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

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 foerdern.
- 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

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

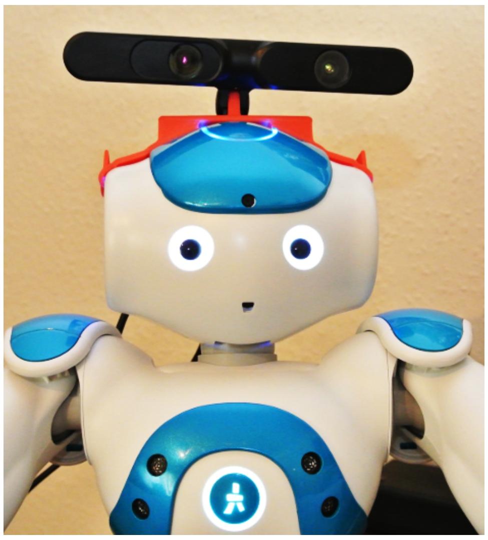
Figure 2.2: Asus Xtion mounted on NAO

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 light, 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.

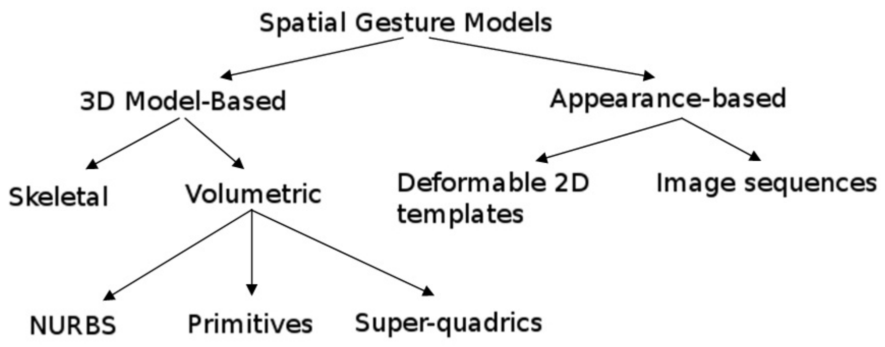

Figure 2.3: Classification of gesture modeling [?].

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.

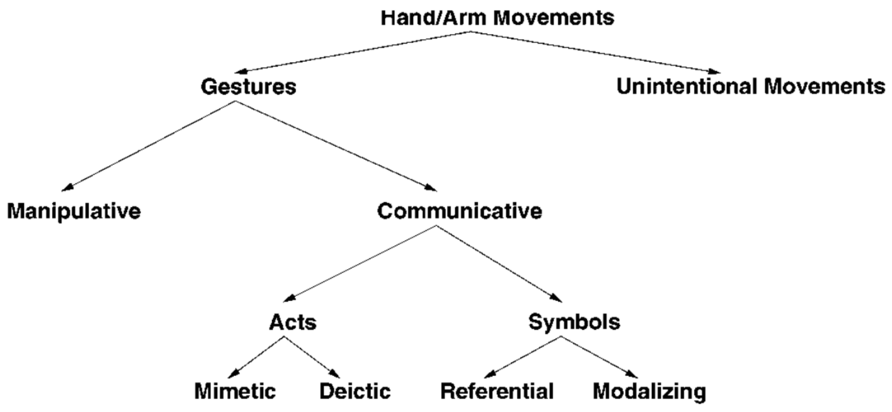

Figure 2.4: Classification of gestures [**?**].

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.

2.3 Software and Libraries

2.3.1 OpenNI 2

2.3.2 NiTE 2

2.3.3 boost

2.3.4 Gentoo Nao OS

2.3.5 UDP

2.3.6 Websocket

2.3.7 WebGL ThreeJS

2.3.8 C++ Javascript Python

2.3.9 Asus Xtion

2.3.10 AL Libraries Motion, TTS, RobotPosture

2.3.11 Gesture Recognition Toolkit

2.3.12 Adaptive Naive Bayes Classifier

2.3.13 Build Environment Xcode, Clang GCC cmake

2.3.14 Traffic Police Hand signals

2.3.15 libStdC++ Hack

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.

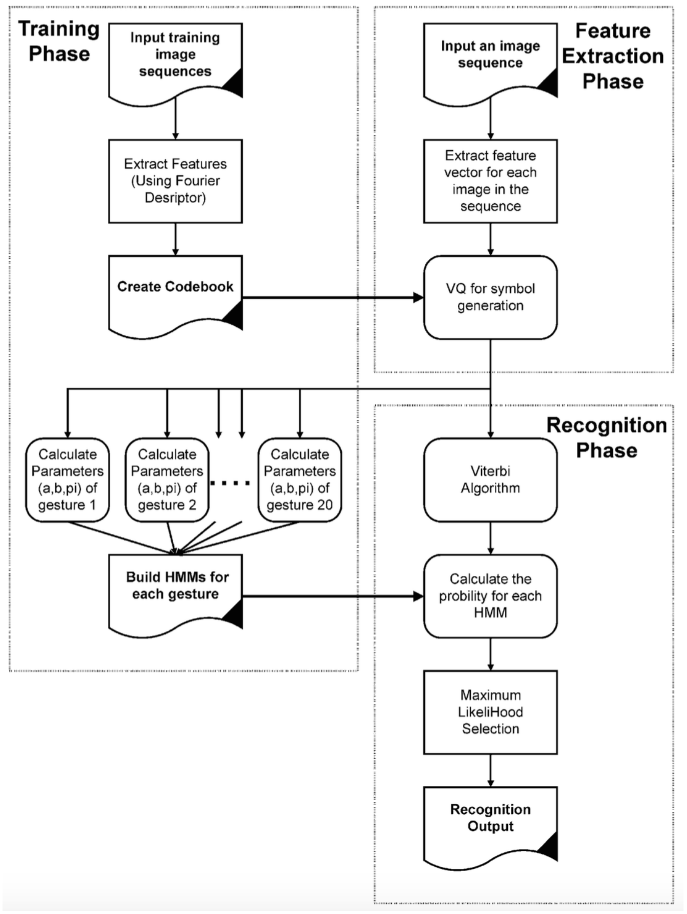
Training Phase
Input training image sequences
Extract Features (Using Fourier Desriptor)
Create Codebook
Feature Extraction Phase
Input an image sequence
Extract feature vector for each image in the sequence
VQ for symbol generation
Calculate Parameters (a,b,pi) of gesture 1
Calculate Parameters (a,b,pi) of gesture 2
Calculate Parameters (a,b,pi) of gesture 20
Build HMMs for each gesture
Recognition Phase
Viterbi Algorithm
Calculate the probility for each HMM
Maximum LikeliHood Selection
Recognition Output

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 [**?**].

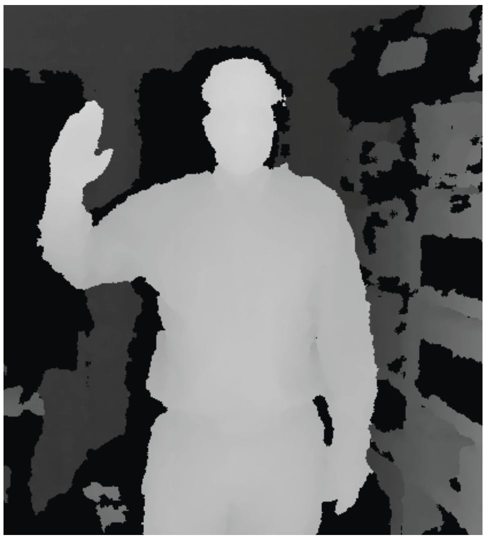

Figure 4.2: Depth image from 3D Camera

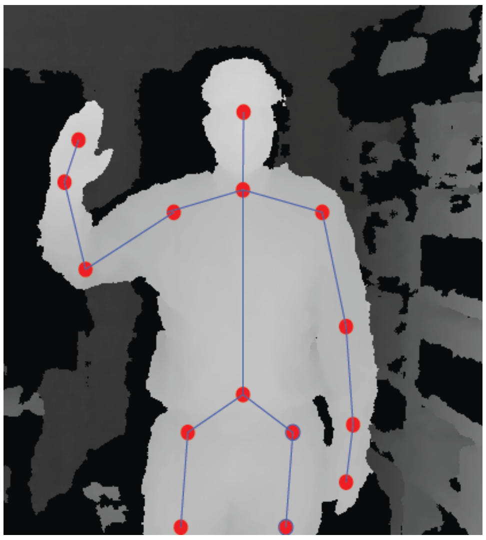

Figure 4.3: Skeleton tracking

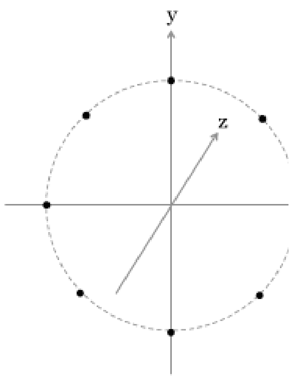

Figure 4.4: HMM States of circular gesture [?].

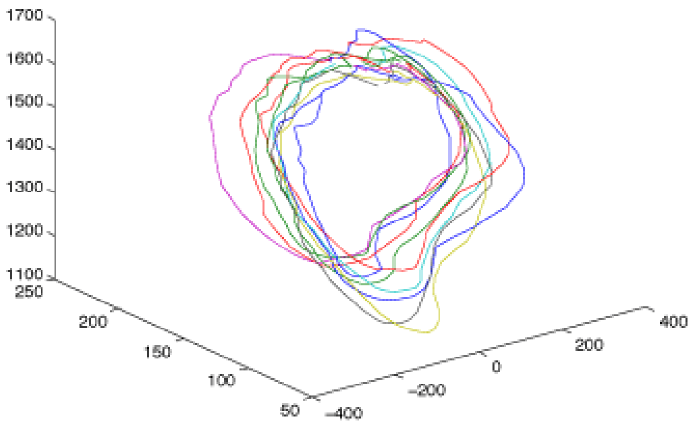

Figure 4.5: Observations of a circular gesture in x, y, z axis which are recorded by a depth camera [?].

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.

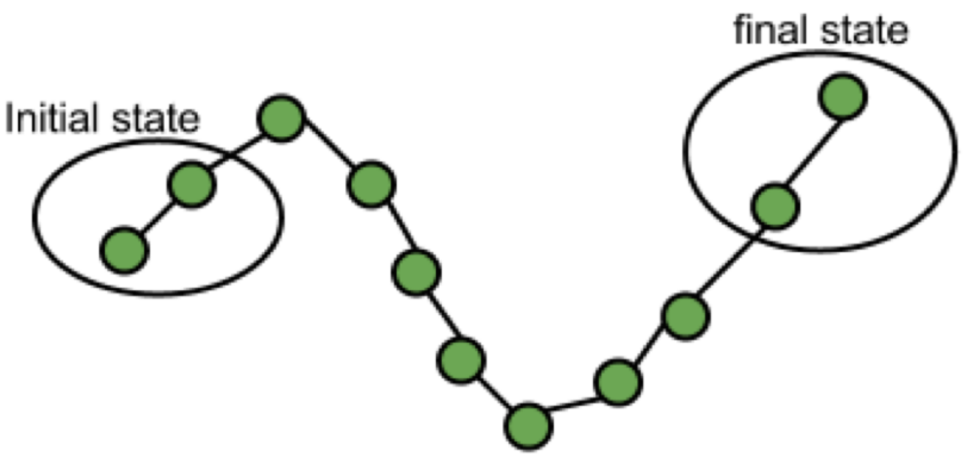

Figure 4.6: The states of Arabic sign language to convey "Hello" [?]. It is consisted of 11 states of Markov models and if there is higher likelihood of the hand position passed through states sequentially from initial to final, then it is recognized as "Hello".

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.

6.1 Accuracy of classifiers, nullRejection, post processing

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.

Bibliography

List of Figures

List of Tables

Abkürzungsverzeichnis

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

```
/**
 * Author: Aravinth Panchadcharam
 * Email: me@aravinth.info
 * Date: 28/12/14.
 * Project: Gesture Recogntion for Human-Robot Interaction
 */

#include <iostream>
#include <sstream>
#include <boost/log/trivial.hpp>
#include <boost/lexical_cast.hpp>
#include <boost/shared_ptr.hpp>
#include "NiTE.h"
#include "gesture_tracker.h"

using boost::property_tree::ptree;
using boost::property_tree::write_json;

/**
 *
 * Constructor
 *
 */
gesture_tracker::gesture_tracker(udp_server *server){
    server_ = server;

    // Initialize the variables here because compiling on virtual nao forbides the
       initialization in the header file
    leftHand = 0;
    rightHand = 0;
    lastLostHand = 0;
    handsSize = 0;
```

```
}


/**
 *
 * Initializes NiTE and takes available OpenNI device
 * NiTE::initialize() takes OpenNI deviceId as argument, if there ar many
 * Wave and Click Gestures are initiated
 *
 */
void gesture_tracker::init_nite(){
    niteRc = nite::NiTE::initialize();
    if (niteRc != nite::STATUS_OK)
    {
        BOOST_LOG_TRIVIAL(error) << "NiTE␣initialization␣failed" ;
    }
    else
    {
        BOOST_LOG_TRIVIAL(info) << "NiTE␣initialized" ;
    }

    // Check for OpenNI device and start hand tracker.
    niteRc = handTracker.create();
    if (niteRc != nite::STATUS_OK)
    {
        BOOST_LOG_TRIVIAL(error) << "Couldn't␣create␣hand␣tracker" ;
    }
    else
    {
        BOOST_LOG_TRIVIAL(info) << "Hand␣tracker␣created" ;
    }

//    handTracker.setSmoothingFactor(0.1);
    handTracker.startGestureDetection(nite::GESTURE_WAVE);
    handTracker.startGestureDetection(nite::GESTURE_CLICK);
    BOOST_LOG_TRIVIAL(info) << "Wave␣your␣hand␣to␣start␣the␣hand␣tracking" ;
}


/**
 *
 * Serializes gesture data with position of hand at which gesture was detected
 * Send it to the connected client
 *
 */
void gesture_tracker::send_gesture(const nite::GestureData& gesture){

    std::string gestureJson;
    if(gesture.getType() == 0){
        gestureJson = "{\"GESTURE\":\"WAVE\"}";
    }else if (gesture.getType() == 1){
        gestureJson = "{\"GESTURE\":\"CLICK\"}";
    }

    boost::shared_ptr<std::string> message(new std::string(gestureJson));
    server_->send(message);
}


void gesture_tracker::send_info(std::string info){

    std::stringstream infoBuffer;
    infoBuffer << "{\"INFO\":\"" << info << "\"}";

    boost::shared_ptr<std::string> message(new std::string(infoBuffer.str()));
    server_->send(message);
}



/**
 *
 * Serializes hand data with position and hand id
 * Send it to the connected client
 *
 */
ptree gesture_tracker::parseToJSON(const nite::HandData& hand){

    // keys and values for json array
    ptree joint_array, xAxis, yAxis, zAxis;

    xAxis.put("", hand.getPosition().x);
    yAxis.put("", hand.getPosition().y);
    zAxis.put("", hand.getPosition().z);

    // Make an array of hand joint for x,y,z axes
    joint_array.push_back(std::make_pair("", xAxis));
    joint_array.push_back(std::make_pair("", yAxis));
    joint_array.push_back(std::make_pair("", zAxis));

    return joint_array;
}


/**
 *
 * Serializes hand data with position and hand id
 * Send it to the connected client
 *
 */
void gesture_tracker::send_hand(const nite::HandData& hand){

    ptree handJson;
    std::string handName = getHandName(hand.getId());

    if(!handName.empty())
    {
        // Parse it json array and add to object
        handJson.add_child(handName, parseToJSON(hand));

        // Stringify the ptree
        std::ostringstream hand_buffer;
        write_json (hand_buffer, handJson, false);
        boost::shared_ptr<std::string> message(new std::string( hand_buffer.str()));

        server_->send(message);
    }
}


/**
 *
 * Serializes hand data with position and hand id
 * Send it to the connected client
 *
 */
void gesture_tracker::send_hand(const nite::HandData& hand1, const nite::HandData& ↩
    hand2){

    ptree handJson;
    std::string handName1 = getHandName(hand1.getId());
    std::string handName2 = getHandName(hand2.getId());

    if(!handName1.empty() && !handName2.empty())
    {
        // Parse it json array and add to object
        handJson.add_child(handName1, parseToJSON(hand1));
        handJson.add_child(handName2, parseToJSON(hand2));

        // Stringify the ptree
        std::ostringstream hand_buffer;
        write_json (hand_buffer, handJson, false);
        boost::shared_ptr<std::string> message(new std::string( hand_buffer.str()));

        server_->send(message);
    }
}


/**
 *
 *
 * Finds the hand name based on given handId
 *
 */

std::string gesture_tracker::getHandName(int handId){
    std::string handName;

    if(handId == leftHand){
        handName = "LEFT";
    }
    else if(handId == rightHand){
        handName = "RIGHT";
    }

    return handName;
}


/**
 *
 * Starts Gesture recognition and Hand tracking based on the position of Hand found by WAVE gesture
 * It tracks it till there is a keyboard ESC Hit and stops
 *
 */
void gesture_tracker::track_gestures(){

    nite::HandTrackerFrameRef handTrackerFrame;

    while (!utils::wasKeyboardHit())
    {
        niteRc = handTracker.readFrame(&handTrackerFrame);
        if (niteRc != nite::STATUS_OK)
        {
            BOOST_LOG_TRIVIAL(error) << "Get␣next␣frame␣failed";
            continue;
        }

        const nite::Array<nite::GestureData>& gestures = handTrackerFrame.getGestures();
        for (int i = 0; i < gestures.getSize(); ++i)
        {
            if (gestures[i].isComplete())
            {
                send_gesture(gestures[i]);
                nite::HandId newId;

                // Dont track more than 2 hands. handTrackerFrame.getHands().getSize() sometime goes to 3 even though
                // there are 2 active hands
                if(gestures[i].getType() == 0){
                    handTracker.startHandTracking(gestures[i].getCurrentPosition(), &newId);
                }
            }
        }

        const nite::Array<nite::HandData>& hands = handTrackerFrame.getHands();

        // hands.getSize() gives the number of active hands, but handId increases
        // hands.getSize() = 0 and goes to hands.getSize() = 2, when there is a new hand detected
        // hands.getSize() = 0, this happens for a fraction of second when tacking is troubled
        for (int i = 0; i < hands.getSize(); ++i)
        {
            // Get Hand data
            const nite::HandData& hand = hands[i];

            // If hand is lost, update the losthandid
            if(!hand.isTracking())
            {
                lastLostHand = hand.getId();
                BOOST_LOG_TRIVIAL(info) << getHandName(hand.getId()) << " Hand with id " << hand.getId() << " is Lost";

                // When there is no active hands, reset all the values
                // Last active hand
                if(hands.getSize() == 1){
                    leftHand = 0;
                    rightHand = 0;
                    lastLostHand = 0;
                    handsSize = 0;

                    send_info("Both hands are lost");
                }
                else
                {
                    send_info( getHandName(hand.getId()) + " Hand is lost");
                }
            }

            // If new hand is found
            if(hand.isNew()){
                BOOST_LOG_TRIVIAL(info) << "Found new hand with id " << hand.getId();

                handsSize++;

                // Check if it is a hand for the first time or second time
                if(handsSize == 1 && lastLostHand == 0){
                    BOOST_LOG_TRIVIAL(debug) << "First hand is found";
                    rightHand = hand.getId();
                }
                else if (handsSize == 2 && lastLostHand == 0){
                    BOOST_LOG_TRIVIAL(debug) << "Second hand is found";
                    leftHand = hand.getId();
                }
                // If a hand was lost and a hand is active, then update the appropriate id to left or right hand
                else if(handsSize > 2 && lastLostHand > 0){
                    if(lastLostHand == leftHand){
                        leftHand = hand.getId();
                    }else if(lastLostHand == rightHand){
                        rightHand = hand.getId();
                    }
                }

                send_info( getHandName(hand.getId()) + " Hand is new");
            }

            if(hand.isTouchingFov()){
                send_info( getHandName(hand.getId()) + " Hand is at FOV");
            }
```

```

        }

        if(hands.getSize() == 2 && hands[0].isTracking() && !hands[0].isNew() && hands[1].isTracking() && !hands[1].isNew()){
            send_hand(hands[0], hands[1]);
        }
        else if(hands.getSize() == 1 && hands[0].isTracking() && !hands[0].isNew()){
            send_hand(hands[0]);
        }

    }

    nite::NiTE::shutdown();
}


/**
 *
 * Called by the main thread and Boost ioService keeps it in the loop
 *
 */
void gesture_tracker::run(){
    init_nite();
    track_gestures();
}
```