even before any robots existed [?]. HRI research is focusing to build an intuitive and easy communication with the robot through speech, gestures, and facial expressions. The use of hand gestures provides an attractive alternative to complex interfaced devices for HRI. In particular, visual interpretation of hand gestures can help in achieving the ease and naturalness desired for HRI. This has motivated a very active research concerned with computer vision-based analysis and interpretation of hand gestures. Important differences in the gesture interpretation approaches arise depending on whether 3D based model or appearance based model of the gesture is used [?].

In this thesis, we attempt to implement the hand gesture recognition for robots with modeling, training, analyzing and recognizing gestures based on computer vision and machine learning techniques. Additionally, 3D based gesture modeling with skeletal points tracking will be used. As a result, on the one side, gestures will be used command the robot to execute certain actions and on the other side, gestures will be translated and spoken out by the robot.

We further hope to provide a platform to integrate Sign Language Translation to assist people with hearing and speech disabilities. However, further implementations and training data are needed to use this platform as a full fledged Sign Language Translator.

## Keywords

Human-Robot Interaction (HRI), Computer Vision, Depth Camera, Hand Gesture, 3D hand based model, Skeleton tracking, Gesture Recognition, Sign Language Translation, Hidden Markov Model, NAO

# Acknowledgements

Der Punkt Acknowledgements erlaubt es, persönliche Worte festzuhalten, wie etwa:

- Für die immer freundliche Unterstützung bei der Anfertigung dieser Arbeit danke ich insbesondere...
- Hiermit danke ich den Verfassern dieser Vorlage, für Ihre unendlichen Bemühungen, mich und meine Arbeit zu fördern.
- Ich widme diese Arbeit

Die Acknowledgements sollte stets mit großer Sorgfalt formuliert werden. Sehr leicht kann hier viel Porzellan zerschlagen werden. Wichtige Punkte sind die vollständige Erwähnung aller wichtigen Helfer sowie das Einhalten der Reihenfolge Ihrer Wichtigkeit. Das Fehlen bzw. die Hintanstellung von Personen drückt einen scharfen Tadel aus (und sollte vermieden werden).

# Contents

<b>Statement of Authorship</b>	<b>II</b>
<b>Abstract</b>	<b>III</b>
<b>Contents</b>	<b>V</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>2</b>
2.1 Computer Vision in Robotics . . . . .	2
2.1.1 NAO Vision . . . . .	2
2.1.2 Extending NAO . . . . .	3
2.2 Gesture Recognition . . . . .	3
2.2.1 Gesture Modeling . . . . .	4
2.2.2 Gestural Taxonomy . . . . .	5
2.2.3 Gesture Recognition . . . . .	6
<b>3 Goal</b>	<b>7</b>
<b>4 Solution</b>	<b>9</b>
4.1 Sensing . . . . .	9
4.2 Feature Extraction . . . . .	9
4.3 Modeling and Classification . . . . .	10
4.4 Gesture Analysis and Recognition . . . . .	12
4.5 Human-Robot Interaction . . . . .	13
<b>5 Results</b>	<b>14</b>
5.1 Schedule . . . . .	14
5.2 Details . . . . .	14

<i>CONTENTS</i>	VI
<b>6 Evaluation</b>	<b>16</b>
<b>7 Conclusion and Futurework</b>	<b>17</b>
<b>Bibliography</b>	<b>18</b>
<b>List of Figures</b>	<b>18</b>
<b>List of Tables</b>	<b>19</b>
<b>Abkürzungsverzeichnis</b>	<b>20</b>
<b>Anhang</b>	<b>21</b>
A HRI Module . . . . .	21
.1 HRI Module . . . . .	21
.1.1 Gesture Tracker . . . . .	21

# Chapter 1

## Introduction

Huge influence of computers in society has made smart devices, an important part of our lives. Availability and affordability of such devices motivated us to use them in our day-to-day living. The list of smart devices includes personal automatic and semi-automatic robots which are also playing a major role in our household. For an instance, Roomba [?] is an autonomous robotic vacuum cleaners that automatically cleans the floor and goes to its charging station without human interaction.

Interaction with smart devices has still been mostly through displays, keyboards, mouse and touch interfaces. These devices have grown to be familiar but inherently limit the speed and naturalness with which we can interact with the computer. Usage of robots for domestic and industrial purposes has been continuously increasing. Thus in recent years, there has been a tremendous push in research toward an intuitive and easy communication with the robot through speech, gestures and facial expressions.

Tremendous progress had been made in speech recognition and several commercially successful speech interfaces are available. However, speech recognition systems have certain limitations such as misinterpretation due to various accents and background noise interference. It may not be able to differentiate between your speech, other people talking and other ambient noise, leading to transcription mix-ups and errors.

Furthermore, there has been an increased interest in recent years in trying to introduce other human-to-human communication modalities into HRI. This includes a class of techniques based on the movement of the human arm and hand, or hand gestures. The use of hand gestures provides an attractive alternative for Human-robot interaction than the conventional cumbersome devices.

# Chapter 2

## Background

### 2.1 Computer Vision in Robotics

Computer vision is a broad field that includes methods for acquiring, processing, analyzing and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions [?].

Proper vision is the utmost importance for the function of any vision based autonomous robot. Areas of artificial intelligence deal with autonomous planning or deliberation for robotic systems to navigate through an environment. A detailed understanding of these environments is required to navigate through them. High-level information about the environment could be provided by a computer vision system that is acting as a vision sensor.

In this thesis, we will focus on the hand gesture recognition using computer vision techniques for a humanoid robot named as NAO, as shown in the figure 2.1. NAO is an autonomous, programmable humanoid robot developed by Aldebaran Robotics. The NAO Academics Edition was developed for universities and laboratories for research and education purposes. Table 2.1 shows the specification of NAO according to Aldebaran Robotics.

#### 2.1.1 NAO Vision

Two identical video cameras are located in the forehead of NAO. They provide up to 1280x960 resolution at 30 frames per second. NAO contains a set of algorithms for detecting and recognizing faces and shapes.



### 2.1.2 Extending NAO

3D cameras such as Microsoft Kinect and Asus Xtion are used not only for gaming but also for analyzing 3D data, including algorithms for feature selection, scene analysis, motion tracking, skeletal tracking and gesture recognition [?].

3D model based gesture recognition needs 3D data. Hence we attempt to use Asus Xtion as an external camera that will be mounted on the head of NAO as shown in the figure 2.2. Computational limitations of NAO hinders us to build an effective real time gesture recognition. Therefore, we propose to use an off-board computer to process the sensor data from NAO, execute the gesture recognition algorithm and finally command NAO to do an appropriate action.

Table 2.1: NAO V5 hardware and software specification

Height	58 centimetres (23 in)
Weight	4.3 kilograms (9.5 lb)
Battery autonomy	60 minutes (active use), 90 minutes (normal use)
Degrees of freedom	21 to 25
CPU	Intel Atom @ 1.6 GHz
Built-in OS	Linux
SDK compatibility	Windows, Mac OS, Linux
Programming languages	C++, Python, Java, MATLAB, Urbi, C, .Net
Vision	2 x HD 1280x960 cameras
Connectivity	Ethernet, Wi-Fi
Sensors	4 x directional microphones 1 x sonar rangefinder 2 x IR emitters and receivers 1 x inertial board 9 x tactile sensors 8 x pressure sensors

## 2.2 Gesture Recognition

Human hand gestures are a means of non-verbal interaction among people. They range from simple actions of using our hand to point at, to the more complex ones that express our feelings and allow us to communicate with others. To exploit the use of gestures in HRI, it is necessary to provide the means by which they can be interpreted by robots. The HRI interpretation of gestures requires that dynamic and/or static configurations of the human hand, arm and even other parts of the human body, be measurable by the machine [?].



Figure 2.1: NAO

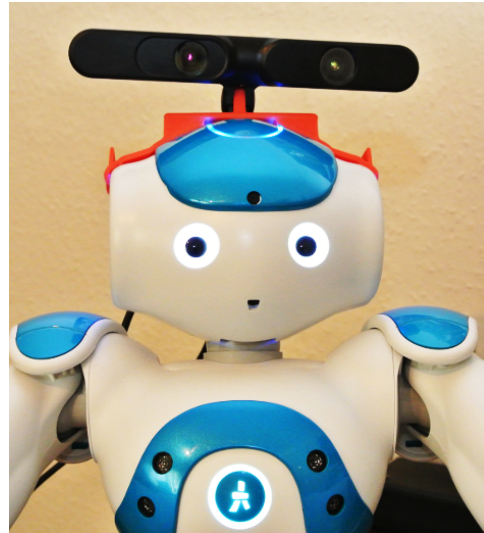


Figure 2.2: Asus Xtion mounted on NAO

Initial attempts to recognize hand gestures resulted in electro-mechanical devices that directly measure hand and/or arm joint angles and spatial position using sensors [?]. Glove-based gestural interfaces require the user to wear such a complex device that hinders the ease and naturalness with which the user can interact with the computer controlled environment.

Even though such hand gloves are used in highly specialized domain such as simulation of medical surgery or even in the real surgery, the everyday user will be certainly deterred by such sophisticated interfacing devices. As an active result of the motivated research in HRI, computer vision based techniques were innovated to augment the naturalness of interaction.

### 2.2.1 Gesture Modeling

Figure 2.3 shows various types of modeling techniques used for Gesture modeling [?]. Selection of an appropriate gesture modeling depends primarily on the intended application. For an application that needs just hand gesture to go up and down or left and right, a very simple model may be sufficient. However, if the purpose is a natural-like interaction, a model has to be sophisticated enough to interpret all the possible gesture. The following section discusses various gesture modeling techniques which are being used by the existing hand gesture recognition applications.

Appearance based models don't use the spatial representation of the body, because they derive the parameters directly from the images or videos using a template database.

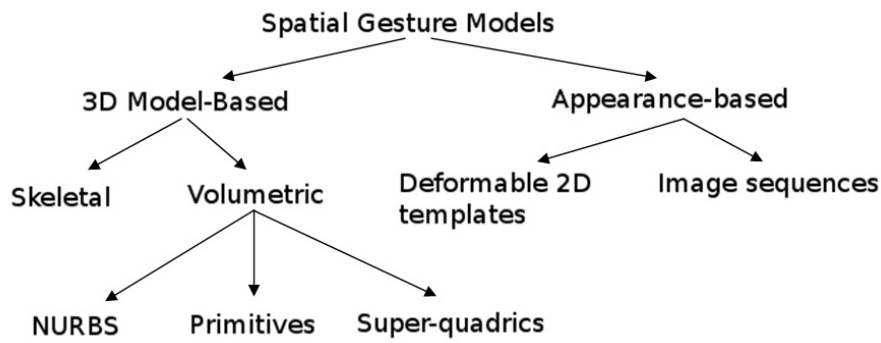


Figure 2.3: Classification of gesture modeling [?].

3D hand based model approach uses volumetric or skeletal models, or even a combination of both. Volumetric approaches have been heavily used in computer animation industry and for computer vision purposes. The models are generally created of complicated 3D surfaces. The drawback of this method is that is very computational intensive.

Instead of using intensive processing of the 3D hand models and dealing with a lot of parameters, one can just use a simplified version that analyses the joint angle parameters along with segment length. This is known as a skeletal representation of the body, where a virtual skeleton of the person is computed and parts of the body are mapped to certain segments [?]. The analysis here is done using the position and orientation of these segments and the relation between each one of them.

In this thesis, we focus on Skeletal based modeling algorithms which are faster because the detection program has to focus only on the significant parts of the body. Moreover, it allows to do pattern matching against a template database.

### 2.2.2 Gestural Taxonomy

Several alternative taxonomies have been suggested that deal with psychological aspects of gestures [?]. All hand/arm movements are first classified into two major classes as shown in the figure 2.4.

Manipulative gestures are the ones used to act on objects. For example, moving a chair from one location to another. Manipulative gestures in the context of HRI are mainly developed for medical surgery. Communicative gestures, on the other hand, have purely communicational purpose. In a natural environment they are usually accompanied by speech or spoken as a sign language. In HRI context these gesture are one of the commonly used gestures, since they can often be represented by static as well as dynamic hand postures.

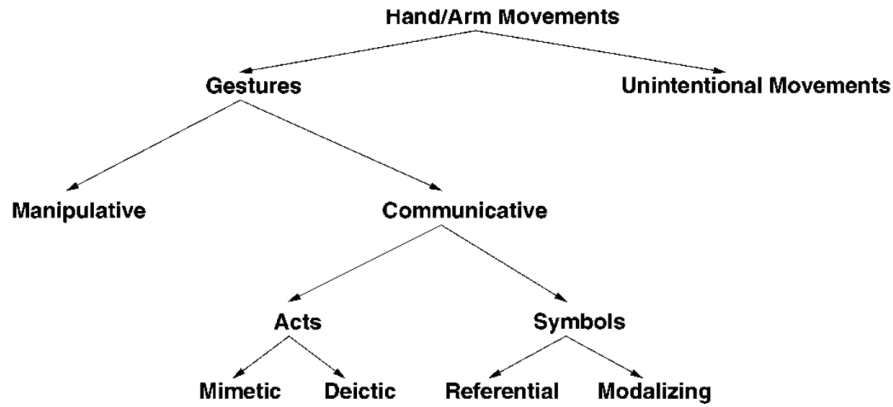


Figure 2.4: Classification of gestures [?].

In this thesis, we focus on communicative gestures in the form of symbols. They symbolize some referential action. For instance, circular motion of hand may be referred as an alphabet "O" or as an object such as wheel or as a command to turn in a circular motion .

### 2.2.3 Gesture Recognition

The task of gesture recognition shares certain similarities with other recognition tasks, such as speech recognition and biometrics. Though alternatives such as Dynamic Programming (DP) matching algorithms have been attempted, the most successful solutions involves feature-based statistical learning algorithm, usually Hidden Markov Models [?].

In this thesis, we have chosen a machine learning technique based on hidden Markov model to recognize gestures. HMM is a statistical model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. In simpler Markov models, the state is directly visible to the observer and therefore the state transition probabilities are the only parameters. In a hidden Markov model, the state is not directly visible, but output dependent on the state is visible.

# Chapter 3

## Goal

As described earlier, HRI research is focusing to build an intuitive and easy communication with the robot through speech, gestures and facial expressions. The use of hand gestures provides the ease and naturalness with which the user can interact with robots.

In this thesis, we attempt to implement the feature for NAO to recognize gestures and execute predefined actions based on the gesture. NAO will be extended with an external depth camera, that will enable NAO to recognize 3D modeled gestures. This 3D camera will be mounted on the head of NAO, so that it can scan for gestures in the horizon. Additionally, skeletal points tracking algorithm with machine learning technique using Hidden Markov Models will be used to recognize the gestures. Due to the computational limitations of NAO, gesture recognition algorithm will be executed on off-board computer. With the hand gesture recognizing feature, NAO will be available to the users in two modes.

- **Command mode:** In this mode, a gesture will be recognized by NAO and related task will be executed. Even though the gesture based interaction is real time, NAO can not be interrupted or stopped by using any gesture while it is executing a task. However, other interfaces such as voice commands can be used in such situation to stop or interrupt the ongoing task execution.
- **Translation mode:** In this mode, NAO will be directly translating the meaning of the gesticulated gestures. To achieve this, text-to-speech library of NAO will be used and recognized gesture can be spoken out using the integrated loudspeaker. In future, it will allow NAO to translate a sign language to assist people with hearing and speech disabilities.

In this thesis, we planned to train NAO with few very simple gestures due to the reason that NAO has computational limitations. Gestures will involve both the hands or single hand to interact with the robot.

# Chapter 4

## Solution

The figure 4.1 shows the flow diagram of hand gesture recognition system that is going to implemented in this thesis. Each block contains different software components that are executed sequentially. However, training phase must be carried out before this system is available for recognition. Finally, these components will be integrated into NAO.

### 4.1 Sensing

3D camera records 30 frames of color image as well as depth image per second and outputs as a data package. Figure 4.2 shows the single frame of depth image taken from Microsoft Kinect where darker gray values represent the farther distance and lighter gray values represent the closer distance from the camera.

### 4.2 Feature Extraction

Output package from sensor data will be inputted to feature detection and extraction unit. OpenNI is a software component that will track the anatomical landmarks of the human body from the package and extract significant joint angle parameters along with segment length and present them three dimensionally as shown in the figure 4.3. Finally, only joints of both the arms will be picked out from the array of features, since it will be the significant joints needed for hand gesture recognition.

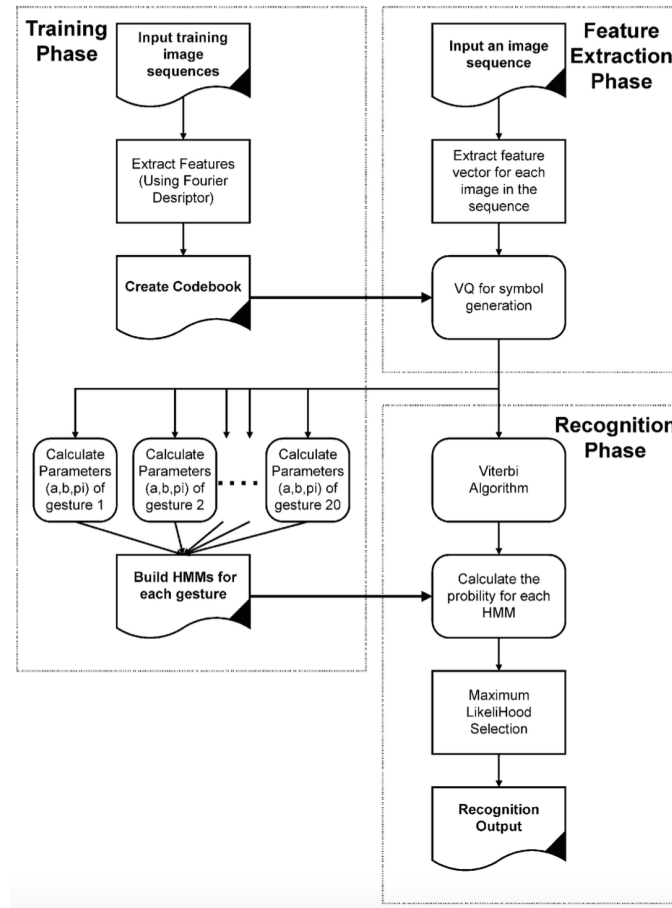


Figure 4.1: Flow diagram of hand gesture recognition system [?].

### 4.3 Modeling and Classification

In order to use this hand recognition system, all chosen gestures must be observed and the system must be trained. Therefore, a set of simple gestures will be chosen and observed for training. Each gesture is isolated in time and gesticulated for certain duration. However, sensors provides 30 frames of discrete states of gesture per second.

For example, a gesture is gesticulated by simply drawing a circle in the air and its ideal states are shown in the figure 4.4. It will be as a position of hand passing through 8 states of the circle. Each state is a point in space with x, y and z axis data. This approach makes it possible for us to reduce our observation data to sequential 3D points and focus on the recognition task without processing all those pixels. Each trained model can then be used to determine with what probability a given gesture appears in test data. Therefore, the trained Hidden Markov models will be used to recognize gestures [?].





Figure 4.2: Depth image from 3D Camera

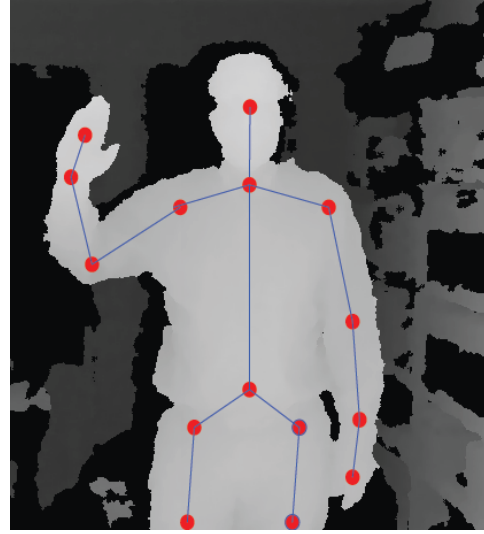


Figure 4.3: Skeleton tracking

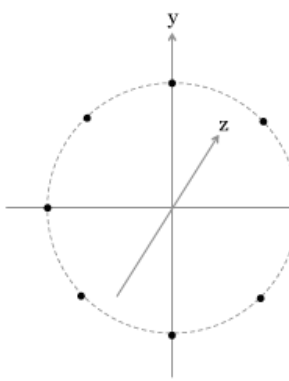


Figure 4.4: HMM States of circular gesture [?].

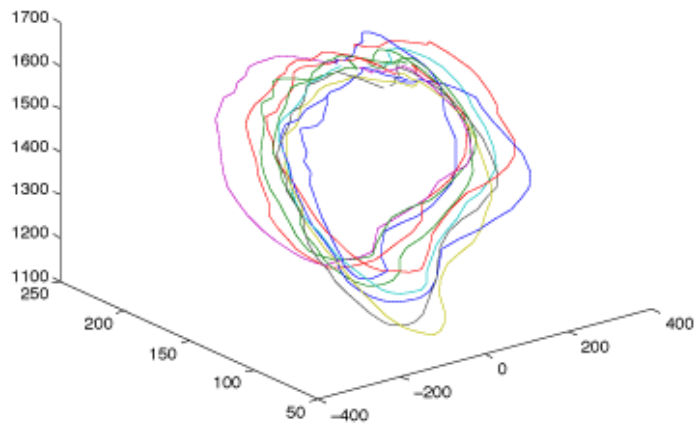


Figure 4.5: Observations of a circular gesture in x, y, z axis which are recorded by a depth camera [?].

## 4.4 Gesture Analysis and Recognition

This step contains the analysis of gesticulated gesture and finding out the likelihood of that gesture with trained data, known as gesture recognition. Figure 4.5 contains colored rings of noisy data of gestures that represent instances of a real circle gesture. Sensing and feature detection module will produce 60 observations of the circle gesture for 2 seconds, since the depth camera records at 30 frame per second. To decide whether a given set of 60 observations contains a circle gesture, we need to first determine the likelihood that the hand passed through the eight states of the gesture in the expected sequence.

Discrete HMM is a finite set of possible output symbols and a sequence of hidden states which reveal some probability. To reduce our real gesture data to a workable number of discrete output symbols and states, we can use any clustering algorithm to cluster the 3D points of all our training data of circle gesture into clusters and label them. That is to say, every point in the training data represents an output symbol that is closely tied to one of the 8 true states of the model.

Looking at the labeled data, we can estimate how likely it is that a hand passed through the 8 clusters in the same sequence as a circle gesture and if the likelihood is high enough, then the gesture is considered to be recognized.

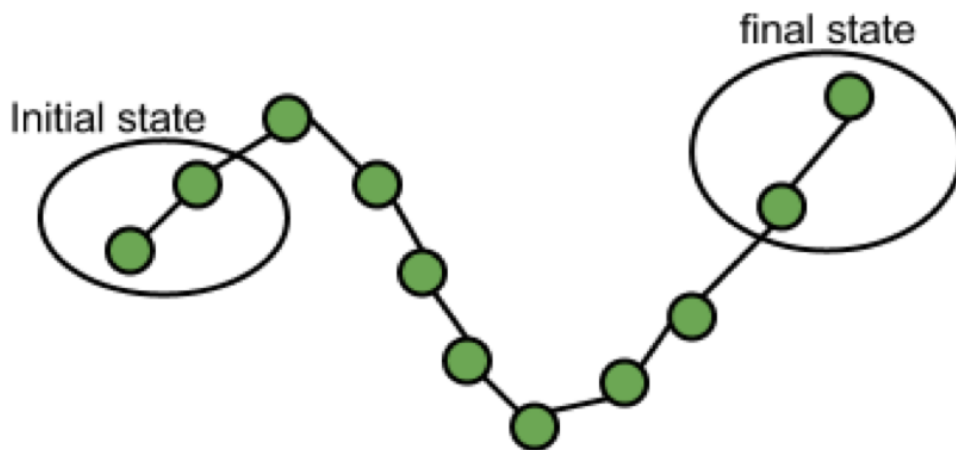


Figure 4.6: The states of Arabic sign language to convey "Hello" [?]. It is consisted of 11 states of Markov models and if there is higher likelihood of the hand position passed through states sequentially from initial to final, then it is recognized as "Hello".

## 4.5 Human-Robot Interaction

Finally, the recognized gesture will be interpreted by NAO to execute a specified task. For example, circle gesture would ask NAO to turn around. However, NAO will also be available in Translation Mode by using ALTextToSpeech Library to translate the recognized gesture.

# Chapter 5

## Results

### 5.1 Schedule

- **November - Setup:** Software and Hardware environment setup. Getting to know the tools that will be used during this project work.
- **December / January - Implementation:** Proposed functionalities will be implemented.
- **January / February - Testing:** This includes testing, bug fixing and improvising the implementation.
- **February / March - Documentation:** This includes the documentation of the code and writing up the thesis.

### 5.2 Details

- Language of the Master Thesis: English
- Document processing system: LaTeX with TexStudio
- Operating system: Linux Ubuntu, Mac OSX
- Software: OpenNI, OpenCV, NAO Choregraphe
- Hardware : NAO, Asus Xtion
- Programming Language: C++, Python
- Version control system: Git

- Advisor: Dr. Yuan Xu
- Professor: Prof. Dr. Sahin Albayrak

# Chapter 6

## Evaluation

Länge: ca 1-5 Seiten

Sind die gesteckten Ziele zur Problemlösung durch die Implementierung erreicht worden? Was kann die vorgestellte Lösung, was kann sie nicht. Des Weiteren gehören zu einer Implementierung auch immer Tests, ob die Implementierung erfolgreich war! Diese Tests müssen auch dokumentiert werden. In diesem Kapitel sollte daher eine Beschreibung des Aufbaus und der Ergebnisse von Testläufen/Simulationen vorhanden sein. Ebenso sollte eine Interpretation der Ergebnisse die Tests abschließen. Es ist auch wichtig, nicht nur positive, sondern auch negative Ergebnisse zu dokumentieren und zu interpretieren.

# **Chapter 7**

## **Conclusion and Futurework**

Länge: ca. 1-2 Seiten

Das Fazit dient dazu, die wesentlichen Ergebnisse der Arbeit und vor allem die entwickelte Problemlösung und den erreichten Fortschritt darzustellen. (Sie haben Ihr Ziel erreicht und dies nachgewiesen).

Im Ausblick werden Ideen für die Weiterentwicklung der erstellten Lösung aufgezeigt. Der Ausblick sollte daher zeigen, dass die Ergebnisse der Arbeit nicht nur für die in der Arbeit identifizierten Problemstellungen verwendbar sind, sondern darüber hinaus erweitert sowie auf andere Probleme übertragen werden können.

# List of Figures

2.1	NAO . . . . .	4
2.2	Asus Xtion mounted on NAO . . . . .	4
2.3	Classification of gesture modeling [?]. . . . .	5
2.4	Classification of gestures [?]. . . . .	6
4.1	Flow diagram of hand gesture recognition system [?]. . . . .	10
4.2	Depth image from 3D Camera . . . . .	11
4.3	Skeleton tracking . . . . .	11
4.4	HMM States of circular gesture [?]. . . . .	11
4.5	Observations of a circular gesture in x, y, z axis which are recorded by a depth camera [?]. . . . .	11
4.6	The states of Arabic sign language to convey "Hello" [?]. It is consisted of 11 states of Markov models and if there is higher likelihood of the hand position passed through states sequentially from initial to final, then it is recognized as "Hello". . . . .	12



# List of Tables

2.1	NAO V5 hardware and software specification . . . . .	3
-----	--	---

# Abkürzungsverzeichnis

<b>AES</b>	Advanced Encryption Standard (Symmetrisches Verschlüsselungsverfahren)
<b>ASCII</b>	American Standard Code for Information Interchange (Computer-Textstandard)
<b>BMP</b>	Windows Bitmap (Grafikformat)
<b>dpi</b>	dots per inch (Punkte pro Zoll; Maß für Auflösung von Bilddateien)
<b>GIF</b>	Graphics Interchange Format (Grafikformat)
<b>HTML</b>	Hypertext Markup Language (Textbasierte Webbeschreibungssprache)
<b>JAP</b>	Java Anon Proxy
<b>JPEG</b>	Joint Photographic Experts Group (Grafikformat)
<b>JPG</b>	Joint Photographic Experts Group (Grafikformat; Kurzform)
<b>LED</b>	Light Emitting Diode (lichtemittierende Diode)
<b>LSB</b>	Least Significant Bit
<b>MD5</b>	Message Digest (Kryptographisches Fingerabdruckverfahren)
<b>MPEG</b>	Moving Picture Experts Group (Video- einschließlich Audiokompression)
<b>MP3</b>	MPEG-1 Audio Layer 3 (Audiokompressionsformat)
<b>PACS</b>	Picture Archiving and Communication Systems
<b>PNG</b>	Portable Network Graphics (Grafikformat)
<b>RGB</b>	Rot, Grün, Blau (Farbmodell)
<b>RSA</b>	Rivest, Shamir, Adleman (asymmetrisches Verschlüsselungsverfahren)
<b>SHA1</b>	Security Hash Algorithm (Kryptographisches Fingerabdruckverfahren)
<b>WAV</b>	Waveform Audio Format (Audiokompressionsformat von Microsoft)
<b>YUV</b>	Luminanz Y, Chrominanzwerte U, V (Farbmodell)

# Anhang

Hier befinden sich für Interessierte Quelltexte sowie weitere zusätzliche Materialien wie Screenshots oder auch weiterführende Informationen.

## .1 HRI Module

### .1.1 Gesture Tracker

```
1  /**
2   * Author: Aravinth Panchadcharam
3   * Email: me@aravinth.info
4   * Date: 28/12/14.
5   * Project: Gesture Recognition for Human-Robot Interaction
6   */
7
8  #include <iostream>
9  #include <sstream>
10 #include <boost/log/trivial.hpp>
11 #include <boost/lexical_cast.hpp>
12 #include <boost/shared_ptr.hpp>
13 #include "NiTE.h"
14 #include "gesture_tracker.h"
15
16 using boost::property_tree::ptree;
17 using boost::property_tree::write_json;
18
19 /**
20  *
21  * Constructor
22  *
23  */
24 gesture_tracker::gesture_tracker(udp_server *server) {
25     server_ = server;
26
27     // Initialize the variables here because compiling on virtual nao forbides the ↵
28     initialization in the header file
29     leftHand = 0;
30     rightHand = 0;
31     lastLostHand = 0;
32     handsSize = 0;
```

```

32 }
33
34
35 /**
36 *
37 * Initializes NiTE and takes available OpenNI device
38 * NiTE::initialize() takes OpenNI deviceId as argument, if there ar many
39 * Wave and Click Gestures are initiated
40 *
41 */
42 void gesture_tracker::init_nite(){
43     niteRc = nite::NiTE::initialize();
44     if (niteRc != nite::STATUS_OK)
45     {
46         BOOST_LOG_TRIVIAL(error) << "NiTE_initialization_failed" ;
47     }
48     else
49     {
50         BOOST_LOG_TRIVIAL(info) << "NiTE_initialized" ;
51     }
52
53     // Check for OpenNI device and start hand tracker.
54     niteRc = handTracker.create();
55     if (niteRc != nite::STATUS_OK)
56     {
57         BOOST_LOG_TRIVIAL(error) << "Couldn't_create_hand_tracker" ;
58     }
59     else
60     {
61         BOOST_LOG_TRIVIAL(info) << "Hand_tracker_created" ;
62     }
63
64     // handTracker.setSmoothingFactor(0.1);
65     handTracker.startGestureDetection(nite::GESTURE_WAVE);
66     handTracker.startGestureDetection(nite::GESTURE_CLICK);
67     BOOST_LOG_TRIVIAL(info) << "Wave_your_hand_to_start_the_hand_tracking" ;
68 }
69
70
71 /**
72 *
73 * Serializes gesture data with position of hand at which gesture was detected
74 * Send it to the connected client
75 *
76 */
77 void gesture_tracker::send_gesture(const nite::GestureData& gesture){
78
79     std::string gestureJson;
80     if(gesture.getType() == 0){
81         gestureJson = "{\"GESTURE\":\"WAVE\"}";
82     }else if (gesture.getType() == 1){
83         gestureJson = "{\"GESTURE\":\"CLICK\"}";
84     }
85
86     boost::shared_ptr<std::string> message(new std::string(gestureJson));

```

```

87     server_->send(message);
88 }
89
90
91 void gesture_tracker::send_info(std::string info){
92
93     std::stringstream infoBuffer;
94     infoBuffer << "{\"INFO\":\"" << info << "\"}";
95
96     boost::shared_ptr<std::string> message(new std::string(infoBuffer.str()));
97     server_->send(message);
98 }
99
100
101
102 /**
103  *
104  * Serializes hand data with position and hand id
105  * Send it to the connected client
106  *
107  */
108 ptree gesture_tracker::parseToJson(const nite::HandData& hand){
109
110     // keys and values for json array
111     ptree joint_array, xAxis, yAxis, zAxis;
112
113     xAxis.put("", hand.getPosition().x);
114     yAxis.put("", hand.getPosition().y);
115     zAxis.put("", hand.getPosition().z);
116
117     // Make an array of hand joint for x,y,z axes
118     joint_array.push_back(std::make_pair("", xAxis));
119     joint_array.push_back(std::make_pair("", yAxis));
120     joint_array.push_back(std::make_pair("", zAxis));
121
122     return joint_array;
123 }
124
125
126 /**
127  *
128  * Serializes hand data with position and hand id
129  * Send it to the connected client
130  *
131  */
132 void gesture_tracker::send_hand(const nite::HandData& hand){
133
134     ptree handJson;
135     std::string handName = getHandName(hand.getId());
136
137     if(!handName.empty())
138     {
139         // Parse it json array and add to object
140         handJson.add_child(handName, parseToJson(hand));
141

```

```

142         // Stringify the ptree
143         std::ostringstream hand_buffer;
144         write_json (hand_buffer, handJson, false);
145         boost::shared_ptr<std::string> message(new std::string( hand_buffer.str()));
146
147         server_>send(message);
148     }
149 }
150
151
152 /**
153  *
154  * Serializes hand data with position and hand id
155  * Send it to the connected client
156  *
157  */
158 void gesture_tracker::send_hand(const nite::HandData& hand1, const nite::HandData&
    hand2) {
159
160     ptree handJson;
161     std::string handName1 = getHandName(hand1.getId());
162     std::string handName2 = getHandName(hand2.getId());
163
164     if(!handName1.empty() && !handName2.empty())
165     {
166         // Parse it json array and add to object
167         handJson.add_child(handName1, parseToJson(hand1));
168         handJson.add_child(handName2, parseToJson(hand2));
169
170         // Stringify the ptree
171         std::ostringstream hand_buffer;
172         write_json (hand_buffer, handJson, false);
173         boost::shared_ptr<std::string> message(new std::string( hand_buffer.str()));
174
175         server_>send(message);
176     }
177 }
178
179
180 /**
181  *
182  *
183  * Finds the hand name based on given handId
184  *
185  */
186
187 std::string gesture_tracker::getHandName(int handId) {
188     std::string handName;
189
190     if(handId == leftHand) {
191         handName = "LEFT";
192     }
193     else if(handId == rightHand) {
194         handName = "RIGHT";
195     }

```

```

196
197     return handName;
198 }
199
200
201 /**
202  *
203  * Starts Gesture recognition and Hand tracking based on the position of Hand found ,
204  * by WAVE gesture
205  *
206  */
207 void gesture_tracker::track_gestures() {
208
209     nite::HandTrackerFrameRef handTrackerFrame;
210
211     while (!utils::wasKeyboardHit())
212     {
213         niteRc = handTracker.readFrame(&handTrackerFrame);
214         if (niteRc != nite::STATUS_OK)
215         {
216             BOOST_LOG_TRIVIAL(error) << "Get_next_frame_failed";
217             continue;
218         }
219
220         const nite::Array<nite::GestureData>& gestures = handTrackerFrame.getGestures();
221
222         for (int i = 0; i < gestures.getSize(); ++i)
223         {
224             if (gestures[i].isComplete())
225             {
226                 send_gesture(gestures[i]);
227                 nite::HandId newId;
228
229                 // Dont track more than 2 hands. handTrackerFrame.getHands().getSize()
230                 // sometime goes to 3 even though
231                 // there are 2 active hands
232                 if (gestures[i].getType() == 0) {
233                     handTracker.startHandTracking(gestures[i].getCurrentPosition(), &
234                     newId);
235                 }
236             }
237         }
238
239         const nite::Array<nite::HandData>& hands = handTrackerFrame.getHands();
240
241         // hands.getSize() gives the number of active hands, but handId increases
242         // hands.getSize() = 0 and goes to hands.getSize() = 2, when there is a new ,
243         // hand detected
244         // hands.getSize() = 0, this happens for a fraction of second when tacking is,
245         // troubled
246         for (int i = 0; i < hands.getSize(); ++i)
247         {
248             // Get Hand data
249             const nite::HandData& hand = hands[i];

```

```

245
246 // If hand is lost, update the losthandid
247 if(!hand.isTracking())
248 {
249     lastLostHand = hand.getId();
250     BOOST_LOG_TRIVIAL(info) << getHandName(hand.getId()) << "_Hand_with_"
        id_ << hand.getId() << "_is_Lost";
251
252     // When there is no active hands, reset all the values
253     // Last active hand
254     if(hands.getSize() == 1){
255         leftHand = 0;
256         rightHand = 0;
257         lastLostHand = 0;
258         handsSize = 0;
259
260         send_info("Both_hands_are_lost");
261     }
262     else
263     {
264         send_info( getHandName(hand.getId()) + "_Hand_is_lost");
265     }
266 }
267
268 // If new hand is found
269 if(hand.isNew()){
270     BOOST_LOG_TRIVIAL(info) << "Found_new_hand_with_id_" << hand.getId();
271
272     handsSize++;
273
274     // Check if it is a hand for the first time or second time
275     if(handsSize == 1 && lastLostHand == 0){
276         BOOST_LOG_TRIVIAL(debug) << "First_hand_is_found";
277         rightHand = hand.getId();
278     }
279     else if (handsSize == 2 && lastLostHand == 0){
280         BOOST_LOG_TRIVIAL(debug) << "Second_hand_is_found";
281         leftHand = hand.getId();
282     }
283     // If a hand was lost and a hand is active, then update the
        appropriate id to left or right hand
284     else if(handsSize > 2 && lastLostHand > 0){
285         if(lastLostHand == leftHand){
286             leftHand = hand.getId();
287         }else if(lastLostHand == rightHand){
288             rightHand = hand.getId();
289         }
290     }
291
292     send_info( getHandName(hand.getId()) + "_Hand_is_new");
293 }
294
295 if(hand.isTouchingFov()){
296     send_info( getHandName(hand.getId()) + "_Hand_is_at_FOV");
297 }

```



```
298
299     }
300
301     if(hands.getSize() == 2 && hands[0].isTracking() && !hands[0].isNew() &&
        hands[1].isTracking() && !hands[1].isNew()){
302         send_hand(hands[0], hands[1]);
303     }
304     else if(hands.getSize() == 1 && hands[0].isTracking() && !hands[0].isNew()){
305         send_hand(hands[0]);
306     }
307
308     }
309
310     nite::NiTE::shutdown();
311 }
312
313
314 /**
315  *
316  * Called by the main thread and Boost ioService keeps it in the loop
317  *
318  */
319 void gesture_tracker::run(){
320     init_nite();
321     track_gestures();
322 }
```