DEVELOPMENT AND EVALUATION OF A HAND GESTURE BASED CONTROL CONCEPT FOR A NATURAL AND ROBUST HUMAN COMPUTER INTERACTION

BACHELOR THESIS
submitted by
Florian Marchner
born on: 25.09.1990

Bürgermeister-Schlederer-Strasse 17
85567 Grafing bei München
0176/47768518

NEUROSCIENTIFIC SYSTEM THEORY
Technische Universität München
Prof. Dr Jörg Conradt

Supervisor: Dipl.-Inf. Nicolai Waniek
Start: 01.04.2014
Final Submission: 18.08.2014
Statement of Affirmation

I hereby declare that the bachelor thesis submitted was in all parts exclusively prepared on my own, and that other resources or other means (including electronic media and online sources), than those explicitly referred to, have not been utilized.

All implemented fragments of text, employed in a literal and/or analogous manner, have been marked as such.

Place / Date

Name
Abstract

This thesis has the goal to provide gestures that are both intuitive and provide good technical performance for controlling a robot arm in an industrial environment. For this, a case study has been conducted that asked 14 probands for gestures to handle certain tasks which are present in typical scenarios of that area. The requirement for these gestures was to be as intuitive as possible without taking their technical implementation into account. This study provided a set of gestures for each task that seem to be intuitive for most users.

The second part examines the realization in terms of implementing these gestures. Herefore, the ThreeGear System Software Development Kit has been used for tracking both hands and all the fingers in order to get their position. This made it possible to develop a system that simulated the following tasks:

1. GUI-Interaction: Moving a two dimensional object
2. GUI-Interaction: Zoom Gesture for a two dimensional object
3. Creating three dimensional objects: Creating a Cylinder
4. Pick and Place of a three dimensional object

Within this system, the gestures that were created by most probands during the study for each task have been implemented. The only exception is the task to create a cylinder. Here the gesture that was created second most has been implemented. With this system, the conclusion was made that all the implemented gestures are well suited for handling these tasks. It also shows that the gestures that were provided most by the probands all seem to be suitable for handling each task.
Contents

1 Introduction 5
  1.1 Problem Statement 5
  1.2 Related Work 6
  1.3 Structure of the Thesis 6

2 Basic Information and Definitions 9
  2.1 Types of Gestures 9
  2.2 Abstract Concepts 9

3 Case Study 11
  3.1 Structure and Realization of the Case Study 11
  3.2 Presentation of the results 12
    3.2.1 Amount of different Gesture Classes 12
    3.2.2 Time for Consideration 12
    3.2.3 Type of Fingers used 13
  3.3 Presentation of the resulting gestures 15

4 Implementation of selected gestures in C++ using the ThreeGear Systems SDK 19
  4.1 Overview of the ThreeGear Systems Software Development Kit 20
  4.2 Implementation of selected gestures 23
    4.2.1 GUI-Interaction: Moving a two dimensional object 24
    4.2.2 GUI-Interaction: Zoom Gesture to change the size of a two dimensional object 26
    4.2.3 Creating three dimensional objects: Creating a cylinder 28
    4.2.4 Pick and Place of a three dimensional object 30
  4.3 Evaluation of the developed system 32

5 Conclusion 35

List of Figures 37

Bibliography 39
Appendices

A  Question Sheet used for the Case Study  

B  Setting up the system
Chapter 1

Introduction

1.1 Problem Statement

Using gestures in order to control certain electronical devices has gained rising attention. Many companies that produce devices used in our day to day life, e.g. cars and televisions, have changed their approach on how to control these devices. They realized the fact that using the hands directly in order to communicate with electronical devices seems to be very intuitive in many cases.

An additional field of application for a gestural based control is an industrial environment. With the uprising replacement of humans by robots, the question remains which is the best device to control these robots. For this, a gestural based user interface could be the answer. The current devices to control these robots do not seem to be intuitive for every user. Humans are accustomed to interact directly with their environment by using their hands.

A first step in this direction is already done. The german company KUKA developed a robot arm that allows the user to “teach” it what to do \cite{Gmb14}. However, for this to do, the user must actually touch the robot and push it to the right places. Therefore, the user must always be able to reach the robot.

A gestural user interface would make it possible to interact with robots that are not in the same locations as the user is. Here, the requirements on human safety issues could cease to apply since the robot is not able to harm the user.

Herefore, a lot of effort is put on finding robust algorithms for tracking hands and fingers. But the work to find the best gestures in terms of being intuitive for most users and, at the same time, providing a good performance when implemented is still left open. This thesis has the goal on doing a first step in this direction by providing gestures for selected tasks.

For this, a case study has been conducted that has the goal to find gestures for different scenarios. These gestures had the requirement on being as natural and intuitive as possible for the user. The second part of this work consists of the examination of these gestures in terms of their technical implementation. The following evaluation
is to provide gestures, that fulfill both criteria: being intuitive and providing good technical performance.

1.2 Related Work

A lot of work is going on in finding the best algorithms for tracking hands and fingers using cameras like the Kinect for XBox from Microsoft. However, the question which gestures are the most intuitive and self-explanatory ones remains almost unanswered. In the following, two examples are shown, one for finding an algorithm to track individual fingers using the Microsoft Kinect Camera and one for developing a gestural communication lexicon for human robot collaboration.

[Cer] uses the Microsoft Kinect Camera to develop an algorithm for tracking the fingertips and the palm of each hand by using the depth data given by the Kinect Camera and some geometric properties. He showed that the Kinect Camera provides good results that make it possible to track individual fingers.

[BG13] conducted an experiment with workers in a car factory. Here, they observed the probands during their work and came up with a set of communication terms and gestures. Their analysis showed that all these gestures are well suited for an implementation on a robot arm, but the interpretation of these gestures are dependent on task context. In conclusion, this work showed the importance of human observation when it comes to implementing human made tasks on a robot control.

Both these examples explain the approach of this thesis. [BG13] shows the importance of human observation for such a task which caused the realization of a case study in the first step of this thesis. [Cer] illustrates the advantages of using the Kinect camera. This fact led to the approach used in chapter 4. Nevertheless, both these examples show as well that work on finding the best gestures remains open. This is exactly what this thesis is about. Its purpose is to provide gestures that are intuitive for most users and, at the same time, provide a good performance when implemented.

1.3 Structure of the Thesis

The structure of this bachelor thesis is as followed. Chapter 2 consists of definitions and explanations of terms that are used throughout this thesis. Here, the focus lies on different types of gestures and terms that are used within the ThreeGear SDK. Chapter 3 gives an overview of the case study that has been conducted in the course of this work. It shows the structure of this study, its purpose and a presentation of the results. Chapter 4 consists of the implementation of a system in order to test some of the
gestures gained by the case study. First, there will be an explanation on the functionality of the ThreeGear System SDK, which has been used to track individual fingers. After that, a detailed description on how these gestures have been implemented using this SDK is provided. An evaluation of the developed system completes this chapter.

Appendix A consists of the question sheet, which has been used for the case study. Appendix B gives a detailed description on how to set up the developed system using Microsoft Visual 2013.
Chapter 2

Basic Information and Definitions

In the following, there will be some definitions made which are used throughout this thesis. First, different type of gestures will be explained. After that, some terms that are used will be explained.

2.1 Types of Gestures

Pointing Gesture *Pointing Gestures* are all hand gestures that use one or more than one finger or the whole hand pointing towards the object which is on focus right now. All fingers, or the whole hand, remain in their starting position during the gesture.

Pinch Gesture *Pinch Gestures* are all hand gestures, in which the thumb and the index finger of one hand are moved towards each other. The starting position is some kind of arc position. Then, the index finger and the thumb are moved towards each other until they touch each others tips. Moving these two fingers away from each other again ends this gesture.

2.2 Abstract Concepts

intuitive [Weh05] explains *intuitive* as “(of computer software, etc.) easy to understand and to use”.

In this thesis, it is used as a term for describing that people do not have to think about the action that is required from them. In a concrete context, this means that the user does not have to think about which gesture to use rather than just use the gesture that comes to his mind first.

HandTrackingServer A Windows Batchfile provided by the ThreeGear SDK which interprets and processes the raw depth data given by the camera. Out of this data, it produces *HandTrackingEvents*. 
HandTrackingEvents Created by the \textit{HandTrackingServer}. Consists of the interpretation of the raw depth data from the \textit{HandTrackingServer} in form of strings (text data).

HandTrackingClient Namespace which includes all the classes provided by the ThreeGear System SDK. Additionally includes enumeration types for the degrees of freedom of the hand model, hand type and joint frames for each finger ([Sys14d]).

HandTrackingMessage “Base class for all hand-tracking messages / events” ([Sys14e]). Messages that are repackaged HandTrackingEvents by the HandTrackingClient and sent to the HandTrackingListener via a callback interface.

HandTrackingListener Class that provides methods to catch HandTrackingMessages that are sent via the callback interface. Here, the user can implement his own interpretations of these different types of HandTrackingMessages.
Chapter 3

Case Study

In the course of this work, a case study was conducted in order to create a certain pool of gestures which could be used in several scenarios of an industrial environment. Further on, these gestures were the basis for the implementation, which is described in the latter chapter.
For this study, 14 people were asked, 3 of them were female and 11 of them were male. All probands were employees at fortiss GmbH and all of them had a background knowledge in the fields of electrical engineering and information technology.

3.1 Structure and Realization of the Case Study

All 14 probands were asked directly and individually while taking part in this case study. For this, a question sheet has been created beforehand. It had 10 questions on it that were about typical scenarios which could take place in an industrial environment, especially in a robot-based environment. During the study, the proband sat at a table vis-à-vis the questioner at all time. At one side of table a camera has been placed which recorded each interview.

Each interview started with an explanatory part in which the proband was first introduced into the topic of this study. He then was asked to always explain what he is thinking as clearly as possible and ask questions when something is not understandable for him. In the end, he was asked to only design Single Hand Gestures where possible. This was due to ensure a reasonable complexity when trying to implement these gestures. However, the proband was allowed to create Double Hand Gestures if that was the best option for him. The request to create intuitive gestures had the highest priority.

The question sheet used for the case study is shown in Appendix A. The code in the brackets behind each scenario will be used in the latter section “Presentation of the results”.
3.2 Presentation of the results

For the analysis of this study, every recorded interview was viewed individually and the focus had been laid on the following data:

- Exact description of every gesture
- The consideration time for creating a gesture (measured from the end of asking the question until the first thought was displayed)
- The amount and type of fingers used for each gesture

After that, all gestures were classified within one scenario in a way that similar gestures were put together in one class. For example, if one proband described a gesture to select an object by pointing towards it with the index finger and another proband described a gesture for the same task by pointing towards it with the middlefinger, these two gestures were put together in one class and seen as the same gesture. This was done in order to reduce complexity.

3.2.1 Amount of different Gesture Classes

Figure 3.1 shows how many different gestures were created for each scenario. This gives a first impression of the variety of gestures for each scenario. Tasks that had many different described gestures do not seem to be manageable by one explicit intuitive gesture. An example for that is the task to create a virtual cuboid (C3D (a)). Here, the evaluation states that this task cannot be described by one explicit gesture which is intuitive for every user.

In contrast, tasks that had only a few different described gestures do seem to be manageable by one explicit intuitive gesture. An example for that is the task to pick up an object and place it on another spot (PaP).

3.2.2 Time for Consideration

Figure 3.2 shows how much time all probands spent in average to create a gesture for each task. This gives an impression on how complex each task has been seen by the probands. The first three tasks (all GUI-Interaction) and the last one (Pick and Place) seem to be manageable by using hand gestures due to their low consideration times. However, it is assumed that the gestures for the first three tasks are all influenced by known gestures that are used for smartphones or tablets which could be another explanation for the rather low consideration time.

In contrast, the tasks to create a virtual cuboid or a cylinder (C3D (a) and (b)) and the task to create a path for a welding robot (PP (a)) seem to be complex and hard to manage by using only hand gestures due to their rather high time for consideration. Here, the user may need some additional input device in order to
3.2. PRESENTATION OF THE RESULTS

3.2. PRESENTATION OF THE RESULTS

Figure 3.1: Amount of different gestures for each scenario

make the gesture *intuitive* for him.

Figure 3.2 also gives a measurement of *intuition* for each task. Low times for consideration indicate that the described gestures for these tasks seem to be *intuitive*. This means that these mentioned gestures are suitable for managing those tasks. Contrary, high times for consideration indicate gestures which cannot be described as intuitive. Therefore, these gestures are not suitable to manage a task in a way that all users feel comfortable with.

3.2.3 Type of Fingers used

Figure 3.3 shows how many different gestures used a certain finger within one task. This information can be used when it comes to the implementation of finger tracking algorithms. Besides the first two tasks (GUI (a) and (b)), the index always has been used most. This shows that the user is most comfortable by using his index for gestures than any other finger. The focus for implementing finger tracking algorithms should therefore lie on tracking the index correctly.

The thumb also seems to be a finger that most users are comfortable with using it. However, when using the thumb, the user always used the index finger as well. Most
of the gestures that were realised by using both, the thumb and index, were *Pinch Gestures*. 
3.3 Presentation of the resulting gestures

In the following, two gesture classes which were created most for each task will be described. If there are gestures which have been described by the same amount of people, the ones that have a lower time for consideration are preferred, since these are more likely to be intuitive.

For all gestures hold, unless described otherwise, that selecting is done by keeping the hand at the same place for a defined amount of time while staying in the mentioned gesture.

1. GUI-Interaction: Moving a two dimensional object

   (a) The hand is above the object in a pointing position with one finger (e.g. the index finger) pointing towards it or touching it. The window is moved if the whole hand is moved while staying in this pointing position. This gesture was described by 6 probands with an average consideration time of 3.8 seconds.

   (b) The outstretched palm is placed above the window with each finger touching its neighbours. The object is moved accordingly to the movement of the palm. This gesture was described by 4 probands with an average consideration time of 4.25 seconds.
2. GUI-Interaction: Changing the orientation of a two dimensional object

(a) Two fingers are placed above the object pointing towards it. The rotation of both these fingers defines the rotation of the object. This gesture was described by 6 probands with an average consideration time of 3.5 seconds.

(b) The hand is placed above the object with all 5 fingers pointing towards it. By rotating the hand, the object is rotated accordingly. This gesture was described by 4 probands with an average consideration time of 5.8 seconds.

3. GUI-Interaction: Changing the size of a two dimensional object

(a) Both hands with each index finger stretched out are placed above the object. The distance between both fingers defines the scaling factor. This gesture was described by 7 probands with an average consideration time of 3.7 seconds.

(b) The thumb and the index finger of one hand are placed above the object in an arc position. The distance between both fingers defines the scaling factor. This gesture was described by 5 probands with an average consideration time of 4.0 seconds.

4. Creating virtual three dimensional objects: Cuboid

(a) First, the ground plane is defined by defining its 4 corner points. After that, a fifth point is defined which sets the height of the cuboid. This gesture was described by 4 probands with an average consideration time of 6.25 seconds.

(b) First, the system will be told that the users wishes to define a cuboid (for example by pressing a button). After that, two opposite points are defined by drawing a line between these two points using a Pinch Gesture. This gesture was described by 4 probands with an average consideration time of 12.25 seconds.

5. Creating virtual three dimensional objects: Cylinder

(a) The ground plane is drawn by using one hand and its index finger in a pointing position. After that, the height gets defined in the same manner. This gesture was described by 5 probands with an average consideration time of 7.6 seconds.

(b) First, the system is told that the user wishes to define a cylinder (e.g. by pressing a button). Then, the middlepoint of the ground plane is to be defined. After that, a point which then defines the radius of the ground plane is to be defined. Last, a point which defines the height of
the cylinder is to be defined. This gesture was described by 3 probands with an average consideration time of 6.3 seconds.

6. Creating virtual three dimensional objects: Plane

(a) The palm defines the orientation of the plane. This gesture was described by 7 probands with an average consideration time of 5.0 seconds.

(b) By using a Pinch Gesture, 2 connected lines will be drawn that define the orientation of the plane. This gesture was described by 2 probands with an average consideration time of 3.5 seconds.

7. Pathplanning for a Welding Robot: Creating a path

(a) The hand is in a pointing position with the index finger stretched out. While moving the hand in this position, the path is defined through the tip of the index finger. By staying at one point for a short amount of time, a welding point is defined. This gesture was described by 4 probands with an average consideration time of 11.3 seconds.

(b) The hand is in a pointing position with the index finger stretched out. While moving the hand in this position, the path is defined through the tip of the index finger. When the path is defined, the welding points are defined by pointing towards them afterwards. This gesture was described by 3 probands with an average consideration time of 10.7 seconds.

8. Pathplanning for a Welding Robot: Changing a path

(a) By pointing towards a point on the path, this point gets selected and is moved accordingly to the movement of the index fingertip. The system automatically connects the line points that are near the selected point. This gesture was described by 8 probands with an average consideration time of 4.9 seconds.

(b) The starting and ending point of the section which needs to be changed is to be defined. Then, only this section is to be changed. This gesture was described by 3 probands with an average consideration time of 11.3 seconds.

9. Orientation of virtual objects in three dimensional space

(a) The hand is in a position with the index finger and the thumb stretched out. These two fingers define a reference plane. The object’s orientation is changed according to the movement of this defined plane. This gesture was described by 8 probands with an average consideration time of 5.0 seconds.
(b) The thumb, the index finger and the middle finger are in a position in which they define a coordinate system. The hand then is to be hold inside the virtual object. The orientation of the object is changed according to the movement of the hand. This gesture was described by 1 proband with a consideration time of 2 seconds.

10. Pick and Place Task

(a) The object will be gripped by using the thumb and index finger of one hand. By moving the hand, the object will be moved accordingly. This gesture was described by 8 probands with an average consideration time of 3.8 seconds.

(b) The hand is moved to the initial position of the object. By closing the hand at that position, the object gets picked up. The object now gets moved to the desired location by moving the hand in that closed state to that location. Placing the object will be done by opening the hand. This gesture was described by 3 probands with an average consideration time of 5.7 seconds.
Chapter 4

Implementation of selected gestures in C++ using the ThreeGear Systems SDK

During this thesis, a system has been developed in order to test several gestures for their performance and usability. It is implemented in C++ using the ThreeGear Systems Software Development Kit (SDK) and OpenGL for the purpose of a visual display. Figure 4.1 shows the world coordinate system which OpenGL is using ([Ano]).

Figure 4.1: OpenGL world coordinate system
This chapter will show how this was done by first giving a short explanation on how the ThreeGear SDK is working. After that, the focus lies on the actual implementation and testing of several gestures. Last, a short evaluation of this system will show the advantages and problems of this implementation.

4.1 Overview of the ThreeGear Systems Software Development Kit

The ThreeGear SDK makes it able to write hand and finger tracking applications in a user friendly way. It consists of several classes which are shown in a simplified class diagram in figure 4.2. A detailed description of every class can be found at [Sys14a]. All these classes provide a recognition technique for simple gestures like “Hand Pointing with Index” or “Hand Relaxed Open”. For the system developed for this thesis, the focus was laid on the class “PoseMessage” because this class makes it possible to detect hand positions like “Hand Pointing” or “Hand Pinched”.

![HandTrackingMessage](image)

Figure 4.2: Simplified class diagram of the ThreeGear SDK ([Sys14e])

This SDK requires the usage of one of the following depth sensors ([Sys14b]):

- PrimeSense Carmine 1.09
- PrimeSense Carmine 1.08
- Asus Xtion Pro/Live
- Kinect for Windows
- Xbox Kinect
states that this system works best with a “PrimeSense Carmine 1.09”. In this thesis, the Kinect for Xbox was used. The camera then must be mounted above the desk (for this work, it was mounted at about 110cm above the desk).

The actual functionality of the ThreeGear SDK is shown in figure 4.3. The Camera produces the raw depth data of the hands which is sent over to the HandTrackingServer. This is simply a Windows Batch File, which needs to be running at all time while executing a developed application based on this SDK. This HandTrackingServer interprets and processes the raw data given by the camera and produces HandTrackingEvents. These events are basically strings (text data) which are then sent over to the HandTrackingClient, that is running in its own thread, via a callback interface. It waits for events, parses and repackages them as HandTrackingMessages. These messages can be of several types that correspond to the classes shown in figure 4.2. All these messages then get sent to the HandTrackingListener, where the user can implement his own interpretation of these messages.
The developed system for this thesis is based on the example *Draw Skin*\(^1\) given by ThreeGear System. This example consists of a class which only tracks both hands and displays them on the screen via some OpenGL module. It does not have any gesture recognition implemented. The reason for basing the system on this provided example is the fact that this thesis lays the focus on creating gestures and testing them for their performance and usability. Its coverage is not about how to best track individual fingers and display them on the screen.

\(^1\)The original example can be found on the attached CD
4.2 Implementation of selected gestures

The developed system has the following functionalities:

- Move a two dimensional object (momentarily a rectangle) in two dimensional space
- Scale a two dimensional object (momentarily a rectangle)
- Create a three dimensional cylinder
- Pick and Place of a three dimensional object (momentarily a cylinder)

To realize all these tasks, several classes have been developed that make it possible to recognize the corresponding gestures. These classes are explained in the following sections.

By starting the system, it checks if there is a connection to the *HandTrackingServer*. If not, it prints a command line message and terminates the program. If the check for the connection was succesfull, the user can choose between 4 modes:

1. GUI-Interaction: Moving a two dimensional object
2. GUI-Interaction: Zoom Gesture to change the size of a two dimensional object
3. Creating three dimensional objects: Creating a cylinder
4. Pick and Place of a three dimensional object

The reason for implementing this selection mode was due to many probands stating during the case study that they would like to have an option to tell the system what they are going to do next. This way, the system does not have to differ between gestures that are similar and knows exactly what to expect from the user. This makes the user feel safe in his actions and the gesture tracking is less error-prone.

The selection itself happens via typing in the according key. The type of this key then gets saved in a char variable. The selection of the appropriate mode happens by checking the variable for its value once. This means, if the user wants to change the mode, he also has to exit the program and restart it.

The execution of each mode itself takes place inside an event handler that is implemented within the class “DrawSkin”. This event handler collects every sent messages which are then used by its functions. Here, the user can call its defined methods in order to let the system do appropriate actions for each type of message.

An extra method inside the class “DrawSkin” handles all drawings. Here again, the user can integrate his drawing methods so the system takes appropriate actions.
4.2.1 GUI-Interaction: Moving a two dimensional object

The analysis of the case study conducted during this thesis states that the most adequate gesture for handling this task seems to be a pointing gesture using one finger (e.g. the index). This is the reason for the implementation of this task in the way it is described in this section.

The display after selecting the mode “GUI-Interaction: Moving a two dimensional object” is shown in figure 4.4. Here, the grey rectangle represents the two dimensional object and the user is able to select it and move it in every direction in two dimensional space.

For selecting the object, the user must perform a *Pointing Gesture* using his right index. The tip of the index must be in the area of at most 20 centimeter above the table and inside the dimensions of the object. By staying at this position for a short amount of time, the object gets selected and can be moved. For that, the objects midpoint will always be a projection of the index tip’s position to the ground plane. Whenever the user changes the gesture type or the index finger is outside the allowed area, the object is dropped at its current position and can be selected again.

The way of selecting the object is displayed in figure 4.5. Here, the picture (a) shows how the object is successfully picked. In picture (b) and (c), the gesture is right but the index tip is outside the allowed area. In picture (d), the index tip is in the right area but the gesture is incorrect.
4.2. IMPLEMENTATION OF SELECTED GESTURES

Figure 4.5: Selection of the object (a) and failed attempts to select it (b) (c) (d)

The realization of this is implemented within the class “DefineRectangle”. Here, a method has been created that checks whether the index tip is inside the right area and the hand gesture is a *Pointing Gesture*. Herefore, the method uses a present “PoseMessage” and checks if its type is “HAND_POINTING”.

If so, the position is written to an array. This is repeated 20 times. When the array is full, the method checks the difference between the first entry and the last entry. If this difference is less than the defined maximum (this is currently 4.0), a flag is set to 1 meaning the object is selected.

Another method then sets the objects dimensions accordingly to the position of the index tip. These dimensions are used by another method which draws the object at its current position.

While the object is selected, a method that checks the current gesture type and the position of the index tip is running. If one of these is not correct, it sets the flag to 0 meaning the object is not selected anymore. The dimensions get set one last time and the object is drawn at its current position one last time.

The display of moving a two dimensional object is shown in figure 4.6. Here, picture (a) shows the selection of the object using the *Pointing Gesture*. Picture (b) shows the object being moved to another location, picture (c) shows the object being dropped at this location by performing another gesture.
CHAPTER 4. IMPLEMENTATION OF SELECTED GESTURES IN C++ USING THE THREEGEAR SYSTEMS SDK

4.2.2 GUI-Interaction: Zoom Gesture to change the size of a two dimensional object

For handling this task, the analysis of the case study shows that the most adequate gesture is by pointing towards the object with both index fingers and changing the size by moving both hands towards or away from each other. The way of implementing this is shown in this section.

While being in this mode, the user is able to scale this rectangle by performing the above mentioned gesture. Herefore, the class “Scale2D” has been implemented in order to provide methods that handle the scaling and drawing of the object.

Figure 4.7 shows the display of this mode. Picture (a) shows the initial state, which is the same as in the mode “GUI-Interaction: Moving a two dimensional object”. Picture (b) shows the display when the user scales the object to make it smaller, picture (c) shows the display when the user scales the object to make it bigger. In this mode, the middlepoint of the object always stays at the same location.
One method for each index finger checks if it is within the right area (inside the dimensions of the current rectangle and at most 10 centimeter above the ground plane). These two functions are then used by another method that additionally checks if the gesture type of both hands is a Pointing Gesture using the index finger. This is done in the same way as explained in the previous section.

If the object is selected, it can be scaled by the user. Therefore, the distance between the two index tips is calculated by

\[
distance_{i,\text{Left},i,\text{Right}} = \sqrt{\text{distance}_{i,z}^2 + (y_{i,\text{Right}} - y_{i,\text{Left}})^2}
\]

\[
= \sqrt{\left(\sqrt{(x_{i,\text{Right}} - x_{i,\text{Left}})^2 + (z_{i,\text{Right}} - z_{i,\text{Left}})^2} + (y_{i,\text{Right}} - y_{i,\text{Left}})^2\right) + (y_{i,\text{Right}} - y_{i,\text{Left}})^2}
\]

\[
= \sqrt{(x_{i,\text{Right}} - x_{i,\text{Left}})^2 + (z_{i,\text{Right}} - z_{i,\text{Left}})^2 + (y_{i,\text{Right}} - y_{i,\text{Left}})^2}
\]

with

\[
x_{i,\text{Right}}, y_{i,\text{Right}}, z_{i,\text{Right}} \text{ being the x-, y-, and z-coordinates of the right index tip}
\]

\[
x_{i,\text{Left}}, y_{i,\text{Left}}, z_{i,\text{Left}} \text{ being the x-, y-, and z-coordinates of the left index tip}
\]

The initial distance that existed while selecting the object is seen as 100 percent regarding its size. Therefore, the scaling factor for each dimension is set by

\[
\begin{pmatrix}
x_{\text{scale}} \\
z_{\text{scale}}
\end{pmatrix} = \begin{pmatrix}
distance_{i,\text{Left},i,\text{Right},\text{current}} \\
distance_{i,\text{Left},i,\text{Right},\text{initial}}
\end{pmatrix}
\]

(4.2)

with

\[
distance_{i,\text{Left},i,\text{Right},\text{current}} \text{ being the current distance between both index tips}
\]

\[
distance_{i,\text{Left},i,\text{Right},\text{initial}} \text{ being the initial distance between both index tips}
\]

This scaling factor is then multiplied with the original length in both directions (x- and z-direction) which results in the current size of the object. This method executes inside the event handler of the class “DrawSkin”.

The actual drawing of the object is handled by an extra method which uses the current dimensions in order to paint the object.
4.2.3 Creating three dimensional objects: Creating a cylinder

Here, the analysis of the case study and its resulting gestures, shown in section 3.3, state that the most adequate gesture to handle this task would be by drawing the ground plane using a *Pointing Gesture*. After that, the height is defined by drawing a line from the ground plane with the desired length. It has been desisted from implementing this gesture due to many reasons. One would be that the system would need to perform a lot of corrections in order to define a real circle. The implementation of algorithms for making these corrections would exceed the complexity of this thesis.

Instead, the gesture mentioned second most has been implemented. Here, the user first defines the middlepoint of the ground plane, then a second point whose difference to the first point defines the radius of the ground plane. At last, a third point whose difference to the second point defines the height needs to be defined and the cylinder can be created by the system.

![Figure 4.8: The three different steps in order to create the cylinder](image)

These three steps to create the cylinder are shown in figure 4.8. Picture (a) shows the definition of the first point which defines the middlepoint of the groundplane. Picture (b) displays the definition of the second point. The distance between this point and the point shown in picture (a) defines the radius of the cylinder. Picture (c) shows the definition of the last point, which results in the cylinder being drawn instantly. The distance between this point and the point shown in picture (b) defines the height of the cylinder.

The definition of these points is handled in a separate class “DefinePoint”. This class provides inter alia a method in order to set points using a *Pointing Gesture* with the index finger. Herefore, it checks if the gesture is a *Pointing Gesture* using the class “PoseMessage” from the ThreeGear SDK. If so, the position of the index tip
is written to an array twenty times. After that, the first and last entry of the array are being compared. If the difference between these two is less than the maximum (currently 4.0), the point gets defined at the mean value of the two entries.

Another method provides a possibility to display this point at its position by drawing a sphere around the defined position. For this, the OpenGL utility toolkit GLUT is used.

The definition of the radius and the height of the cylinder happens inside one method within the class “DefineCylinder”.

For the radius, the difference in the x-plane is calculated by first comparing the x-coordinate of both points and subtracting the one with the lesser value from the one with the greater value. The same is done for the z-coordinates of both points. The radius is then defined by

\[
\text{radius} = \sqrt{x^2 \cdot z^2}
\]  

(4.3)

with \(x\) being the difference of both x-coordinates and \(z\) being the difference of both z-coordinates.

For the height, the method subtracts the y-coordinate of the second point from the y-coordinate of the third point. This allows the user to define a third point for the height that is geometrically “under” the second point and still getting the cylinder he wanted to define.

The drawing of the cylinder is handled by a separate method inside the class “DefineCylinder” which draws lines from the middlepoint of the ground plane to the points on the circle around this middlepoint with the defined radius. All these points on the circle are defined by

\[
\begin{pmatrix}
    x \\
    y \\
    z
\end{pmatrix} = \begin{pmatrix}
    x_{\text{middlepoint}} + \text{radius} \cdot \cos \phi \\
    y_{\text{middlepoint}} \\
    z_{\text{middlepoint}} + \text{radius} \cdot \sin \phi
\end{pmatrix}
\]  

(4.4)

with \(\phi \in [0; 2\pi]\). The same is done for the top plane.

The cylinder casing is drawn in the same manner meaning that a method draws lines from the points of the ground circle to the according points on the top circle.
### 4.2.4 Pick and Place of a three dimensional object

For this task, the analysis of the case study states that most probands created a gesture which used the index finger and the thumb for a *Pinch Gesture* in order to grip the object. It is moved accordingly to the movement of the hand while performing this gesture and is released when the *Pinch Gesture* is not present anymore. This gesture has been implemented for this system.

After selecting the mode “Pick and Place of a three dimensional object”, the user is able to move a three dimensional object (this is currently represented by a cylinder) in three dimensional space. Therefore, he must pick up the object by performing a *Pinch Gesture* inside the object. While performing this gesture, he can move the object in any direction, its current position will always be the current position of the index tip. By moving the index and thumb away from each other, the object is placed at its current position.

![Figure 4.9: Pick and Place of a three dimensional object](image)

Figure 4.9 shows the display of this mode in three different states. Picture (a) shows the starting position with the object placed in the middle of the screen. Picture (b) shows the *Pinch Gesture* which needs to be performed with the right hand. However, the object is not gripped yet due to the hand being outside the object. Picture (c) shows the object being gripped and moved to another location.

For realizing this, the system needed to recognize the *Pinch Gesture*. For this, a method has been implemented that calculates the distance between the thumb and the index tip. First, the distance in two dimensional space (x-z-plane) is calculated which is then used for calculating the final three dimensional distance as shown in equation 4.5.
4.2. IMPLEMENTATION OF SELECTED GESTURES

\[ distance_{\text{thumb, index}} = \sqrt{\text{distance}_{x,z}^2 + (y_{\text{index}} - y_{\text{thumb}})^2} \]

\[ = \sqrt{(x_{\text{index}} - x_{\text{thumb}})^2 + (z_{\text{index}} - z_{\text{thumb}})^2 + (y_{\text{index}} - y_{\text{thumb}})^2} \]

\[ = \sqrt{(x_{\text{index}} - x_{\text{thumb}})^2 + (z_{\text{index}} - z_{\text{thumb}})^2 + (y_{\text{index}} - y_{\text{thumb}})^2} \]  
\( (4.5) \)

with

\( x_{\text{index}}, y_{\text{index}}, z_{\text{index}} \) being the current \( x-, y-, z \)-coordinate of the index tip

\( x_{\text{thumb}}, y_{\text{thumb}}, z_{\text{thumb}} \) being the current \( x-, y-, z \)-coordinate of the thumb tip

If the distance calculated in equation \( 4.5 \) is less than the maximum value allowed, the system recognizes this as a Pinch Gesture. This maximum value has been set to 15.0 since this gave the most solid recognition of this gesture. This value cannot be set to 0.0, because the recognition of the index tip and the thumb tip done by the ThreeGear SDK always has some jitter in it, which causes the result of the calculation done in equation \( 4.5 \) being always greater than zero.

Another method checks if the index tip is inside the object. Since this is currently a cylinder, it checks if the current index tip is inside a sphere, with the same radius as the cylinder radius, around the current middlepoint of the cylinder.

If both these requirements, the current gesture being a Pinch Gesture and the index tip being inside the object, are fulfilled, the object is selected and its middlepoint is set to the current index tip position as long as these requirements hold true.

Another method handles the drawing, which is basically the same method explained in the section “Creating three dimensional objects: Creating a Cylinder”.

4.3 Evaluation of the developed system

The system has been developed on a computer with the following configurations:

- **Processor**: Intel(R) Core(TM) i5-2500K CPU with 4 × 3.30GHz
- **Random Acces Memory**: 4 × 2GByte
- **Operating System**: Windows 7 Professional 64bit
- **Graphical Card**: AMD Radeon HD 7800 Series
- **Camera**: Kinect for XBox using the PrimeSense Driver
- **Application Development System**: Microsoft Visual Studio Ultimate 2013

The runtime for executing the while-loop once inside the main method with this configuration is in a range of 1-3ms. This is sufficient enough to run the system in a fluent way. Nevertheless, there have been some problems that are shown in this section.

Since this system is based on the ThreeGear SDK for recognizing both hands and tracking them, it also depends on the quality of the finger tracking algorithms that are implemented in this SDK. All in all, one can say that this tracking mechanism is sufficient enough in order to provide a good system performance. However, sometimes the tracking of individual fingers is not accurate enough. This is existent especially in the mode “Pick and Place Task of a three dimensional object”. Here the user must perform a Pinch Gesture in order to select the object. For recognizing this gesture, the distance between the thumb tip and the index tip is calculated. For this distance being in the allowed range, the user has to perform this Pinch Gesture in a unique way.

This behavior is shown in figure 4.10. Both gestures on the left side are Pinch Gestures. However, as seen on the right side, the ThreeGear SDK is not able to track the index and thumb accurate enough if the other three fingers are stretched out. Therefore, the system does not recognize this Pinch Gesture. A sufficient recognition is only possible, if the other three fingers “hide” under the index finger.

However, the recognition is also dependend on the type of camera used. In this work, the Kinect for XBox has been used due to their popularity and good aquisition possibilities. Though, ThreeGear states that the most accurate recognition is given using a PrimeSense Carmine 1.09 ([Sys14b]). It could be that this behaviour is not present anymore when using a different camera, but in the course of this thesis, it was not possible to compare different cameras.

\footnote{inside this loop, all events are polled and processed and all drawings are done}
4.3. EVALUATION OF THE DEVELOPED SYSTEM

Figure 4.10: Performed Pinch Gesture (left) and recognized hand gesture of the ThreeGear SDK (right)

The most accurate recognition of all gestures used in this system seems to be the Pointing Gesture using the index finger. Here, the system tracks the index tip successfully at all times, even if the user does not perform the gesture exactly (e.g. not stretching out the index finger as much as possible or not holding the other 4 fingers as tight as possible). This also holds true for using both hands while performing this gesture.

All in all, one can say that the hand recognition and finger tracking algorithms work best if both hands are used or if both hands are at least within the field of view of the camera. This fact makes the use of this system, which is at most times based on Single Hand Gestures, not as intuitive as possible since the user should always hold both hands inside the field of view in order to get the best results, even if he is only using one hand.

The implementation for defining three dimensional points or selecting the two dimensional object depends on the computing power of the computer on which the system is running. Here, the position of the index tip is written to an array twenty times. This means the operations to do this needs to be done fast enough by the
computer in order to make the system run fluent. However, with the configuration
mentioned above, this is given.
An alternative to this approach is to implement a time measurement. This way, the
system would be independent on the computing power.

Since this system is implemented in C++ using different classes for every mode that
the user can select, it can be easily included in other projects. The only requirement
is the usage of the ThreeGear SDK. This implementation method also makes it easy
to expand this system, e.g. implement new modes or different gestures for the modes
already implemented. With the ThreeGear SDK being open source, the user can
extend or change the system as he desires.

In conclusion, all the gestures that have been used for realizing the different tasks
are suitable for implementing them using the ThreeGear SDK. The only exception
is the *Pinch Gesture* for the Pick and Place task since the user has to perform this
gesture in a unique way in order to get a good recognition. This also shows that
the gestures that were created most by the probands all seem to be suitable for
developing a hand gesture based control.
Chapter 5

Conclusion

This work had the target to create gestures for certain tasks in an industrial environment that were intuitive and had a good performance if implemented. For this, first a case study has been conducted in order to create a pool of gestures which has been used for the implementation. This implementation itself has been done using the ThreeGear SDK for tracking the hands and fingers and getting the position of each finger tip.
For each task that has been realized, always the gesture that has been created by most probands has been used (except for the task “Creating three dimensional objects: Creating a Cylinder”). All implemented gestures had a performance which was sufficient enough in order to provide a good performance, except the Pinch Gesture used for moving a three dimensional object.

The case study provided the interesting fact that for most gestures, only the index finger or the thumb are relevant. This fact can be used for further research in finding algorithms for tracking individual fingers in such a way that the focus should lie on tracking the index and thumb correctly.

This work also showed that the ThreeGear SDK is a user friendly and good way to realize a gesture based control system. The recognition is sufficient most times in order to create gesture based applications.

Due to being implemented using an object orientated programming language, this system can be easily extended in the future. Here, possible tasks could be implementing the other scenarios or implementing different gestures for the ones already created.

Another interesting work that could be done using this system is comparing different types of cameras for their recognition performance. \[\text{Sys14b}\] already gives information about 5 different cameras, but this could be extended also for more and different types.
A practical example of use is the “Kinect-Projector-Working Cell” shown in figure 5.1. Here, the user is standing in front of a cell that consists of a table and a Kinect camera mounted about 1.5m above this table. Vis-à-vis the user is installed a robot arm. A projector mounted 1.5m above the table projects the screen on the table so the user can interact with it.

Here the developed system can be used in order to “show” the robot what to do, e.g. picking up objects and moving them to another location.
List of Figures

3.1 Amount of different gestures for each scenario ........................................... 13
3.2 Average Time for Consideration for each scenario ...................................... 14
3.3 Amount and Type of Fingers used for each Task .......................................... 15

4.1 OpenGL world coordinate system .............................................................. 19
4.2 Simplified class diagram of the ThreeGear SDK ([Sys14e]) .......................... 20
4.3 Functionality of the ThreeGear System SDK ............................................. 21
4.4 Display of the initial state ............................................................................ 24
4.5 Selection of the object (a) and failed attempts to select it (b) (c) (d) .......... 25
4.6 Display of moving a two dimensional object ................................................. 26
4.7 Scaling a two dimensional object .................................................................. 26
4.8 The three different steps in order to create the cylinder ............................... 28
4.9 Pick and Place of a three dimensional object ............................................... 30
4.10 Performed Pinch Gesture (left) and recognized hand gesture of the ThreeGear SDK (right) ................................................................. 33

5.1 Kinect-Projector-Working Cell ([fG]) ......................................................... 36

B.1 Camera setup and frame [Sys14d] ............................................................ 45
Bibliography


[Cer] F. Trapero Cerezo. 3d hand and finger recognition using kinect. Published at Universidad de Granada (UGR), Spain.

[fG] fortiss GmbH.


Appendices
Appendix A

Question Sheet used for the Case Study

1. Gestural User Interface Interaction (GUI)
   For the first scenario, the proband should imagine a desktop screen that is projected on the surface of the table in front of him. This could be done by a simple beamer for example. On this screen is an open window. The proband now had to create gestures for the following tasks:

   (a) How would you move this window from one point to another?
   (b) How would you rotate this window?
   (c) How would you change the size of this window?

2. Creating three-dimensional objects (C3D)
   For the second scenario, the task for the proband was to create virtual three-dimensional objects using hand gestures. This was asked of him by the following questions:

   (a) How would you define a virtual, three-dimensional cuboid?
   (b) How would you define a virtual, three-dimensional cylinder?
   (c) How would you define a virtual separating wall respectively a plane in three-dimensional space?

3. Pathplanning for a Welding Robot (PP)
   For the third scenario, the proband was asked to define and change a path for a welding robot, such that this robot knows which way to go and at which points he needs to weld. Herefore, he had to create gestures for the following questions:

   (a) How would you create a path for a welding robot, such that it knows where to go and where to weld?
(b) How would you change a created path?

4. Orientation of virtual objects in three-dimensional space (O3D)

For the fourth scenario, the proband should imagine a three-dimensional object which is displayed in the area in front of him. Here, the task that was asked of him was the following:

How would you change the orientation of a three-dimensional object which is displayed in the area in front of you?

5. Pick and Place Task (PaP)

In the last scenario, the proband should imagine a scenario in which he wants to tell a gripping robot to pick up an object and move it to another place where it should drop it:

How would you show a gripping robot to pick up an object, move it to another place and drop it there?
Appendix B

Setting up the system

In the following, it will be shown how to set up the developed system using Microsoft Visual Studio Ultimate 2013 and the libraries provided on the attached CD.

Figure B.1: Camera setup and frame [Sys14c]
In figure B.1 it is shown how to set up the camera. It must be mounted above the table looking vertical down on the table. The exact system requirements can be looked up at [Sys14f] by selecting the appropriate operating system under “Installation Part 1”. At the same webpage, it can be looked up how to calibrate the camera by selecting the appropriate operating system under “Installation Part 2”.

For setting up Visual Studio, a new Project needs to be created and all the C++ header files and source code files provided on the attached CD must be included (alternatively, the existing project file on the CD can be used). Then, there must be made some adjustments:

1. Go to the following location: Right Click on the project name inside the project explorer → Properties → C/C++ → Additional Include Paths. Here the following paths must be included:
   - *Drive name of the CD*: \OpenGL\GLUT\glut-3.7.6-bin
   - *Drive name of the CD*: \OpenGL\OpenSceneGraph\include
   - *Drive name of the CD*: \OpenGL\GLFW\include\GLFW

2. Go to the following location: Right Click on the project name inside the project explorer → Properties → Linker → Additional Libraries. Here the following must be included:
   - *Drive name of the CD*: \OpenGL\GLUT\glut-3.7.6-bin
   - *Drive name of the CD*: \OpenGL\GLFW\lib-msvc120

3. Go to the following location: Right Click on the project name inside the project explorer → Properties → Linker → Input → Additional Dependencies. Here the following must be included:
   - opengl32.lib
   - glu32.lib
   - glfw3.lib
   - glut32.lib

When all this is done, the user must start the camera calibration inside the folder *Drive name of the CD*:\NimbleSDK. The calibration can be started by double clicking the Windows-Batchfile called “camerasetup”.

Afterwards, the Windows-Batchfile called “nimble_server”, which represents the *HandTrackingServer*, must be started. Then, the system can be executed.

---

1These files are in the folder DevelopedSystem → Bachelorarbeit_V01
License

This work is licensed under the Creative Commons Attribution 3.0 Germany License. To view a copy of this license, visit [http://creativecommons.org](http://creativecommons.org) or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.