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 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.

# Bibliography

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| **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) |

# Attachments

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;
```

```
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 by WAVE gesture
204  * It tracks it till there is a keyboard ESC Hit and stops
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         for (int i = 0; i < gestures.getSize(); ++i)
222         {
223             if (gestures[i].isComplete())
224             {
225                 send_gesture(gestures[i]);
226                 nite::HandId newId;
227
228                 // Dont track more than 2 hands. handTrackerFrame.getHands().getSize() sometime goes to 3 even though
229                 // there are 2 active hands
230                 if(gestures[i].getType() == 0){
231                     handTracker.startHandTracking(gestures[i].getCurrentPosition(), &newId);
232                 }
233             }
234         }
235
236         const nite::Array<nite::HandData>& hands = handTrackerFrame.getHands();
237
238         // hands.getSize() gives the number of active hands, but handId increases
239         // hands.getSize() = 0 and goes to hands.getSize() = 2, when there is a new hand detected
240         // hands.getSize() = 0, this happens for a fraction of second when tacking is troubled
241         for (int i = 0; i < hands.getSize(); ++i)
242         {
243             // Get Hand data
244             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 }
```