Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Implementation and Evaluation of a Deictic Gesture Interface with the NICO robot

Bachelorthesis
at Research Group Knowledge Technology, WTM
Prof. Dr. Stefan Wermter

Department of Informatics
MIN-Faculty
Universität Hamburg

submitted by
David Biertimpel
Course of study: Human-Computer-Interaction
Matrikelnr.: 6672756
on
13.08.2018

Examiners: Dr. Doreen Jirak
Dr. Matthias Kerzel

Abstract

In everyday interactions, people intuitively reference entities in their environment by pointing at them. These so-called deictic gestures allow directing other people's attention to a desired referent. In the field of Human-Robot-Interaction deictic gesture are of frequent interest as they enable people to apply familiar behavior to shift the robot's focus. However, despite being intuitive, deictic gestures possess an inherent ambiguity. Depending on the perspective and the target's proximity to other entities, the actual target of a deictic gesture may sometimes be difficult to identify, even for a human interaction partner. To this end, this thesis investigates whether we can create a natural deictic gesture interface with the humanoid robot NICO that is capable of recognizing a gesture's target also in ambiguous object constellations. In order to address this task we introduce two approaches: First, we approximate a pointing array from the hand posture of the experimenter. Subsequently, we predict the gesture's target by using a Growing When Required network (GWR). Finally, we create experimental set-ups to evaluate our approaches.

Zusammenfassung

In täglichen Interaktionen referenzieren Menschen intuitiv Entitäten in ihrer Umgebung indem sie auf sie zeigen. Diese so genannten deiktischen Gesten erlauben es, die Aufmerksamkeit anderer Menschen auf einen gewünschten Referenten zu lenken. Im Bereich der Mensch-Roboter-Interaktion sind deiktische Gesten frequentativ von Interesse, da sie es Menschen ermöglichen, mittels vertrautem Verhalten den Fokus eines Roboters zu lenken. Obwohl sie intuitiv sind, besitzen deiktische Gesten eine inhärente Mehrdeutigkeit. Abhängig von der jeweiligen Perspektive und der Nähe des Ziels zu anderen Entitäten, ist das eigentliche Ziel selbst für Menschen manchmal schwer zu identifizieren. Aufgrund dessen untersuchen wir mit dieser Bachelorarbeit, ob wir mit dem humanoiden Roboter NICO eine natürliches Interface für deiktische Gesten kreieren können, welches in der Lage ist, auch bei mehrdeutigen Objektkonstellationen das Ziel einer Geste zu erkennen. Um diese Aufgabe anzugehen stellen wir zwei Ansätze vor: Zuerst approximieren wir einen Zeigestrahl aus der Handpose des Experimentators. Anschlieend sagen wir mittels eines Growing When Required network (GWR) die Zielposition der Geste vorher. Am Ende erstellen wir Versuchsaufbauten, um unsere Ansätze zu evaluieren.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

People in an interaction intuitively point to objects or entities in their environment when they wish to reference them. With these so-called deictic gestures joint attention between people is established, which forms the basis for a communication about the referenced entity [21]. The understanding and performing of deictic gestures is an integral part of people's communication and thus deeply integrated into our natural interaction patterns [38]. This becomes particularly evident when one imagines describing spatial relations only verbally. Especially in scenarios in which people collaborate and solve tasks in a joint interaction, the ability to point is indispensable. This applies across different professions and includes both craftsmen pointing at a tool and computer scientists referencing a line of code.

In scenarios where people collaborate and solve tasks with robots, the integration of deictic gestures is interesting because it allows people to non-verbally shift the robot's focus towards a specific entity in the environment. This provides the basis for further cognitive or motor actions of the robot involving the object, such as recognition or grasping. The integration of this familiar and ubiquitous means of interaction may contribute to a more natural and fruitful human-robot interaction.

In order to enable an even more natural interaction, not only the form of interaction itself but also the robot must reflect this naturalness. Therefore, the interaction with the robot must come as close as possible to human-human interaction. To facilitate that, the robot needs to resemble human-like characteristics, such as an anthropomorphic appearance, an appropriate physical size and human-like sensorimotor capabilities. The humanoid robot platform NICO (Neuro-Inspired COmpanion) [15] is designed to meet those demands, which makes it attractive for natural Human-Robot-Interaction (HRI) scenarios. Therefore, we use NICO as a platform for implementing deictic gestures.

Despite being very intuitive, deictic gestures exhibit an inherent ambiguity. Depending on the personal perspective and proximity of the respective entities, a gesture's goal is sometimes difficult to identify, even for humans. This makes iden-

tifying the target of deictic gestures a challenging task for robots and raises the question of which levels of ambiguities are resolvable.

The above motivates the implementation and evaluation of a natural, intuitive deictic gesture interface with a humanoid robot. Since deictic gestures are deeply interwoven with ambiguity, it is exciting to investigate to what extent a robot can resolve them.

1.1 Deictic Gestures

Deictic gestures are gestures that allow people to reference entities in their environment. The most prominent example of a deictic gesture is the pointing with the index finger, although head nodding, discernibly directing gaze or moving the whole arm in one direction can also be considered a deictic gesture [18]. McNeill [24] distinguishes between concrete and abstract deictic gestures. Concrete deictic gestures are used to coordinate people's attention to specific objects or actions of interest, for instance when people work together ("Give me that tool"), do sports ("throw the ball to that person") or give directions ("Do you see the church over there?"). In this context, deictic gestures aid linguistic accuracy and facilitate "*communicative joint activities*" by establishing a joint focus of attention [21]. In contrast, abstract deictic gestures do not reference specific objects, but visualize abstract thoughts and metaphors ("All across the horizon" - *while pointing from left to right*). In the following chapters of this thesis, we will refer to concrete deictic gestures as deictic gestures.

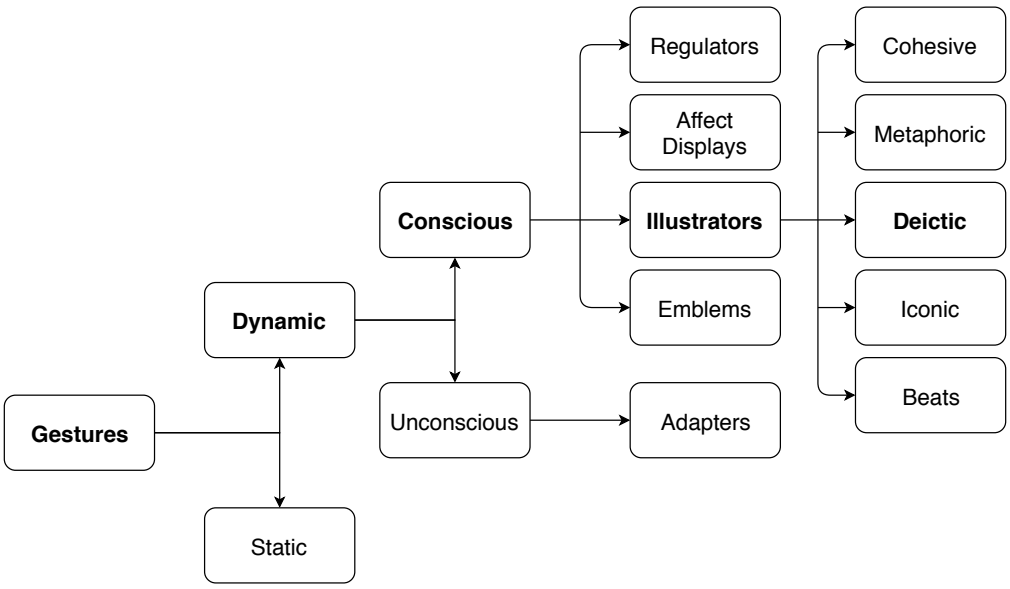


Figure 1.1: Kaâniche's gesture taxonomy [13] classifying deictic gestures.

Despite the great interest in deictic gestures in the research community over the last decades, it seems that no uniform taxonomy including deictic gestures has been established. However, in order to give an overview of where deictic gestures are classified, it is reasonable to pursue a gesture taxonomy. To this end, we consult a gesture taxonomy by Kaâniche [13], which provides a comprehensive picture of the various aspects of deictic gestures and where they differ from other gesture categories. Kaâniche's taxonomy is presented in figure 1.1. According to Kaâniche, deictic gestures fall into the category of so-called *illustrators*, which he derives by combining gesture categories introduced by Ottenheimer and McNeill [28, 24]. *Illustrators*, are consciously executed, dynamic gestures that depict verbal information to support and expand spoken language. The term "dynamic" signifies that the motion aspect of the gesture is decisive for understanding its meaning. Furthermore, in contrast to unconscious gestures (*adapters*), which are performed to facilitate our thinking, fluent speech and lexical access [33], *illustrators* are performed deliberately. They act as a part of language itself as they carry semantic information [28]. *Illustrators* can either stand alone or complement what the communicator is saying (e.g. making a sawing movement while talking about felling a tree).

Deictic gestures are an integral part of human communication and are intuitively used in everyday situations [38]. In fact, people come into contact with deictic gestures early in life, such as in situations where parents talk to their child, while using deictics to draw the child's attention towards a particular object. The use of deictic gestures constitutes a major aspect of infant learning and the child's level of understanding and performing deictic gestures is a valuable indicator of the current stage of his/her communicative development [5]. Moreover, early confrontation with deictic gestures significantly aids the child's language development. [5].

At about 12 months, infants perform their own deictic gestures [20, 1]. At the same time, they start distinguishing between different functionalities of deictic gestures, such as imperative (requesting an object: "*I want this!*") and declarative (showing an object: "*Look at this!*") deictic gestures [9]. During this process of learning deictic gestures, infants are made familiar with the concept of attention and show awareness for the other and the self. Therefore, a key aspect of how infants learn to communicate and develop a language is based on the understanding and performing of deictic gestures.

Later in adulthood, deictic gestures are ubiquitous in any interaction. They are "*almost inevitable*" when communicating about referents (referent: *the entity signified by the gesture*) which are locatable in the current environment [16]. Especially in situations where people collaborate, they are essential for referencing objects, actions and information [24]. The need to direct attention or describe spatial relations through pointings is present over various fields, whether it is programmers pointing to lines of code or mechanics requesting tools while repairing a car. Even in situations where the referent is not locatable, we fall back on abstract deictic gestures to visualize the referencing [16]. Given the existing research on deictics,

it is plausible that those gestures are deeply interwoven in our concept of shared attention and mutual communication.

1.2 Related Work

In recent decades, deictic gestures in a human-robot interaction scenario have been repeatedly discussed in the research community and different approaches have emerged over time.

Canal et al. [6] realized a deictic gestures scenario using the commercially available robotic platform NAO by Aldebaran Robotics[1], which they equipped with a Kinect™ v2 sensor[2]. They extracted the positions of the elbow, hand and hip joints based on the Kinect's depth map and calculated distances and angles between them. A deictic gesture is detected if these values lie within predefined threshold ranges for a certain time period. Then a pointing array is extrapolated using the hand and elbow joints. To model the goal of the deictic gesture, the intersection of the array with the ground surface is calculated. Finally, objects are searched in a predefined area around the intersection using the depth information of the Kinect. If more than one object is detected, a disambiguation process is started in which the NAO verbally asks for feedback. The approach yielded satisfactory results in the scenario and the ambiguity of deictic gestures is addressed. However, the detection of the intention to point is based on a set of fixed rules and thresholds. This makes the interaction less natural, as the rules limit the experimenter's interaction capabilities.

Stiefelhagen et al. [26] employ Hidden Markov Models (HMMs) for recognizing deictic gestures with a humanoid robot platform. For obtaining features, they extract the position of the experimenter's head and hands as well as the orientation of the head. The positions of the hand and head are determined with color and disparity information provided by a stereo camera. The head orientation is measured by a magnetic sensor attached to the experimenter's head. Subsequently, they model the temporal structure of a deictic gesture by dividing it into three phases (*begin*, *hold*, *end*) and training an HMM with the extracted features for each phase. Based on this, the point at which a deictic gesture starts is detected and the angle of the pointing direction is estimated during the *hold* phase. This angle is then compared with ground truth angles, that represent the object positions in the scene. The approach achieves to model the dynamic aspect of deictic gestures. However, the object positions in the scene are only implicitly modeled by the ground truth angles and also do not dynamically change. Therefore, no dynamic gesture target mapping is achieved and ambiguous object constellations are not evaluated. In addition, an external magnet sensor is used to determine head orientation, resulting

[1] `https://www.softbankrobotics.com/emea/en/robots/nao` (accessed 08.08.2018).
[2] `https://developer.microsoft.com/de-de/windows/kinect` (assessed 08.08.2018).

in cumbersome and unnatural interaction.

Park et al. [30] use a cascade of two HMMs with which they aim to recognize deictic gestures independent of their movement intensity. After tracking the face and hands with 3D particle filters, the first stage HMM maps the detected hand to a position on a sphere around the shoulder of the experimenter. Small and large movements with the same direction are projected onto the same point of the sphere. This makes the interface independent of the gestures' scale of motion. The second stage HMM detects the gestures' temporal structure similar to Stiefelhagen's approach [26]. When a deictic gesture is detected, the pointing direction is estimated by calculating the line between the experimenter's head and the processed hand position on the sphere. Then, the angle estimate is compared with annotated ground thruth angle values. The implementation of the sphere, allows people to perform deictic gestures more variable. However, as with Stiefelhagen's approach [26], all object positions in the scene are fixed and are just implicitly represented by the ground truth angles. For this reason, neither ambiguous object constellations are evaluated nor a dynamic gesture target mapping is realized. Furthermore, due to the multiple HMM layers, this approach requires a large training data set and is computationally expensive.

In contrast to the approaches above, Richarz et. al. [36], realize a deictic gestures interface by solely using a monocular low-cost RGB camera. Instead of referencing objects, the experimenter indicates areas on the floor to navigate a mobile robot platform to a specific position. The gesture recognition process is based on a cascade of 14 Multi-Layer Perceptron (MLP) networks. For preprocessing the input images, they detect the experimenter's face and shoulder position by using a method that is based on the Viola & Jones algorithm [39]. This position is used to estimate a region of interest (ROI), that is then further processed by applying empirical measures and different image filtering techniques. The resulting images are used as the input for the MLP cascade. As the input passes through the networks, the radius and angle of the corresponding deictic gesture is determined. Since the position of the experimenter is fixed, both values combined yield the target position. This approach is capable on accurately determining two dimensional target positions without using depth sensors. However, this approach does not focus on a mapping from gesture to target and does not contain a humanoid robot.

1.3 Thesis Goals

This thesis aims to contribute a natural and intuitive deictic gesture interface for the NICO robot. Natural in this context means that the experimenter entering the scene can perform deictic gestures to the arranged objects without major artificial constraints or external devices. For this purpose, we recognize deictic gestures exclusively with NICO's vision capabilities. To enable a fluent interaction, the

interface should be able to give a response in real-time and therefore be computationally efficient.

Regarding the referenceable objects, the interface should be adaptable so that their arrangement, with the exception of positioning in a predefined area, is not further restricted. To achieve that, we implement an object detection that provides us with the positions of the objects in the scene. From here, we emphasize on modeling a dynamic *gesture-target* mapping so that we assign each performed gesture to its corresponding target object.

The overarching goal of this interface is to address the ambiguity of deictic gestures and realize a deictic gesture recognition that is capable of resolving ambiguous object constellations. In order to assess this ability, we use the adaptable object positions to create different levels of ambiguities.

Finally, since NICO's vision capability is a monocular RGB-camera, the question arises whether these goals can be achieved with solely relying on computer vision methods without depth information.

1.4 Thesis Outline

To address our research goals, we organize this thesis as follows:

- In Chapter 2, we provide all information regarding our scenario and implementation. First of all, we motivate our scenario and describe its structure including the object constellations and procedure for taking the recordings. Then we go into the implementation of the deictic gesture interface. We attach particular importance to reproducibility.
- Chapter 3, discusses our evaluation metrics and evaluation set-up for each of our approaches. After describing the set-up, we present and analyze our results.
- Finally, in chapter 4 we discuss our approaches and conclude the thesis.

Chapter 2

Methods

This chapter provides comprehensive information on the context and implementation of the designed deictic gesture interface. First, we motivate and portray our scenario and describe the resulting recordings. Subsequently, we expand on the implementation details of our approach. To this end, we first examine the libraries we use and then describe the implementation of the individual subtasks. These divide themselves into hand detection and tracking, object detection, detection of the pointing intention and the estimation of the goal of a deictic gesture.

2.1 Scenario

In our scenario, the NICO robot (Neuro-Inspired COmpanion) [15] and an experimenter participate in an interaction where the experimenter performs deictic gestures towards NICO. To motivate the use of NICO as a robot platform and the use of deictic gestures in a human-robot interaction scenario, we first take a closer look at the characteristics of NICO as well as at deictic gestures in human interaction. Afterwards, we provide the details of the scenario's construction and portray our created recordings.

2.1.1 Motivation

A significant aspect in the field of Human-Robot-Interaction (HRI) is to create scenarios that enable a natural interaction between humans and robots. Naturalness is achieved above all when the scenario allows people to apply familiar behavior to the interaction. For this purpose, the interaction should be intuitively understandable and thus as close as possible to human-human interaction. As a consequence, it is essential that the robot reflects human interaction patterns. To do so, the robot needs to resemble several human-like characteristics, such as an anthropomorphic

appearance, an appropriate physical size and human-like sensorimotor capabilities. These characteristics lay the foundation for a human-robot interaction in which the appearance and behavior of the robot correspond to the human equivalent, so the human does not feel restricted in its interaction capabilities. The humanoid robot platform NICO is built to meet these requirements and therefore attractive for implementing and evaluating natural HRI scenarios (see figure 2.1).

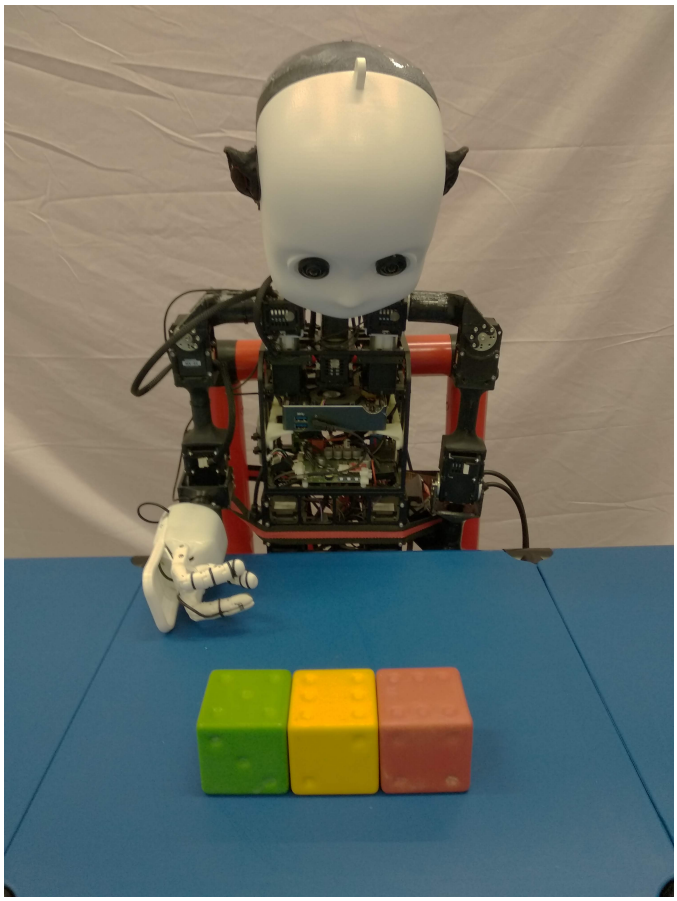


Figure 2.1: The NICO robot (Neuro-Inspired COmpanion) [15] in our scenario.

As described in section 1.1, deictic gestures are deeply interwoven in human communication and play an integral part in our concept of shared attention. They provide an intuitive way to direct attention towards objects of interest and help to establish a joint focus of attention. Therefore, in situations where humans collaborate, deictic gestures are indispensable. Across different professions and fields, people have the need to establish joint attention and reference entities in their environment. In those situations, people intuitively perform deictic gestures. This applies to craftsmen ("*Give me that tool* "), office workers ("*Look at this graph*"), athletes ("*Run over there*") or engineers ("*That gear wheel*"), among others. Deictic gestures are an efficient way to support linguistic accuracy and contribute to a fruitful and more productive interaction. This motivates implementing deictic gestures in an HRI scenario, as it allows people to use this ubiquitous and

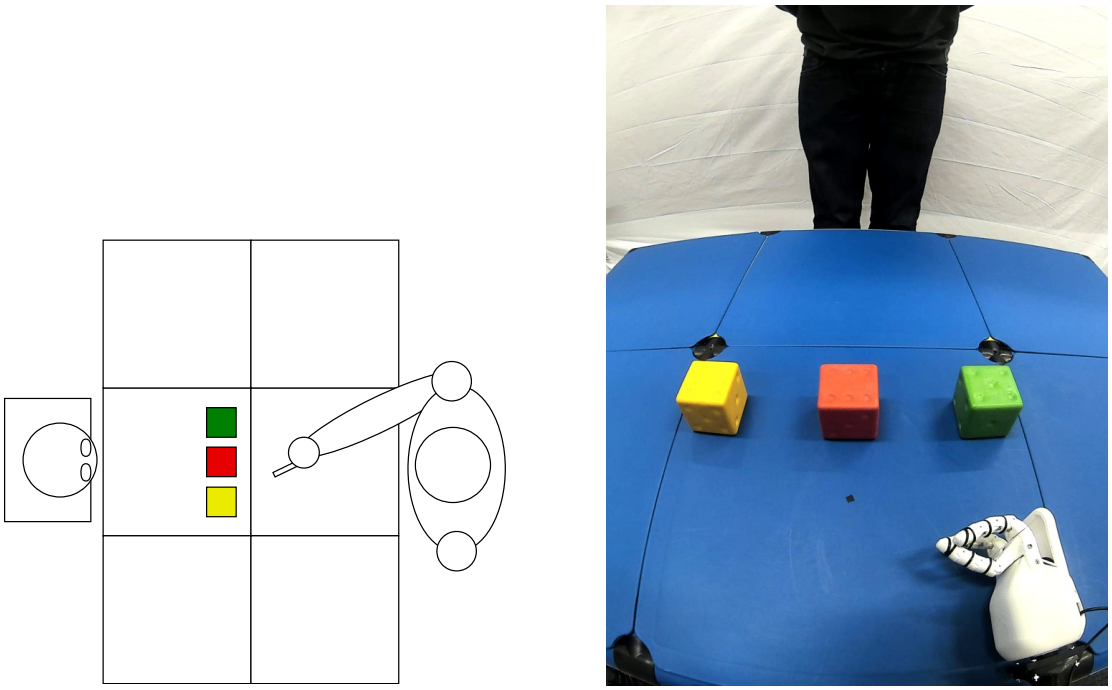
familiar form of interaction when cooperating with robots. In the end, this may contribute to making cooperation between humans and robots more productive and less frustrating for the human counterpart. By providing NICO with this additional interaction capability, we hope to lay the foundation for the evaluation of further scenarios involving deictic gestures.

2.1.2 Scenario Layout and Structure

To accomplish the previously motivated goals, we design a scenario in which an experimenter interacts with the NICO robot through deictic gestures. To signal an object of interest, the experimenter performs a deictic gesture towards objects located in front of NICO. NICO perceives the gesture solely through its vision capability, which is a monocular RGB-camera. Our task is to identify the object referenced by the experimenter from NICO's perspective.

We aim to implement a natural, lightweight and flexible deictic gesture interface. To enable a natural interaction it should be able to give a response in real-time and therefore be computationally efficient. The scenario should also be adaptable so that the number and arrangement of objects can vary. This flexibility in the object constellations allows us to create different levels of ambiguity to evaluate the performance and robustness of our approach. Furthermore, adaptable object positions provide the basis for a dynamic *gesture-target* mapping.

As illustrated in figure 2.2, NICO is placed in front of a table on which we arrange one to three differently colored cubes. These cubes are the referenceable objects in our scenario. The experimenter stands on the other side of the table opposite NICO against a white background. The positions of the NICO robot and the experimenter are fixed during the entire interaction. The reason for this is that the perspective from which a gesture is perceived can have a considerable influence on its interpretation. So, in contrast to our adaptable object positions, we first focus on assessing our approach's performance with gestures executed from one perspective. This is because different viewpoints add significantly more complexity to the task an we want to evaluate the performance of our approach with one viewpoint. As mentioned above, the scene is captured through NICO's vision. From NICO's perspective, the table with the objects, its right hand and the experimenter are visible. The desk has a blue surface and the three objects are red, green and yellow. NICO's hand is discernible in the lower right corner of the recordings (see figure 2.2b), but is not used in the scenario. Since the experimenter's face is not relevant for the scene, only his hands and lower body are captured. We put the experimenter's hands and the referenceable objects in the centre of NICO's field of vision, which facilitates the detection of subtleties in this area. We assume that the ambient conditions, e.g. the lighting, remain unchanged. Furthermore, no other disturbing factors or persons are allowed in the scenario.



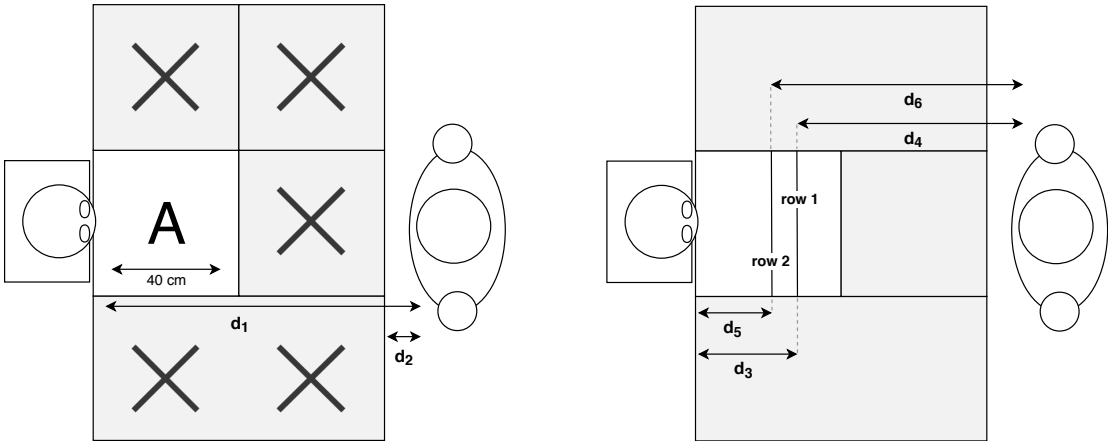
(a) (b)

Figure 2.2: Detailed structure of our scenario. Figure (a) shows the scenario from a bird's eye view with the experimenter to the right and the NICO robot to the left of the table. Figure (b) shows the scenario from the perspective of the NICO robot.

The interaction starts when the objects are positioned on the table. The experimenter performs deictic gestures towards one object at a time by pointing with the right arm. For reasons of simplicity, we define a deictic gesture as pointing with the index finger. The deictic gesture recognition is intended, in conjunction with the object detection, to recognize the object the experimenter targets. In the course of the experiments, the number and arrangement of the objects vary in order to evaluate the robustness of the implementation and to explore the performance of our approach at different levels of ambiguity.

2.1.3 Distances and measurements

Since the positions of the NICO robot and the experimenter are fixed, the distances between these positions can be measured exactly. All distances and specific areas are illustrated in figure 2.3. The distance d_1 between NICO and the experimenter measures $100cm$. Both NICO and the experimenter are directly in front of the table. Since NICO ends with its body directly to the edge of the table, we define its distance to the table as $0cm$. Due to its physiognomy, however, the experimenter leaves a gap of $20cm$ to the table, which cannot be undercut. We call this distance



(a) (b)

Figure 2.3: Fixed distances in our scenario. Figure (a) shows the position of area A, on which the referenceable objects are placed. In addition, the distances d_1 ($100cm$) from experimenter to NICO and d_2 ($20cm$) from experimenter to table are illustrated. Figure (b) describes the distances of the robot and the experimenter to *row 1* and *row 2* on area A. These distances are: $\boldsymbol{d_3}$: robot – *row 1* ($28cm$), $\boldsymbol{d_5}$: robot – *row 2* ($21cm$), $\boldsymbol{d_4}$: experimenter – *row 1* ($65cm$), $\boldsymbol{d_6}$: experimenter – *row 2* ($72cm$).

d_2. The area A in front of NICO where the objects are arranged measures $40cm \times 40cm$. The other table plates around the area A are not used. The differently coloured cubes are all the same size and have a uniform edge length of $5.8cm$. The cubes can be placed on two predefined horizontal lines in area A, which we refer to as *row 1* and *row 2*. The line *row 1* is $28cm$ away from NICO (d_3) and $65cm$ away from the experimenter (d_4). *Row 2* is $21cm$ distant from NICO (d_5) and $72cm$ distant from the experimenter (d_6). From NICO's perspective, *row 1* is located above *row 2*.

With the definition of *row 1* and *row 2*, we deliberately limit the space in which the objects can be arranged in order to keep the number of object permutations manageable. While this does not have much influence on the vision-based pointing estimation, it has a significant impact on the training of the Growing When Required network, which we will discuss in detail later.

2.1.4 Recordings

To implement, test and evaluate our deictic gesture interface, we have to take recordings in the previously defined scenario. As mentioned above, these video sequences are captured with the camera of the NICO robot. The recordings vary in the number and arrangement of objects on *row 1* and *row 2* (see figure 2.3) but are otherwise uniform. In each recording, the experimenter performs exactly

one deictic gesture towards each referenceable object in the scene. Each of the conducted gestures lasts for approximately five seconds.

In the following we provide a detailed description of the recordings. First, we describe the setup of the laboratory environment in which the images are taken. Afterwards, we explain the preprocessing steps applied to the raw video sequences. Finally, we give an overview of the exact object constellations that are recorded.

2.1.4.1 Laboratory Setting

The recordings are taken in a Knowledge Technology Group laboratory at the Department of Computer Science at the University of Hamburg. During the recordings, all ceiling lights are switched on and the shutters at the back window (which was the only window in the room) are half closed. This limits the amount of natural light entering the room and makes it easier to reproduce the lighting conditions. The experimental setup is arranged as described in the scenario. The experimenter stands with his back to the window, so that the white background, in our case a white curtain, separates the window from the scenario.

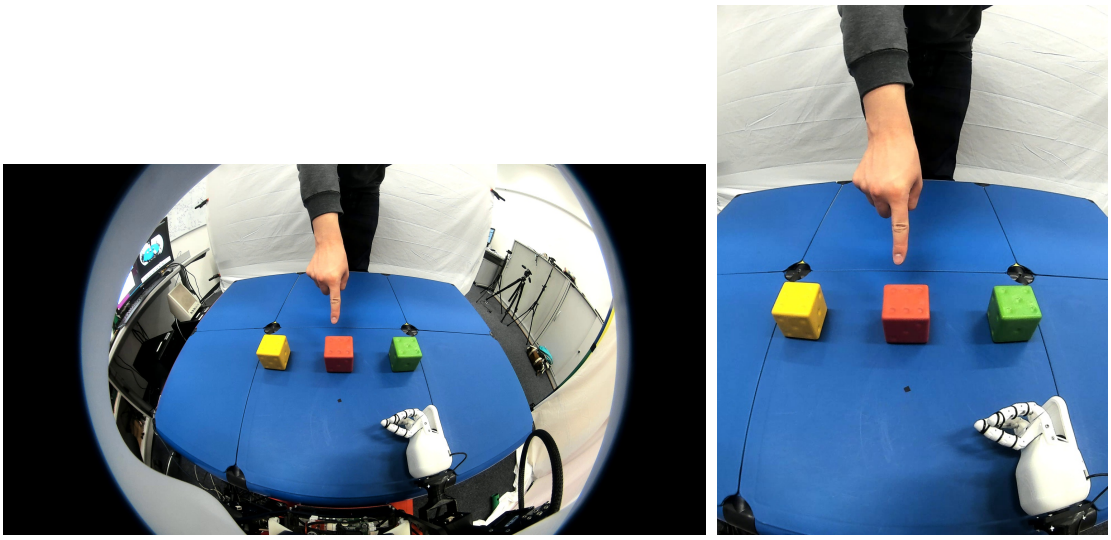
2.1.4.2 Camera Settings and Preprocessing

The raw video sequences are captured through NICO's right face camera, which is a monocular RGB-camera. To allow maximum flexibility in further processing, we initially select a resolution of 4096×2160 pixels. NICO's head is in the lowest position so that it looks straight down capturing the table and lower body of the experimenter.

Due to the fisheye effect in NICO's camera, large parts of the laboratory environment beyond the white background are visible in the raw recordings (see Figure 2.4a). Therefore, after scaling the resolution down to 2048×1080 pixels, we crop the recordings to 700×900 pixels. This is achieved by manually setting the points for cropping the image and then transferring the absolute pixel values in percentages, so that the cropping works regardless of the original resolution. Finally, we crop the images at 0.3125% and 0.6543% of the image width and 0.8796% of the image height. The result of this preprocessing step can be observed in Figure 2.4b.

2.1.4.3 Object Configurations

To implement a robust approach and assess performance at different levels of ambiguity, various recordings with different object constellations are necessary. We therefore capture deictic gestures towards a variety of object arrangements on *row 1* and *row 2*. It is important that the recordings show different levels of ambiguity



(a) (b)

Figure 2.4: Illustration of the effect of our preprocessing on the recordings. Figure (a) shows a frame of a raw video sequence, figure (b) shows the result of our preprocessing applied to the frame.

so that the robustness and performance of our designed interface can be assessed gradually. We record a total of 95 different object constellations with the following distribution of the number of shown objects:

- 16 constellations with one object
- 33 constellations with two objects
- 46 constellations with three objects

We further record 23 object constellations to cover edge cases and allow qualitative reasoning on the general behavior of our interface.

To describe the ambiguity of the object constellations with more than one object, we introduce four ambiguity classes. These classes rank the constellations in their ambiguity, whereby constellations of ambiguity class a_1 are explicit and constellations of class a_4 are highly ambiguous. Note that constellations with less than two objects cannot be ambiguous. The number of recordings by ambiguity class and object count are given in table 2.1.

/	**no ambiguity**	**ambiguity class a_1**	**ambiguity class a_2**	**ambiguity class a_3**	**ambiguity class a_4**	**total**
one object	16	x	x	x	x	16
two object	x	6	10	10	7	33
three object	x	4	12	10	20	46
total	16	10	22	20	27	95

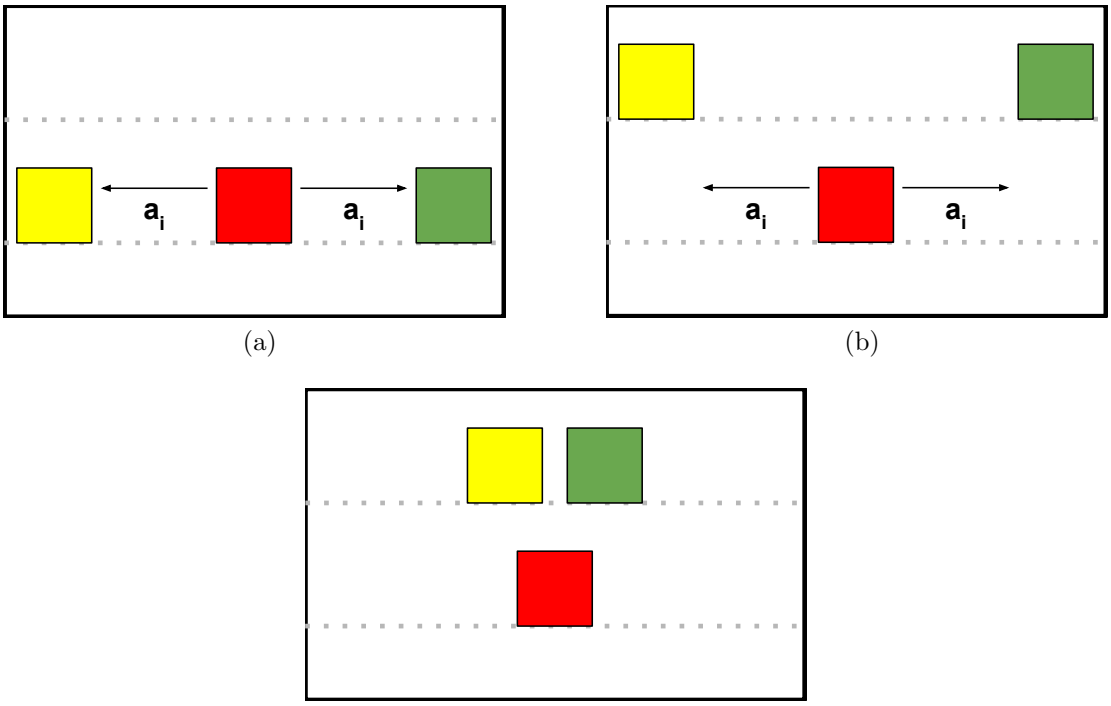
Table 2.1: Number of taken recordings by ambiguity class and object count.

The ambiguity classes represent the horizontal distance between objects on *row 1* or *row 2*. When classifying an object constellation we consider the distances between objects that are placed on the same row as well as between objects on different rows (see figure 2.5). These distances are defined for the different ambiguity classes as following:

- Ambiguity class a_1: 10 cm
- Ambiguity class a_2: 5 cm
- Ambiguity class a_3: 0 cm
- Ambiguity class a_4: overlapping

An object constellation with a defined number of objects and ambiguity class can have different appearances. All objects of the constellation as a whole can be aligned to the left, to the center or to the right of the surface. When aligned to the left (right), the farthest left (right) object can be either $1cm$ or $5cm$ from the edge of the surface. Note that the distances between the objects are defined by the respective ambiguity class.

Moreover, there are various possibilities how the objects can be distributed on *row 1* and *row 2*. For each of the ambiguity classes a_1, a_2 and a_3, there are constellations in which all objects are placed on the same row (see figure 2.5a). In the case of two or more objects, these can be further divided into *row 1* and *row 2* (see Figure 2.5b). For constellations with three objects, this results in either a V-shape (e.g. two objects in *row 1* + one object in *row 2*) or a Λ-shape (e.g. one object in *row 1* + two objects in *row 2*). In ambiguity class a_4, the objects are always divided into *row 1* and *row 2*, so that they overlap by $2cm$. This is illustrated in Figure 2.5c.



(a) (b)

(c)

Figure 2.5: Illustration of the different cases of object arrangements in our recordings. Across all cases a_i describes the distance connected to the respective ambiguity class. Figure (a) shows the case where all objects are placed on the same *row*. Figure (b) shows the case where the objects are divided into *row 1* and *row 2*. These two cases are present in recordings of the ambiguity classes a_1, a_2 and a_3. Figure (c) illustrates an object constellation of a_4, where the objects overlap by $2cm$.

However, there is one exceptional constellation belonging to *ambiguity class 4*, in which two objects are arranged exactly one above the other. These records form the basis for the implementation and evaluation of our deictic gesture interface.

2.2 Implementation

In the following, we address the implementation of our deictic gesture interface. After introducing the libraries we use, we provide an in-depth description of our subtasks. Hereby, we especially emphasize the reproducibility of this work. The subtasks are outlined as follows: Firstly, we focus on the hand detection and tracking, for which we emphasize on a lightweight and intuitive interface and solely use computer vision methods. Secondly, we present our object detection approach, which is employed to dynamically detect the target of the deictic gesture. Thirdly, we will discuss the question of when the experimenter intends to point ("pointing

intention"). Finally, we describe how we estimate the direction and target of a deictic gesture. For the latter task, we present two approaches: A computer vision-based approach that extrapolates a pointing array from the extended index finger and another approach that employs a Growing When Required network, which promises to be stable against noise and offers interesting applications for future research. With these two approaches we realize a dynamic *gesture-target* mapping, which aims to assign each gesture to its target object.

2.2.1 Language and Libraries

For implementing the deictic gesture interface we use *Python 2.7*. The main reason for this is its compatibility with existing projects. Nevertheless, the code would also be compatible with *Python 3*. For all libraries and dependencies we use the Python interface.

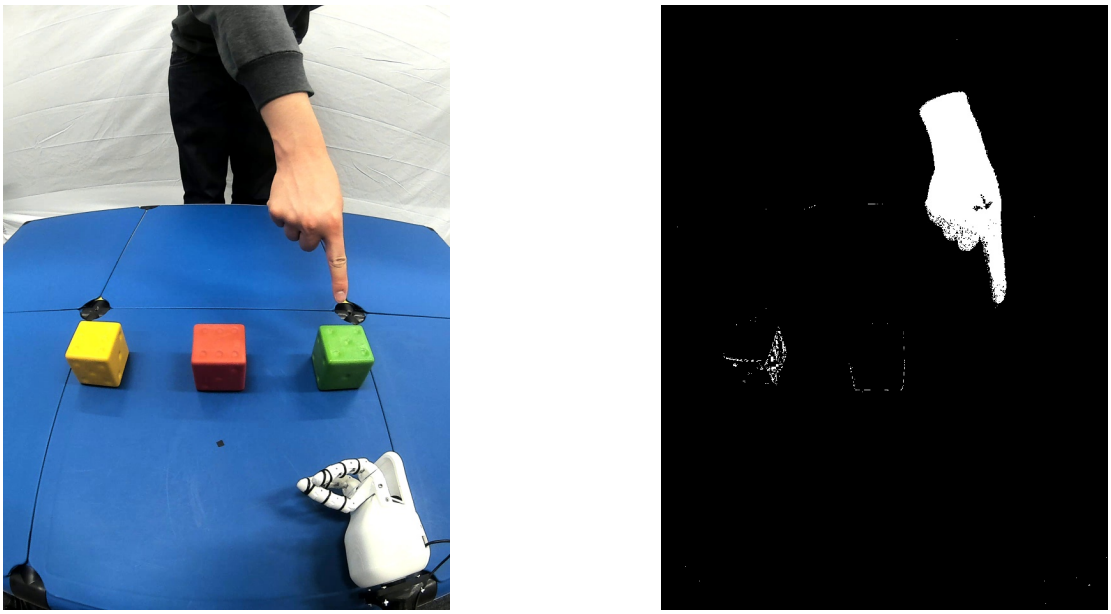
The entire implementation is mainly based on the two libraries, *OpenCV* [3] and *Numpy* [27]. *OpenCV* is an open source computer vision library that includes several functionalities ranging from simple arithmetic operations to image processing functions and complex video analysis methods. The major advantage of *OpenCV* is that it is highly optimized and designed to fuel real-time applications. For our purposes, we mainly use *OpenCVs* image processing module.

Numpy is a core package from the the scientific computing library *SciPy* [11]. Its major functionality is the creation and processing of multidimensional data represented in so-called *Numpy arrays*. Most of *OpenCV's* functionalities and data processing is based on *Numpy* arrays.

Besides *OpenCV* and *Numpy*, we use the data plotting library *Matplotlib* [10] for visualizations. In the process of detecting the pointing intention, we use a function for hierarchical clustering from the general *SciPy* library.

2.2.2 Hand Detection

Hand detection is usually the first step in most vision-based gesture recognition applications. This step is essential, as it forms the basis for further processing, such as tracking and pose estimation. Due to the many degrees of freedom of the hand and the resulting variance in possible hand positions, hand detection is regarded as a complex task. Especially in real-time applications, where the hand detection method is constraint to run efficiently, the task gets even more challenging. Over the years, various techniques have been introduced, that utilize features such as color, shape or motion. Rautaray and Agrawal's survey summarizes the most prominent methods [34].



(a) (b)

Figure 2.6: Result of the skin color segmentation. Figure (a) shows the original recording and figure (b) shows the binary image resulting from our skin color segmentation.

2.2.2.1 Skin Color Segmentation

Skin color segmentation is the process of extracting skin-colored areas of an image by distinguishing between skin color and non-skin color pixel values. Due to its computational efficiency, robustness against rotation, partial occlusion and scaling, skin color segmentation is a popular approach for detecting hands in computer vision scenarios. Furthermore, skin color information conveniently complements other detection approaches such as depth, shape or motion based methods. Therefore, skin color segmentation is often used to aid the accuracy of other detection methods. The principle behind skin color based segmentation lies in the fact that the distribution of skin color clusters in a distinct area of the color space and overlaps only moderately with the distributions of non-skin-colored pixels [12]. For that reason, skin color can be segmented with reasonable accuracy, even despite varying skin tones across different ethnic groups. The process of skin color segmentation leads to a binary image, where the white pixels are classified as skin color and the black parts as non-skin color, respectively. Figure 2.6 shows the result of such a segmentation.

However, due to only relying on color information, skin color based hand detection is highly susceptible to other skin color like objects in the background and is also not stable against varying illumination. Therefore, the use of color information alone requires a stable and controlled laboratory environment with moderate

background noise.

2.2.2.2 Color Space

Besides a controlled laboratory environment, the selected color space plays an important role for achieving a well-performing skin color segmentation. Over the last decades, abundant research on different color spaces and their effects on segmentation performance has been done. The surveys of Kakumanu and Phung [14, 32] provide a broad overview of different color spaces, classification algorithms and their resulting performance.

Depending on the selected color space, the distribution of the skin and non-skin color pixels varies. Choosing an appropriate color space can reduce the overlap between these two classes and thus massively improve the classification results. To this end, it was observed that the skin color pixels variation in luminance is a greater predictor for the total difference between skin color pixels, than their variation in chrominance components [42]. Due to this, many approaches remove the luminance component to achieve a denser skin color distribution and to make segmentation more robust against changing illumination. Therefore, color spaces that implicitly separate luminance from chrominance components are usually preferred in the literature. One of these color spaces is called *YCbCr*, which we will use for our segmentation approach. The *YCbCr* color space is illustrated in figure2.7.

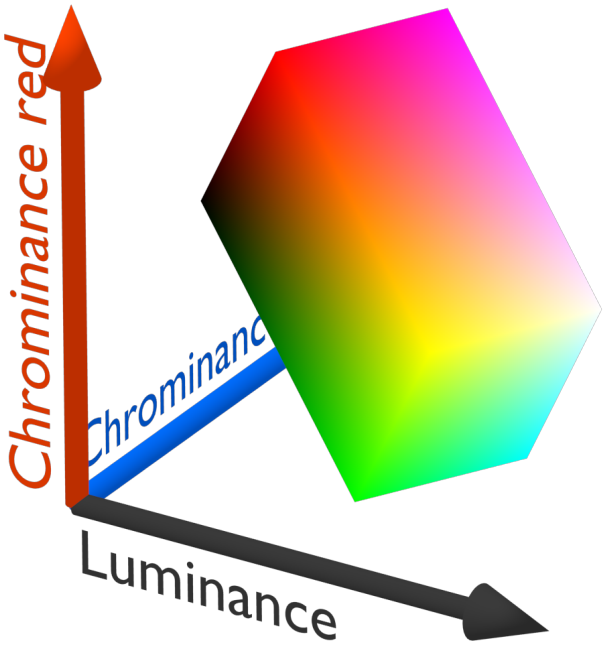


Figure 2.7: Illustration of the *YCbCr* colorspace. The depicted axes are luminance (black), chrominance red (red) and chrominance blue (blue). The colored cube in the center shows the corresponding RGB values.

The *YCbCr* model divides the color information into the luminance component Y and the two chrominance components Cb (Blue-Yellow Chrominance) and Cr (Red-Green Chrominance). For the reasons set out above, we will also drop the luminance and only consider the Cb and Cr components for our classification model.

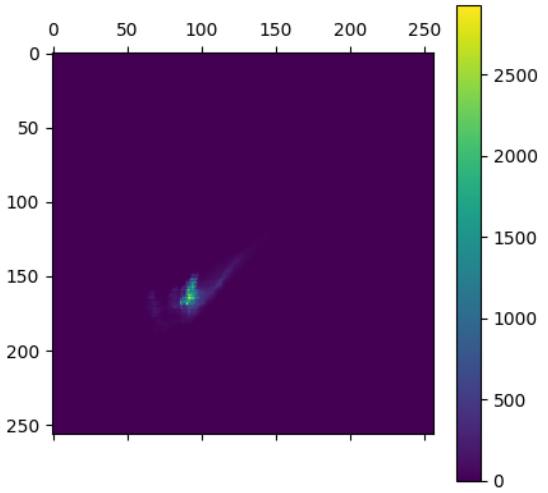
2.2.2.3 Training Data

In order to train our skin color classifier, skin- and non-skin-colored pixel data is required. For this purpose, we take 15 sample images of our recordings from our controlled laboratory environment. These images contain different pointing gestures to different object positions so that a variety of different shades of skin color are reflected in the data. The separation of skin and non-skin pixels is achieved by manually creating binary ground truth copies of all images in our training data. This is done by setting all skin-colored pixels to white and the non-skin-colored pixels to black. This results in original and ground truth pairs, where the ground truth mask allows the localization and extraction of skin and non-skin pixels from the respective original image. After extraction, the RGB-pixels tripels are then saved to a *Numpy* binary file. This pixel triples are then converted to the *YCbCr* colorspace.

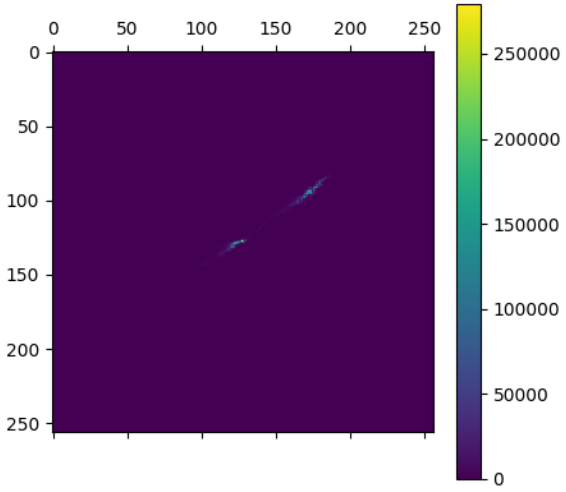
2.2.2.4 Bayesian Classifier with the Histogram Technique

Skin color classification can be modeled as a typical binary classification problem, where each pixel is assigned to either the skin color or non-skin color class. For our approach we use the *Bayesian classifier with the histogram technique* described by Jones and Rehg [12]. The process of skin color classification with this method is divided into two parts: First, the distribution of skin and non-skin-colored pixels is modeled. Second, a classification rule is defined that assigns each pixel of an image to either the skin or non-skin color class. To model the distribution of skin-colored and non-skin-colored pixels, we create a 2-dimensional histogram for each class. We achieve this by mapping the Cb and Cr components of the *YCbCr* colorspace to the histogram dimensions. Both dimensions are divided into 256 bins ($[0, \ldots, 255]$), which corresponds to the value range of the color space components. This way, each histogram cell represents the count of the corresponding Cb, Cr pairs from the training data (see section 2.2.2.3). The resulting histograms are visualized in figure 2.8.

Now we convert each histogram into a probability distribution. For this, each cell



(a)



(b)

Figure 2.8: Distribution of the extracted *CbCr* pairs of skin color data displayed as histograms. Figure (a) is the skin color histogram, (b) the non-skin color histogram. The y-axis of the histograms represent the *Cr*, the x-axis the *Cb* values.

is divided by the total count of the histogram (the sum over all cell values):

$$P(CbCr \mid skin) = \frac{s[CbCr]}{T_s}, \tag{2.1}$$

$$P(CbCr \mid \neg skin) = \frac{n[CbCr]}{T_n} \tag{2.2}$$

where $s[CbCr]$ is the count of the corresponding (Cb, Cr) pair in the skin histogram, $n[CbCr]$ is the equivalent count from the non-skin histogram, and T_s and T_n are the total counts of the skin and non-skin histograms, respectively [12]. This results in two probability distributions $P(CbCr \mid skin)$ and $P(CbCr \mid \neg skin)$, representing the conditional probability that the corresponding (Cb, Cr) pairs belong to skin color or non-skin color respectively. Given these two histograms of conditional probabilities, a classifier can be defined using the Bayes' maximum likelihood approach [7] . With this, a pixel can be classified as skin if

$$\frac{P(CbCr \mid skin)}{P(CbCr \mid \neg skin)} \geq \Theta \tag{2.3}$$

where $0 \leq \Theta \leq 1$ is the threshold that determines the ratio of true and false positives. We empirically determined that $\Theta = 5$ gives us the best classification results for our purposes. By applying the threshold, all histogram cell values above Θ are set to 255 and all values below Θ are set to 0. In this way, each potential (Cb, Cr) is assigned to either 255 or 0.

Finally, for the actual skin color segmentation, the binary histogram must be applied to the individual frames of the recordings. For this purpose, every current frame is converted into the $YCbCr$ color space and and the (Cb, Cr) values of the individual pixels are used to access their corresponding histogram cell. Then each pixel in the frame is assigned its histogram value, resulting in a binary image in which the skin color pixels are set to 255 (white) and the non-skin color pixels to 0 (black). This binary image is the basis for further processing steps in the task of hand detection.

The Bayesian classifier with the histogram technique is computationally very efficient, as it is based on only a few *Numpy* array accesses. Therefore, this method is very attractive for scenarios with limited computational resources, that need to run in real time. A drawback is that it does not generalize as well with little training data as for example Gaussian mixture models or multilayer perceptrons [14]. However, since the performance in our research environment is sufficient for our purposes and hardware on robotic platforms is often constrained, we made our decision towards efficiency.

2.2.2.5 Detection step

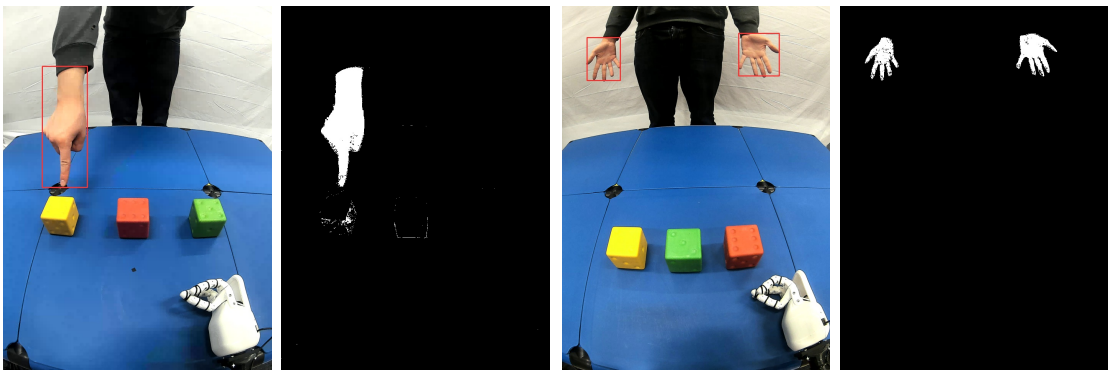
After distinguishing between skin- and non-skin-colored pixels, the actual hand must be detected. For this, we fall back on the assumptions made when defining our scenario. There we stated that we do not need the experimenters head in the scene for our experiments. Therefore, we moved the hands and objects into the focus of the recordings, which allows us to monitor subtleties in the hand movements more clearly. In addition, the fisheye effect of NICO's camera is less pronounced in the center of the recordings, so the hand movements appear less

distorted. As the experimenter is the only person allowed in the scene, we assume the experimenter's hands to be the only skin-colored entities in the recordings.

From this foundation, we find all the contours in the binary image. A contour is a finite set of two-dimensional coordinates that encloses the outer boundary of a recognized shape. From now on we will refer to two-dimensional coordinates as *points*. In *OpenCV*, a contour is represented as a *Numpy* array of *points*. In the next step, we filter out all contours smaller than an empirically determined threshold of 50. This removes noise, that has been identified as a contour. The rationale behind this is that even a well-functioning skin color segmentation produces some false positives, which are recognized as hands when the experimenter is not present in the scene. Since a hand contour is significantly larger than the noise, the two are well distinguishable. After the noise is removed, the remaining contours are considered hands, although a maximum of two hands is detected. Finally, we calculate the bounding boxes around each of the detected hands to save their position.

2.2.2.6 Tracking

Besides implementing a stable hand detection, the hand must be tracked. Tracking can be defined as the "*the frame-to-frame correspondence of the segmented hand regions or features*" [44]. Numerous approaches for hand tracking were introduced in the literature over the last years. A broad overview of different methods is given by Zabulis or Rautaray and Agrawal [44, 34].



(a) (b) (c) (d)

Figure 2.9: Tracking of one and two hands. (a) shows the tracking of one hand. The background noise in (b) indicates that the second hand is still being searched for. (c) shows the tracking of two hands. Since both hands are already identified, (d) indicates that the rest of the image is no longer considered.

Tracking algorithms are usually used on account of inefficient hand detection methods, which do not allow real-time performance when calculated every frame. Therefore, the hand detection is only performed when tracking is lost. However, if the

detection method is efficient enough to be computed in real-time, detection can be used for tracking as well. Since skin color segmentation is computationally efficient and stable enough to meet our requirements, we will use our detection approach for tracking.

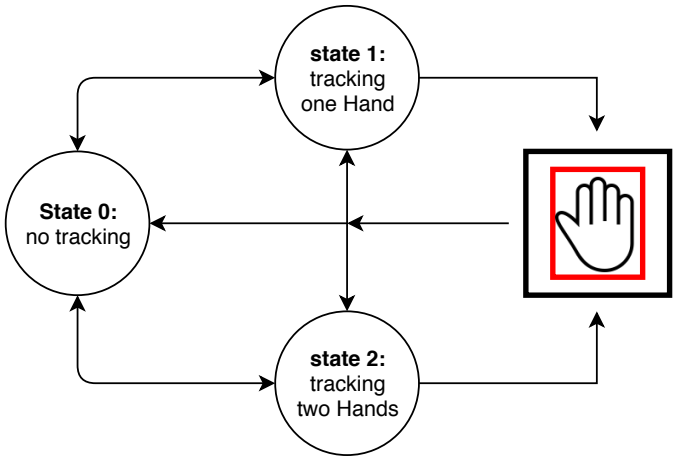


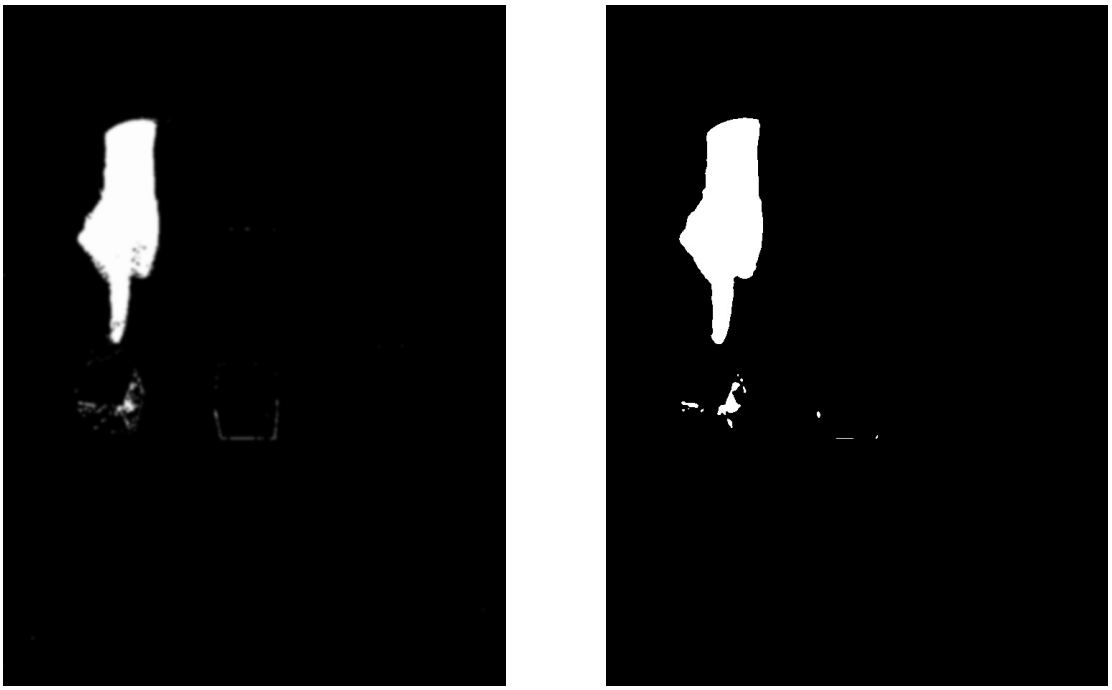
Figure 2.10: Life cycle of the hand tracking. When no hand is tracked, the tracking is in **state 0**. Depending on how many hands are detected after each frame, the state changes to **state 1** or **state 2**. Before searching the new positions of hands, each saved bounding box is enlarged by a padding of 30 pixels.

To formalize our tracking approach, we introduce a tracking life cycle (see figure 2.10). Such a life cycle has the advantage that the current state of the tracker is comprehensible at every step. This provides helpful insight into which functions can be executed at which point and lays the foundation for the following subtasks. In our tracking life cycle we make a general distinction between three states:

- State 0: no hand is tracked
- State 1: one hand is tracked
- State 2: two hands are tracked.

For each new frame coming from the camera or the recording, the tracking undergoes an iteration of its cycle. The only information transmitted between the frames is the current state and the respective bounding boxes of the hands.

When nothing is tracked, only the hand detection step is performed. Then the state changes according to the number of detected hands. If the state changes to state 1 or state 2, we save the bounding boxes for the next iteration. Here, we take each saved bounding box and enlarge it by a padding of 30 pixels. This is done for searching the new position of each tracked hand. We do not expect the respective hand to have moved out of their enlarged bounding box since the last frame. Therefore, we do not search for the new hand position outside that area.



(a) (b)

Figure 2.11: This figure shows the different morphological operations that are applied to the obtained binary image after skin color segmentation. In (a) a Gaussian Blur is applied. In (b) a binary threshold is applied on (a), that sets every pixel above a grey value of 100 to white and below that value to black.

This procedure provides the tracking with positional identity and prevents uncontrolled jumps of the bounding boxes. In state 1, we additionally search for the second hand in the original binary image. In state 2, as both hands are already found, we only search for their new positions. The tracking states are illustrated in figure 2.9.

For finding the new positions, we perform the initial detection step after applying a Gaussian blur and a binary threshold to the binary image. These operations smooth the contour, remove noise in the binary image and help maintaining a stable and less shaky hand contour over successive frames. The morphological transformations are shown in figure 2.11. Again, depending on how many hands are detected after this step, we change our state and save the new hand positions for the next iteration.

2.2.3 Object Detection

In order to be able to estimate the target of a deictic gesture, the referenceable objects present in the scene must be detected. For this, we need to model the

location of the objects and derive some characteristic to distinguish them from each other. However, more sophisticated, e.g. deep learning based object recognition approaches such as the *RetinaNet* by Lin et al.[19] or *YOLO* by Redmon et al. [35], require powerful hardware to run in real-time and are therefore not suitable for us. To this end, we are particularly not concerned with recognizing the specific type of an object, but rather with implementing a straightforward and efficient detection.

Based on this, we conclude that the most practical approach for detecting objects in our scenario, is to exploit the color values of the objects. This is reasonable since all colors present in the scene and especially the color of the objects (red, green, yellow) are clearly distinguishable. We, therefore, do not risk that colors with similar properties heavily confound with the detection of the actual objects. Furthermore, due to this clear differentiability, we do not have to learn complex color distributions as in the case of our skin color segmentation approach (see section 2.2.2.1) and can select the corresponding areas in the color space manually. Contrary to the *YCbCr* color space of the last section 2.2.2.2, use the HSV color space (HSV= Hue, Saturation, Value), as its properties make it especially convenient to distinguish the primary colors, which are most dominant in our scene.

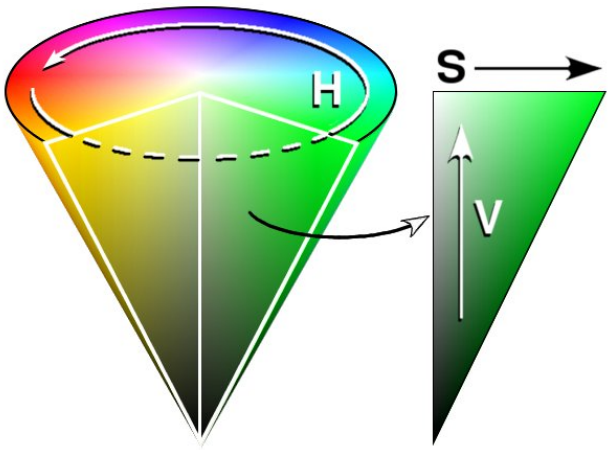


Figure 2.12: Illustration of the HSV (Hue, Saturation, Value) color space. The radial shape on top of the cone represents the different hue valus. The radius of the cone from medial to lateral defines the saturation. The height of the cone is the value component.

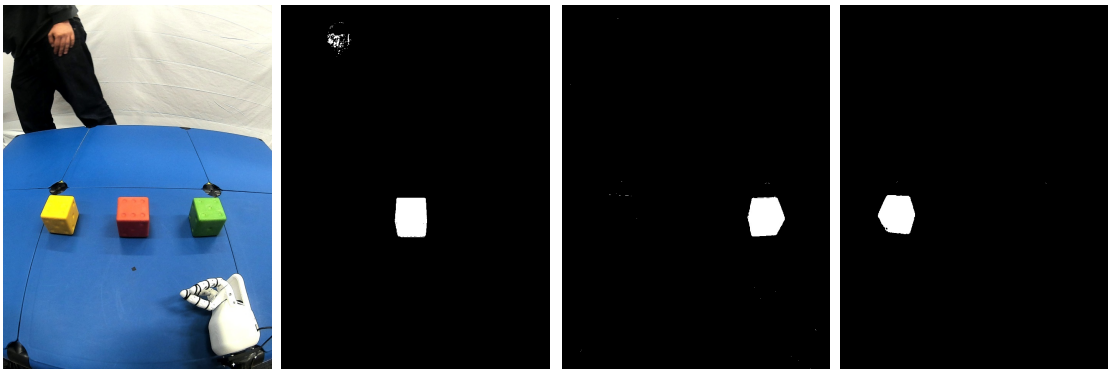
The HSV color space separates the hue information of a color from its saturation and value components. As shown in figure 2.12, the hue channel describes major color tones in a radial shape with color values ranging from 0 to 359, while the saturation and value components influence the colorfulness and intensity of the color and both range from 0 to 359. In this way, the most important properties of

a color can be described by selecting one interval in the hue channel. The saturation and value channels allow the filtering of very dark, gray and light tones by limiting the channels extreme values. This makes manually selecting color regions with the HSV color space intuitive and efficient.

For the final modeling of the object colors, we apply empirically determined threshold values to the individual HSV channels. Since each channel must be limited from above and below, we define six values per object color:

$$T_{color} = \{(min_h, min_s, min_v), (max_h, max_s, max_v)\} \tag{2.4}$$

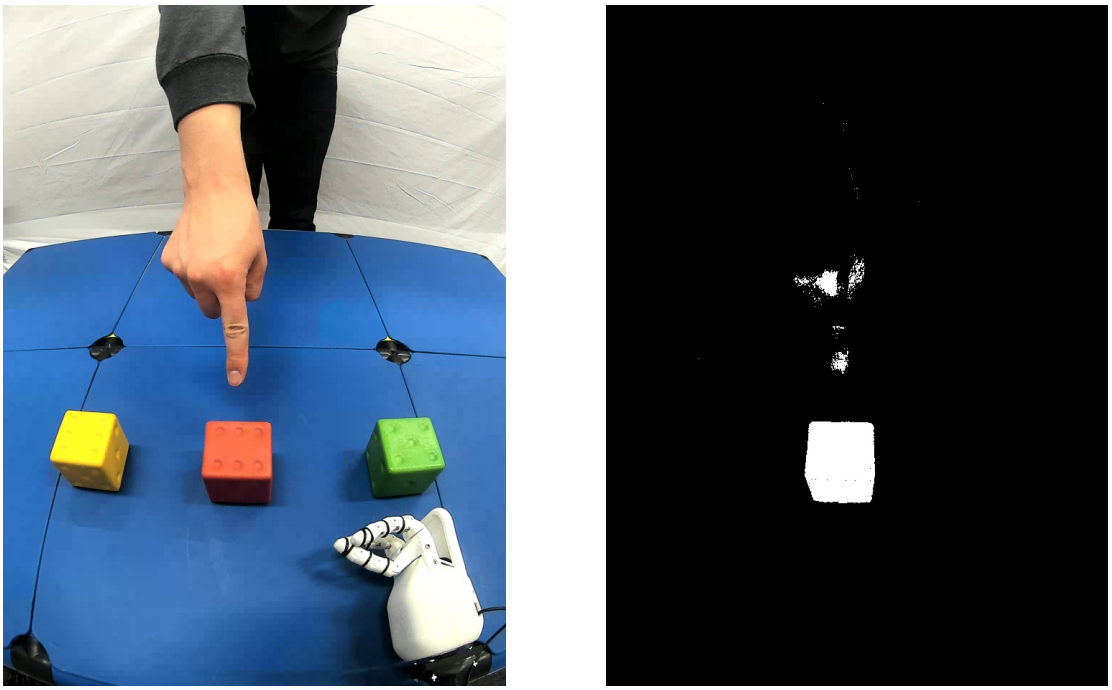
For the green object we determined $T_{green} = \{(40, 78, 26), (90, 255, 255)\}$ and for the yellow object $T_{yellow} = \{(17, 130, 80), (37, 255, 255)\}$. Regarding the threshold values for the red object, we need to define two vector pairs. This is due to red being distributed around degree 0 on the hue channel. Since we can not represent this radial shape in *Python*, we define one vector for the range before, and one vector for the range after 0 degrees. This results in $T_{red1} = \{(0, 145, 175), (10, 255, 255)\}$ and $T_{red2} = \{(130, 50, 10), (180, 255, 255)\}$.



(a) (b) (c) (d)

Figure 2.13: Result of the segmentation for the differently colored objects. Figure (a) shows the initial recording and the figures (b), (c) and (d) show the result for the red, green and yellow object respectively.

In order to detect the objects in the scene, we first convert the corresponding image to the HSV color space and then apply each threshold vector separately to different copies of that image. In this process, we set each pixel that is in the defined range to 255 (white) and each pixel outside this range to 0 (black). As illustrated in figure 2.13, this results in three different binary images in which the recognized object is represented by a white blob. Similar to hand detection, we now find the largest contour in each binary image to obtain the objectcs positions. Afterwards, we model the respective object positions by calculating a minimum bounding box around each contour. In addition to the location, we also store the color of the objects to be able to distinguish them more conveniently in the further process.



(a) (b)

Figure 2.14: Example of a recording in which the skin color of the experimenter's hand is confused with the color of the red object by our initial object detection. This issue could be preserved by considering the hand position of the last time step in the object detection.

While testing the object detection, we empirically discovered that the skin color of the experimenters hand slightly confounds with the detection of the red object (see figure 2.14). We, therefore, assume that the color red in the HSV color space is close to the distribution of skin color. To avoid these interferences, we pass the hand position of the last time step to the object detection and set each pixel in the area of the hand's bounding box in the binary image of each object to 0 (black). With this we do not consider the hand any more when detecting objects, which provides our approach with more stability.

2.2.4 Pointing Intention

Before being able to estimate the direction and target of a deictic gesture, we need to detect if a gesture is performed. To achieve this, we have to incorporate the experimenter's intention to perform a deictic gesture. For modeling this "*pointing intention*", two features are interesting: the hand pose and the motion of the hand. First, considering the hand pose is essential, as it allows distinguishing arbitrary hand movements from deictic gestures. As described in our scenario (section 2.1.2), we define a deictic gesture as pointing with the index finger. This hand pose is

an indication that the experimenter intends to perform a pointing. Second, since deictic gestures are dynamic gestures and therefore always associated with motion, it is logical to consider their temporal structure. Regarding this, we originally planned to model the gestures motion with optical flow. However, this proved to be incompatible with our approach in the end. In section 2.2.4.3 we outline the reasons for this. To this end, we constrained ourselves to model the pointing intention solely with the pose derived from the segmented hand. Since our primary focus is on pointing to object mappings, we argue that in this work the temporal structure of the deictic gesture can even be neglected. Above all, we aim to investigate the accuracy and ambiguity of deictic gestures with the NICO robot. In the following, we describe the methods for determining if the experimenter is performing a deictic gesture. Our approach is based on publications by Wen [40] and Yeo [43].

2.2.4.1 Convex Hull

In the literature, a popular approach for estimating the pose of a segmented hand is to calculate the *convex hull* of its contour [4, 40, 43]. We remember that in the hand detection step we obtain the contour of the hand, which is represented as a finite set of *points*. The *convex hull* of a finite set of *points* can be defined as the "*minimum area convex polygon enclosing the set*" [25]. In other words, if a contour has non-convex segments, a minimal convex polygon can be drawn to enclose it. This polygon is the *convex hull*. The cavities of the non-convex contour segments are called *convexity defects*. Since these *convexity defects* describe every cavity in the contour, they provide a comprehensive model of its shape. For calculating the *convex hull* we use an algorithm introduced by Jack Sklansky [37]. In *OpenCV*, the *convex hull* is represented as a finite set of indices, corresponding to *points* in the contour. Figure 2.15, illustrates the *convex hull* of a hand contour.

The bold black line enclosing the hand is the *convex hull*. The areas A to I are the *convexity defects*. Each convexity defect is described by four variables C_s, C_e, C_f and C_d. The variables C_s, C_e and C_f are *points* on the contour and C_d represents a distance. C_s and C_e indicate the start and end *point* of each convexity defect. C_f lies between C_s and C_e and is the *point* with the maximal distance to the corresponding hull segment. This distance is described by the variable C_d. These variables describe the geometrical characteristics of each defect. Since these characteristics reflect the shape of the contour, they offer a canny way to approximate the hand's current state.

[1]The source for the picture of the hand: `http://pngimg.com/download/905`, (accessed: 05.08.2018).

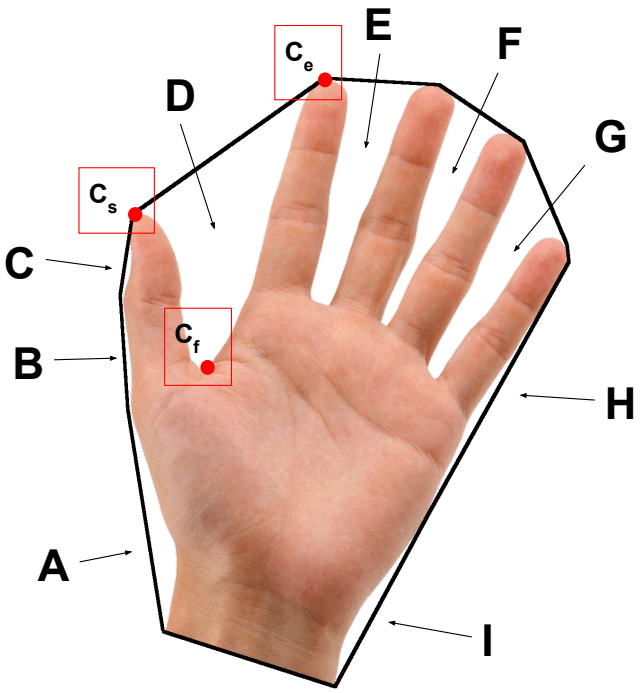


Figure 2.15: Illustration of a hand's[1] *convex hull*, which is the bold black line around the hand. The areas $\mathbf{A} - \mathbf{I}$ are the convexity defects of the hand. $\boldsymbol{C_s}$ is the *point* on the hand contour where the convexity defect starts, $\boldsymbol{C_e}$ where it ends and $\boldsymbol{C_f}$ is the point with maximal distance to the corresponding hull segment (from $\boldsymbol{C_s}$ to $\boldsymbol{C_f}$).

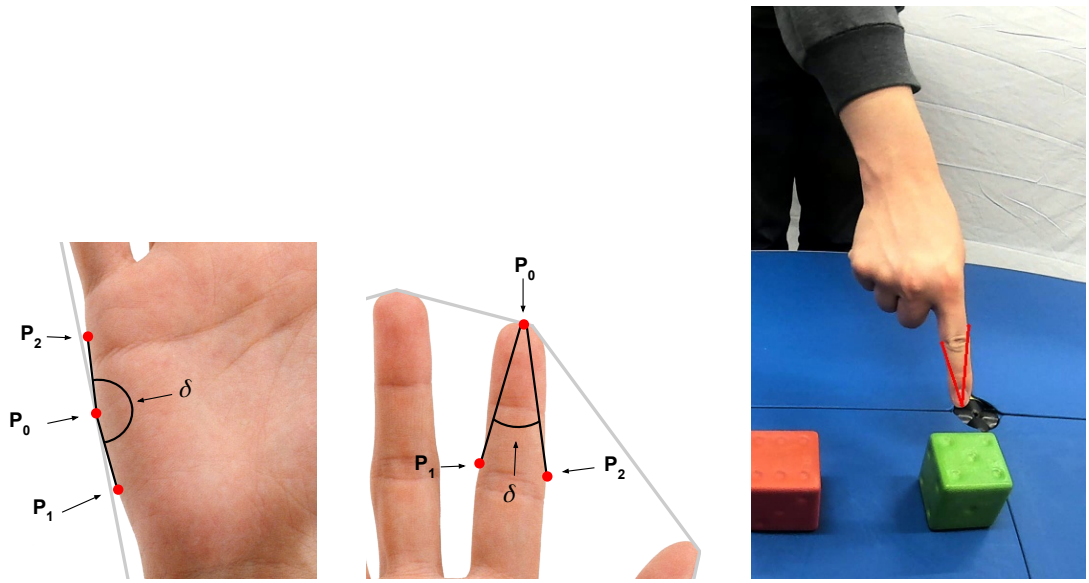
2.2.4.2 Fingertip Detection

In the following, we exploit the properties of the *convex hull* and *convexity defects* to determine if the experimenter intends to perform a deictic gesture. This is achieved by finding the fingertip of each extended finger. Our procedure divides itself into detecting the hand's fingertips and verifying that exactly one finger is extended. If this is the case, we assume that the experimenter is currently pointing.

For detecting the fingertips, we first notice that all *convex hull points* are actually a subset of their contour, since they are located directly on it. Second, due to the hand's physiognomy, the contour *points* of an extended fingertip will most likely be part of the *convex hull*. This obviously depends on the camera angle and the possible hand position, but in our scenario this was usually the case.

We examine each *point* of the *convex hull* to check if it is located on a fingertip. For this, we first calculate the *convex hull* and *convexity defects* of the current hand contour. Then, all *convexity defects* with C_d below a threshold of 4.0 are filtered out, as otherwise noisy, frayed contour segments are detected as small *convexity*

defects. We remember that in *OpenCV* a *convex hull* is represented as a set of indices that map to their corresponding *points* in the contour array. For each *point* P_0 in the *convex hull*, we find two *points* P_1 and P_2 in the contour array, where P_1 is located 12 array indices before and P_2 14 array indices after P_0. Thereby, we obtain a triangle for each *convex hull point* (see figure 2.16).



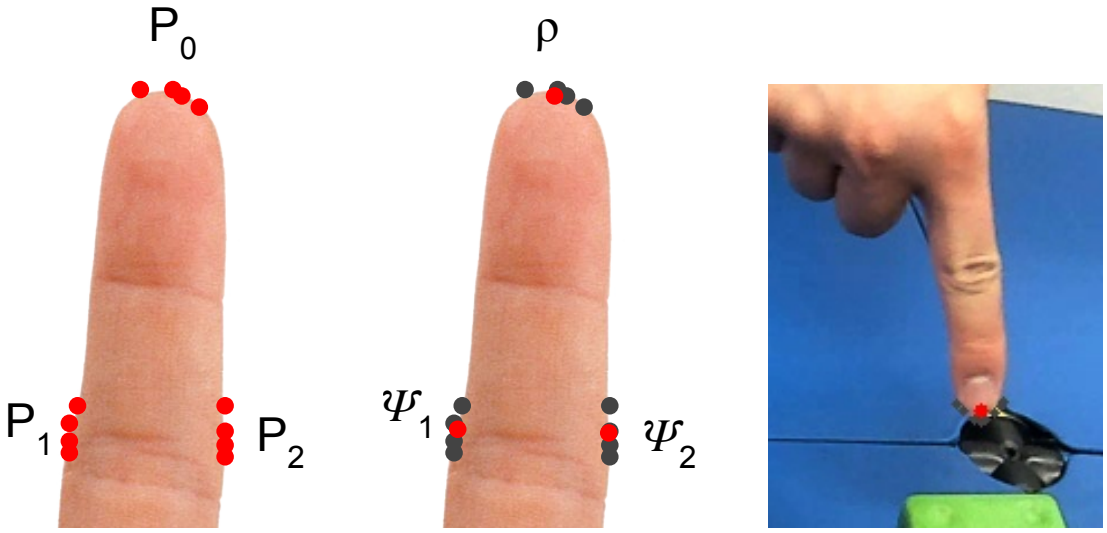
(a) (b) (c)

Figure 2.16: Illustration of the rationales of our fingertip detection procedure. We assume the contour points on the fingertips are also part of the convex hull. We calculate the angle $\boldsymbol{\delta}$ of each fingertip candidate $\boldsymbol{P_0}$ by using the points $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$. Due to the geometrical properties of the hand, if $\boldsymbol{\delta}$ is an acute angle, we consider $\boldsymbol{P_0}$ to be located on a fingertip. (a) shows the case that no fingertip is detected, (b) shows the case that a fingertip is detected. (c) shows the result of the procedure in the scenario.

For each triangle, we calculate the angle δ of the line segments $\overline{P_1P_0}$ and $\overline{P_1P_0}$. We argue that δ reflects the local geometric structure of the area around P_0. Consequently, the value of δ can be used to make assumptions regarding the position of P_0. When observing the entire hand, it is noticeable that the fingers are geometrically unique due to their long and narrow shape. This can be used to the effect that if P_0 is located on a fingertip, the *points* P_1 and P_2 are also on the same finger, as the finger is usually longer than 12 contour *points*. In this case, δ would be an acute angle, since the distance between P_1 and P_2 would be, by reason of the narrow finger shape, comparatively small. Conversely, when P_0 is at palm level, δ represents a wide angle given the flatter contour. Since we only consider hull *points* acute angles can only occur on fingertips. The rationale behind this approach is illustrated in figure 2.16.

Based on empirical tests, we define $\omega = 40$ as the threshold value for δ. If δ is

smaller than ω, we consider P_0 a fingertip candidate. After this step, we find several fingertip candidates. However, a fingertip usually consists of multiple hull *points*, all satisfying the angle constraint. As a result, clusters of fingertip candidates emerge at each fingertip. Since we know neither the number nor the position of the fingertips, we need to find these clusters in order to estimate the fingertip positions.



(a) (b) (c)

Figure 2.17: Illustration of the calculation of the final fingertip position $\boldsymbol{\rho}$. (a) shows the calculated point groups $\boldsymbol{P_0}$, $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$, whose clustering results in the estimated cluster centroids $\boldsymbol{\rho}$, $\boldsymbol{\psi_1}$ and $\boldsymbol{\psi_2}$. (c) depicts the fingertip candidates (grey) and the final fingertip position (red) in the scenario.

For this purpose, we first perform hierarchical clustering using the single linkage algorithm and then convert the created hierarchy into a flat representation by using a distance threshold t [11]. We use this algorithm because it does not require us to know the number of clusters in advance. The linkage algorithm creates a hierarchical cluster structure in a bottom-up manner. In each iteration it combines the two clusters closest to each other, starting with each element being its own cluster. Then, each hierarchy level from bottom to the top is checked, whether some pair of *points* in the clusters exceed the distance threshold t. The first hierarchical level, that does not violate this assumption determines the detected clusters. An overview of this method can be found in Rui Xu's survey [41]. After detecting the clusters, we calculate each cluster's centroid, to receive the final fingertip positions. Each detected fingertip is denoted by ρ_i, where $1 \leq i \leq n$ and n is the amount of detected fingertips. In addition, for each fingertip ρ_i, we take the P_1 and P_2 *points* of each fingertip candidate and calculate their centroid respectively. We denote the centroid of the P_1 *points* as ψ_{i1} and the centroid of P_2 as ψ_{i2}, where i represents

the number of the corresponding fingertip. This leaves us with a set of fingertips ρ and for each fingertip ρ_i two lateral *points* ψ_{i1} and ψ_{i2}. The fingertip candidates and the subsequent clustering is illustrated in figure 2.17.

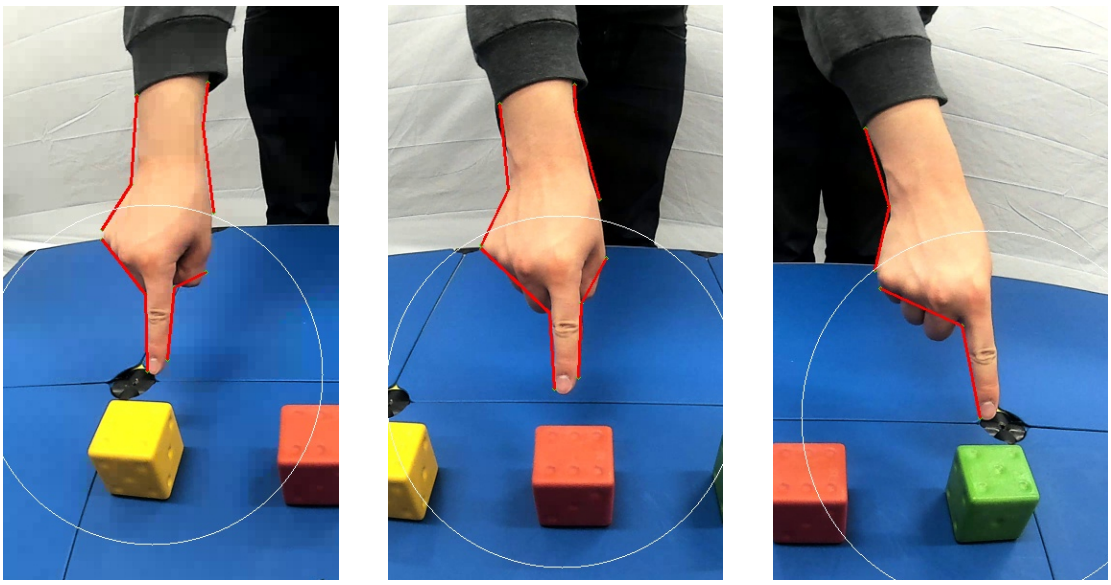
Although this method is stable in most situations, a noisy segmented hand contour or interlocking fingers can easily lead to misidentified fingertips. In order to improve robustness, it is reasonable to implement additional methods for identifying the pointing intention. With this in mind, we observe the *convexity defects* in the area between the detected fingertips and the hand's centroid. For reasons of simplicity, we estimate this area with a circle that has the fingertip as the center and the distance to the hand's centroid as the radius. If exactly one finger is extended, this area contains one to two *convexity defects*. These defects are created by the two cavities left and right the pointing finger. Although, in some pointing angles, only one convexity defect occurs. In the following, we will only consider a hand pose as pointing, if we detect one or two *convexity defects* in the area estimated by the circle. A convexity defect is detected when the *point* C_f of the convexity defect is in the area of the circle. We empirically found this additional assumption to significantly decrease falsely detected pointings caused by misidentified fingertips. In Figure 2.18, different hand pointings are shown to illustrate the relationship between different pointing angles and *convexity defects*. Figure 2.18b shows a straight pointing with two *convexity defects*. Furthermore, the mentioned area between the fingertip and the hand centroid is shown. In figure 2.18c, the hand *points* sharply to the right, such that only one convexity defect can be observed.

Represents the changing convexity errors with changes in the hand.

This method of detecting fingertips and identifying the pointing intention has its advantages and disadvantages: The major drawbacks of this approach lies in its high dependence on a well-performing hand segmentation. As soon as the segmentation performance decreases and the contour no longer accurately reflects the geometric characteristics of the current hand shape, the correctly identified pointing intentions drop significantly. Moreover, since the segmented hand shape is merely a two-dimensional projection of a complex three-dimensional object, there are some ambiguous hand positions that repeatedly lead to incorrectly identified pointing intentions. On the positive side, the method is robust against rotation and is, therefore, applicable to other camera angles and hand orientations. Furthermore, the *convex hull* and detected fingertips, provide an accurate model of the current state of the hand, allowing for the estimation of various hand positions.

2.2.4.3 Gesture's temporal structure and Optical flow

Initially, we aimed to model the temporal structure of deictic gestures with optical flow. Optical flow is a method for estimating motion between consecutive video frames. This motion is represented by displacement vectors that map the alternations of pixel positions over individual frames. This allows analyzing the direction



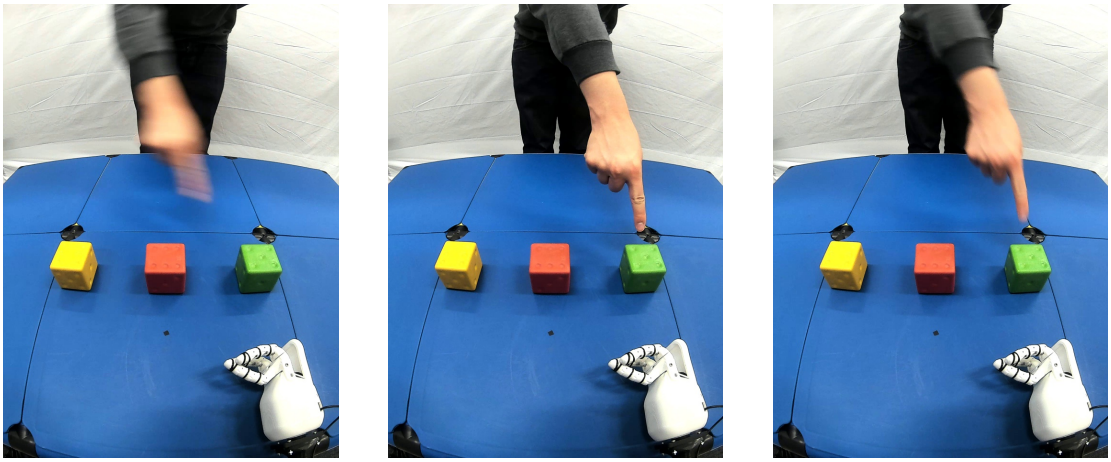
(a) (b) (c)

Figure 2.18: Change of convexity defects with corresponding pointing direction changes. When one to two convexity defects (the *point* $\boldsymbol{C_f}$ in figure 2.15) occur in the area of the white circle, a pointing intention is recognized. (c) illustrates that some pointing angles only produce one convexity defect.

and strength of certain movements in a video sequence. There are several methods available for calculating optical flow. A prominent and computationally efficient example is the Lucas-Kanade method, which we have used for our efforts. A detailed overview of this method can be found in the Bouguet's paper [2]. In this section, we explain why optical flow was not compatible with our approach in the end.

As outlined in Pavlovics survey on hand gestures [31], the temporal structure of a dynamic gesture divides itself into three phases: *preparation*, *peak* and *retraction*. In the *preparation phase*, the gesture starts to develop from a previous hand position by means of a movement. Thereby, the hand moves until it comes to a standstill. This is exactly where the gesture reaches its *nucleus* or *peak*. In this second phase, the gesture communicates most of its semantic information by remaining static for an indefinite time. Finally, the gesture ends with the hand moving again. This is the *retraction phase* in which the hand either returns to a resting position, performs another gesture or moves on to some other activity. These three stages are portrayed in figure 2.19. A deictic gesture follows this exact same pattern. In the *preparation phase* of the pointing, the arm and the index finger both extend and align towards the target object. Then the gesture reaches its *peak* by statically pointing to the targeted object. Finally, the hand and arm *retract*. Therefore, we decided to model these temporal characteristics in order to make the gesture recognition more robust against misidentified pointings and to provide a more

comprehensive picture of the gesture.



(a) (b) (c)

Figure 2.19: Illustration of a deictic gesture's temporal structure. (a) shows the gesture during *preparation*, (b) at its *peak* and (c) shows the exact point where the hand *retracts*.

We started our first naive approach by applying a motion mask to the binary image obtained from the hand tracking. The term motion mask can be understood as the segmentation of the image into moving and non-moving parts. Since we stated in our scenario that the position of NICO and the experimenter is fixed throughout the entire experiment, we argue that the hands are the only moving skin-colored objects in the scene.

We define a resting hand as inactive and a moving hand as active. We further define that only an active hand is considered to be pointing. At this point we encountered first problems. Since optical flow is a model for detecting pure motion over successive video frames, no motion is detected during the gesture's peak. Therefore, as soon as the gesture reaches its peak, the hand will become inactive and NICO would stop recognizing the current gesture. Consequently, for implementing a gestures temporal structure all three states need to be modelled and moreover, always be distinguishable.

One way to solve this would be to wait for the peak after the hand becomes active. When the hand comes to a standstill, a peak is detected and possible pointings are recognized. If a pointing is recognized, the next movement is expected to initiate the retraction phase. After the retraction phase, when the hand comes to a standstill again, the hand gets set to inactive. If no pointing is detected, the hand gets set to inactive as no gesture occured at all.

Although these rules might work in some cases, an approach with this rationale soon runs into edge cases. For instance, the retraction phase of a pointing is often

simultaneously the preparation phase for the next pointing. For example, the experimenter *points* to object A and changes the target to object B while pointing. The described approach would identify this movement as a retraction and set the hand to inactive, although the experimenter is still pointing. This example quickly illustrates that modeling the temporal structure of deictic gestures would require a large set of rules for handling edge cases.

This would neither result in a coherent implementation nor contribute to a better understanding of deictic gestures itself. Rather, this would lead to a bloated and inflexible implementation. Since we aim for a slim architecture and moreover focus on pointing to object mappings, we decided to ignore the temporal structure of deictic gestures.

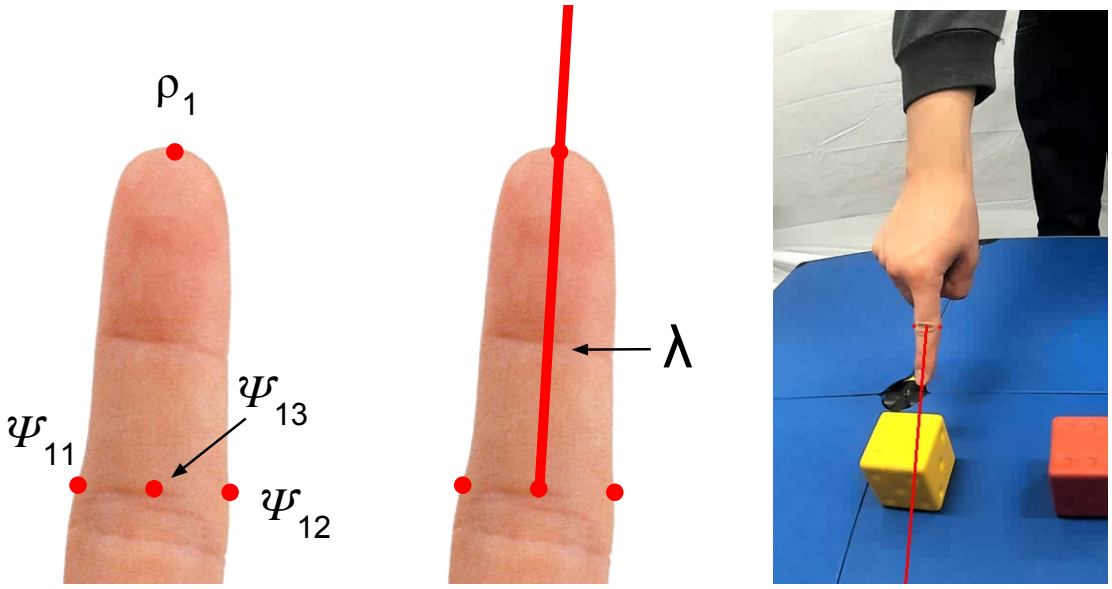
2.2.5 Pointing Estimation based on Computer Vision

After the last section (2.2.4), where we discussed the process of identifying the pointing intention, we now need to recognize the target of the deictic gesture. To achieve this, we approximate a pointing array from the geometric features of the segmented hand, which serves as our model for the pointing direction. We combine this pointing direction with the object detection described in section 2.2.3, to incorporate the gesture's potential targets. The following describes how this pointing array is calculated and how we use it to predict the target of the deictic gesture.

2.2.5.1 Estimating a Pointing Array

The first obvious solution for recognizing the target of a deictic gesture is to approximate a pointing array from the extended index finger of the person executing the gesture. This pointing array hits the objects in its path and the object most accurately hit is determined as the target of the gesture.

For approximating the pointing array, we first remember that in the process of detecting fingertips, we saved two *points* ψ_{i1} and ψ_{i2} on the finger contour, for each detected fingertip ρ_i. Since approximating a pointing array is preceded by a successfully identified pointing intention, we know that only one fingertip ρ_1 is detected. Therefore, we obtain the fingertip position ρ_1 and the two lateral finger contour *points* ψ_{11} and ψ_{12} for our calculations. On this basis, we calculate the point between ψ_{11} and ψ_{12}, which we call ψ_{13} (see Figure 2.20a). Now we calculate the line segment between ρ_1 and ψ_{13}, which we immediately extrapolate to the edge of the current frame. The result is a line segment λ, that starts at ψ_{13}, continues over ρ_1 and ends at the end of the frame. We define this line segment as our model for estimating the pointing direction (see Figure 2.20b). This line segment is recalculated for each new frame.



(a) (b) (c)

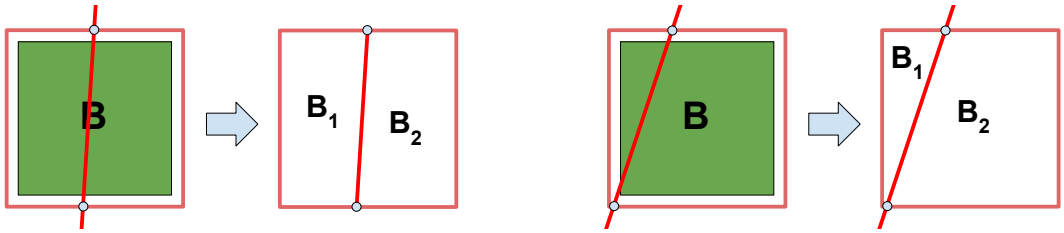
Figure 2.20: Calculation of the pointing array. The line segment between the *points* $\boldsymbol{\rho_1}$ and $\boldsymbol{\psi_{13}}$ in (a) are extrapolated to the pointing array $\boldsymbol{\lambda}$ in (b). (c) shows the calculated pointing array in the scenario.

2.2.5.2 Predicting with the Pointing Array

For recognizing the target to which the experimenter is currently pointing, we first detect all the referenceable objects in the scene. Afterwards, we monitor which of the detected objects are hit by the pointing array. We then utilize a measure of quality to determine which object is hit most accurately. With this, we aim to resolve ambiguous object constellations and to evaluate this approach in the end.

After identifying a pointing intention, we estimate a pointing array as described in the last section. To determine whether this pointing array hits an object, we check if it intersects with the bounding boxes of the detected objects. Every object where the bounding box intersects with the pointing array is considered hit and its pointing quality is calculated. The level of that quality depends on how centrally the bounding box is intersected. For calculating this, we consider the exact intersection *points* of the pointing array and bounding box. With these *points* we divide the bounding box into two halves B_1 and B_2 (see Figure 2.21). Now we calculate the area of each half and divide the smaller half by the larger half. This gives us a value between 1.0 and 0.0, which we define as the pointing quality ϕ.

The rationale behind this approach is that the more centrally the object is hit, the higher is the probability that the experimenter intended to target the object. So, if the object is hit centrally, the area of the smaller and larger half of the bounding box are almost equal and therefore the value would be close to 1.0 (see Figure



(a) (b)

Figure 2.21: Illustration of the calculation of the pointing quality. (a) shows an accurately hit object, which results in an evenly divided object and thus in a higher rated pointing quality. In contrast, (b) shows an only laterally *hit* object that leads to an unequal division of the referenced object and thus to a low rated pointing quality.

2.21a). By contrast, if the object is hit only laterally, one half would be significantly smaller than the other and the value would be close to 0 (see Figure 2.21b). If ϕ is smaller 0.2, we annulate the detected hit and consider the pointing a miss. Otherwise, the detected hit remains. If the object constellation is ambiguous, such that the pointing array intersects with multiple objects, the object with the highest pointing quality ϕ is chosen. In the case where the index finger itself intersects with a bounding box and thus no two intersection *points* occur, the line segment between ψ_{13} and ρ_1 is used to find the missing intersection. The object with the highest pointing quality is predicted the target of the current gesture and the objects position and color are saved.

Due to the simplicity of this approach, some disadvantages emerge, which have to be considered. First of all, since the pointing array is simply extrapolated to the end of the screen it may hit objects that are far from its actual target. This might especially be the case for more uncontrolled environments with multiple objects. Another downside is that all objects must be detected beforehand and tracked over time, which does not scale well with an increasing number of objects in the scene. Furthermore, if the pointing array does not hit a bounding box at all, this approach fails to establish a *gesture-target* mapping. This models a pointing without a goal, which does not exist in the real world. Finally, misidentified pointing intentions, that arise from ambiguous hand postures, are immediately converted into pointing arrays. This leads to uncontrolled pointings, which the experimenter may not intend.

On the positive side, this approach allows a very direct pointing, which provides the experimenter with a lot of control over his gesture. Moreover, in our controlled laboratory environment, with our assumptions, this approach works stable in most cases.

2.2.6 Pointing Estimation with a Growing When Required Network

Although pointing with a pointing array is a reasonable approach, we want to address its drawbacks outlined in the last section. These are the unfiltered mapping of incorrectly identified pointing intentions, the modeling of pointings without a goal, the fact that all objects present in the scene must be detected a priori and that objects far from the actual target might be considered. As an attempt to tackle these challenges, we want to investigate whether and how we can model the distribution of deictic gestures in our scenario.

This could be a promising approach, as it allows us to identify outliers which could help to reduce misidentified pointings, as these probably do not match the distribution of the remaining gestures. Furthermore, knowing the distribution of these gestures provides the basis for deriving their target positions. The rationale behind this is that each individual deictic gesture in the distribution maps to a specific position on the table. If we model the distributions of these gestures, we may be able to find clusters of positions to which gestures with specific characteristics map. This has the potential to solve the problem of modeling pointings without a goal and the need to detect all objects beforehand, as we first receive the target position from our model and subsequently check whether an object is detected at that position. This reduces the space we have to consider for the detection of objects and gives us more control over further actions if no object is found. It might also solve the problem of considering objects far from the target, however, it is unclear where the gestures to these ambiguous object constellations are located in the distribution and whether they are clearly distinct from another. This can only be evaluated if object constellations with more distant objects were represented in our recordings, which is unfortunately not the case. However, this approach has the potential to employ a more efficient and natural model for the task of recognizing deictic gestures.

In the following section, we describe our approach for modeling the distribution of deictic gestures in our scenario. For this, we first describe how we quantify expressive properties of the hand while pointing and link these to their corresponding target position on the table. Afterwards we detail how we preprocess this data into individual observations and labels. Finally, we describe the Growing When Required network (GWR) [22], which we use to model the distribution and to predict the target positions of the deictic gestures. We chose a GWR because it fulfils the requirements described above by mapping the topological properties of the gestures and has interesting tools for noise filtering. Since the GWR is an unsupervised learning model, which is intrinsically not necessarily suitable for classification, we extend it with two labeling functions which allow us to learn and predict the targets of deictic gestures.

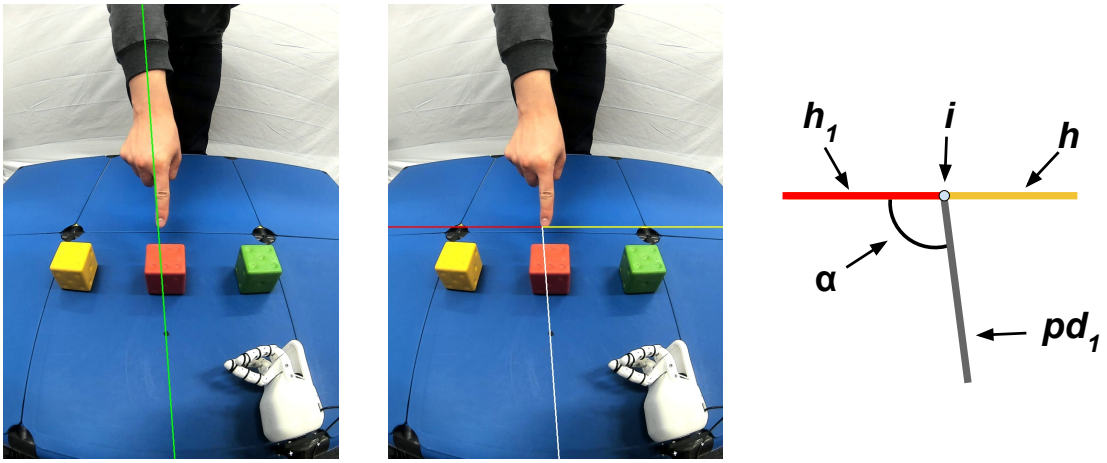
2.2.6.1 Data Acquisition and Preprocessing

To collect the data for representing the distribution of deictic gestures, we fall back on our recordings which we describe in section 2.1.4. These consist of 95 videos, each containing one object constellation. In each constellation, the experimenter *points* to every object exactly once. Since these 95 constellations are divided into 16 constellations containing one object, 33 constellations containing two objects and 46 constellations containing three objects, we obtain 220 individual deictic gestures from our recordings. In order to extract the features of these pointings, we consider their individual frames. To obtain the exact frames in which the experimenter is pointing, we annotate the start and end positions of each pointing in each recording. The number of frames each pointing provides depends on its duration and how *OpenCV* encodes the frames, which is inconsistent over the different recordings. For modeling the position to which the pointing refers, we manually draw a bounding box around the object that the experimenter currently targets. Finally, we save the color of this object to make the pointing comparable with the object detection. This gives us 220 annotated pointings, with a start position, end position and the color and bounding box of the targeted object attached. We also save the filename of the recording in which the gesture took place to access the image associated with the specific gesture. With the filename we also save the ambiguity class of the object constellation. We save this data in a csv-file for further processing.

To obtain the data representing the distribution of deictic gestures, we extract expressive properties of the hand of each annotated frame. To this end, we first apply the methods from the hand detection section (see section 2.2.2) to receive a segmented hand shape. Afterwards, we obtain the properties extracted in the process of detecting the pointing intention in section 2.2.4. Based on this, we decide on meaningful properties that reflect different characteristics of the gestures to represent their distribution. Besides the integration of features that cover a multitude of hand properties, we especially aim at features that exhibit a certain uniqueness across the various gestures. This is essential for the different gestures to be clearly distinguishable by their characteristics. On that note, we select the x- and y-coordinate of the hand's centroid and fingertip. These four features provide significant positional information of the hand, as each pointing will show a different permutation of these values. Furthermore, we integrate the angle of the hands pointing direction. While pointing to different target positions, the pointing angle changes continuously as the direction of the pointing changes. This allows us to capture not only the position of the hand but also the direction of the pointing.

Since we already have the positions of the hand's centroid and fingertip from applying the methods of the pointing intention section (see sections 2.2.4.1 and 2.2.4.2), we now determine the angle of the pointing direction. For this, we must first decide how to approximate this pointing direction in general. One approach would be taking the estimated pointing array of the previous section 2.2.5. This approach is useful for modeling the pointing direction for real-time scenarios, because it maps

changes of the hand's index finger directly to the pointing array, which provides the experimenter with a lot of controll over the direction of the pointing. For our data acquisition, however, it introduces too much variance into the pointing array, since it depends only on three contour *points* (see Figure 2.20a). Consequently, any noise in those *points* is directly mapped to the pointing array. When estimating the distribution of deictic gestures, our aim is to incorporate features with stable characteristics, so that the overall variance of the distribution remains low. This has the background that a high variance in the input data inevitably leads to a higher prediction error in the trained model. Therefore, we approximate the pointing direction by considering the entire contour (see figure 2.22a). This results in a lower variation, as alterations in the pointing array now depend on all contour *points* and thus on a wider range of factors. The approximation of the pointing direction with the entire contour is achieved by calculating the weighted least-squared error of the distance between the line and the contour *points*. The line with the ultimately lowest error is used for fitting the contour. We call this line *pd* and use it as our model for the pointing direction.



(a) (b) (c)

Figure 2.22: Calculation of the pointing direction neccessary for modeling the distribution of deictic gestures. (a) shows the approximated pointing direction by considering the entire contour. (b) shows the horizontal line $\boldsymbol{h}$ and the line segments $\boldsymbol{h_1}$ (red) and $\boldsymbol{pd_1}$ (white) resulting from calculating the intersection $\boldsymbol{i}$ of $\boldsymbol{h}$ and the pointing direction $\boldsymbol{pd}$. (c) illustrates how the angle $\boldsymbol{\alpha}$ is obtained.

Finally, for calculating the angle of the pointing direction, we draw a horizontal line h at the level of the fingertip over the entire width of the frame. Then we calculate its intersection with pd and obtain two line segments h_1 and pd_1. We call the intersection point i. As illustrated in figure 2.22b, h_1 goes from the left edge of the frame to i and pd_1 from i to the bottom edge of the frame. We then take these two line segments and calculate the angle α as depicted in figure 2.22c.

In the process of extracting the hand's properties, we observe the frames in chrono-

logical order. This allows us to take the average of the last 5 calculated angles for each frame of each pointing to further reduce introduced noise. The total hand properties of each processed frame are saved in a vector:

$$V_i = (\alpha_i, c_{xi}, c_{yi}, \rho_{xi}, \rho_{yi}) \tag{2.5}$$

where α_i is the angle of the pointing direction, c_{xi} the x-coordinate of the hand's centroid, c_{yi} the y-coordinate of the hand's centroid, ρ_{xi} the x-coordinate of the hand's fingertip and ρ_{yi} the y-coordinate of the hand's fingertip , of the i-th observation. The target of the individual gestures, represented by a bounding box, are saved as well:

$$B_i = (x_{1i}, y_{1i}, x_{2i}, y_{2i}) \tag{2.6}$$

where (x_{1i}, y_{1i}) represent the top left corner of the bounding box and (x_{2i}, y_{2i}) its bottom right corner. Besides the target object, we also save the positions of the remaining objects present in the scene seperately for evaluation purposes. Since each bounding box is represented in pixel values, we convert them into percentages, such that the position of the bounding box is independent of the frame's resolution. Each feature vector is represented as a *Numpy* array and appended to an array of observations and labels respectively. We call the *Numpy* array of the entire observations D_{gwr} and the *Numpy* array of the entire labels L_{gwr}.

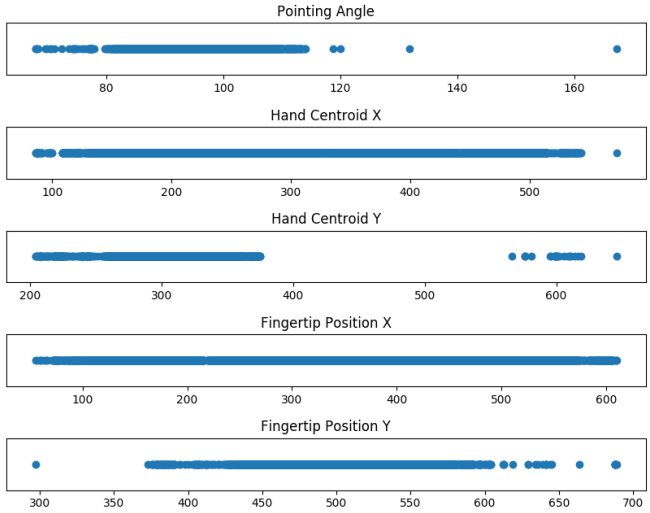
After processing all the 220 deictic gestures, we obtain 7642 observations with corresponding labels. Afterwards, we detect and remove outliers from out data. We do this by separately calculating the *standard score* of each dimension and remove any value that is more than 3.0 standard deviations from the mean. The standard score normalizes data such that it calculates the deviation of each value from the mean of the sample:

$$Z_{ij} = \frac{x_{ij} - \bar{x_i}}{S_i} \tag{2.7}$$

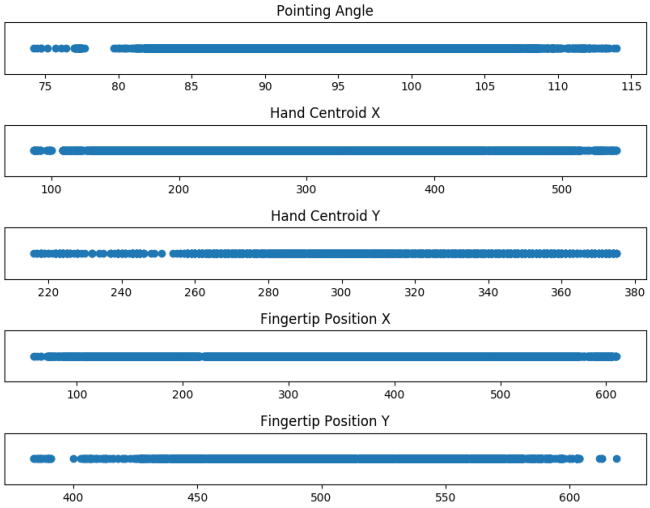
where x_{ij} is the j-th value of the i-th dimension, $\bar{x_i}$ is the mean average of the i-th dimension and S_i is the estimated standard deviation of the i-th dimension. On this basis, we can determine values that significantly deviate from the mean of the dimension and can therefore be regarded as noise. After this step, we are left with 7561 observations. The data before and after removing noise is shown in figure 2.23. Finally, we normalize the data with the following formula:

$$D'_i = \frac{D_i - min(D_i)}{max(D_i) - min(D_i)} \tag{2.8}$$

where D_i is the i-th dimension of the observations and D'_i the normalized dimension. We use this data to train our GWR.



(a)



(b)

Figure 2.23: Illustration of the result of our observation filtering process. Every observation containing a value with a standard deviation of more than 3.0 is removed. (a) shows the data before and (b) shows the data after filtering noise.

2.2.6.2 Growing When Required Network

A GWR [22] is a special form of an artificial neural network used for unsupervised learning, which means it approximates the distribution of the input data, without the data being categorized into specific classes. The GWR shows a lot of similarities to the Self-Organizing-Map (SOM) [17], as it also produces a compound of neurons and edges in the process of mapping a high-dimensional input space to lower-

dimensions (often two-dimensional). However, the big advantage of a GWR over a SOM is that it does not have a fixed network topology. This means that the number of neurons and edges in the network are not predefined and change dynamically during training, by the GWR adding and deleting nodes. This allows the GWR to fit the input distribution of the data more naturally and with less constraints. In this process, the GWR is topology preserving such that the order and structure of the input distribution is reflected by the neurons in the network. This means, each observation in the input space maps to a corresponding neuron in the network and nearby observation map to nearby neurons. This allows reasoning about the inner structure of the network and approaching high dimensional data through a two dimensional visualization. From now on, we refer to neurons as nodes.

The GWR is an attractive method for modeling the distribution of deictic gestures as it has an implicit measure for noise detection that can address the misidentified pointing intentions. Furthermore, with its functionality for novelty detection it proves to be an expandable plaform. In the following the functionality of a GWR is described. This description is closely aligned with the explanations of Marsland in Section 3 of his paper [22].

A GWR is composed of a set of nodes, a set of weight vectors and a set of edges. Each node has an associated weight vector representing its position in the input space. The edges create a neighborhood by connecting nodes representing similar perceptions. As the network is able to grow and adapt, both the edges and nodes can be created and removed. The process behind the creating and removing of network edges is called *competitive Hebbian learning* [23]. Here, for each network input, an edge is created between the best matching unit (*bmu*) and second best matching unit (*sbmu*). The *bmu* and *sbmu* are determined as the two nodes closest to the input. Each edge in the network has an age, which is initially set to zero. At each time step, the age of the nodes that are connected to the bmu are incremented by one. Except the edge between *bmu* and *sbmu*, which is reset to zero if the edge already exists. When the age of an edge exceeds a certain threshold age_{max}, the edge is deleted. A node that has lost all of its edges is removed.

With this *competitive Hebbian learning* approach a GWR is trained on a dataset. Hereby, the network iteratively adjusts to its input until it converges. The GWR extends the Hebbian learning approach by equipping each node with a firing counter. The firing counter monitors how often the node already fired since its creation. Firing in this context means becoming the *bmu* or being a neighbor of a *bmu*. The firing counter is represented by a variable that decreases exponentially from 1 to 0. This results in new nodes having a fire rate close to 1 and nodes that frequently fired will have a value close to 0.

Before describing the procedure of the GWR training, we define the variables used in the calculations of the process: Let A be the set of nodes in the GWR, and $C \subset A \times A$ be the set of edges between the nodes in the GWR. Let the input distribution be $p(\boldsymbol{\xi})$ and the individual observations $\boldsymbol{\xi}$. Further, we define $\boldsymbol{w}_n$ as

the weight vector of the node n. The network is initialized with two nodes n_1 and n_2 ($A = \{n_1, n_2\}$), created by two random observations from the input distribution. Then each observation $\boldsymbol{\xi}$ from $p(\boldsymbol{\xi})$ is successively presented to the network. For determining the *bmu* and *sbmu*, the distances between the current observation and each node in the network are calculated. These distances are obtained by the *euclidean distance* between the observation and the nodes weight vector:

$$\|\boldsymbol{\xi} - \boldsymbol{w}_i\| \tag{2.9}$$

where $\boldsymbol{w}_i$ the weight vector of the i-th node. The two nodes with the shortest distances become the *bmu* and *sbmu*:

$$bmu = arg \min_{n \in A} \|\boldsymbol{\xi} - \boldsymbol{w}_i\|, \tag{2.10}$$

$$sbmu = arg \min_{n \in A \setminus \{s\}} \|\boldsymbol{\xi} - \boldsymbol{w}_i\| \tag{2.11}$$

If there is no connection between *bmu* and *sbmu* yet, one is created:

$$C = C \cup \{(bmu, sbmu)\}, \tag{2.12}$$

otherwise the age of the existing connection is reset to zero. Afterwards, the activity of the *bmu* is calculated:

$$a = \exp(-\|\boldsymbol{\xi} - \boldsymbol{w}_{bmu}\|) \tag{2.13}$$

The activity a is a measure of how well a node matches the current input. If the *bmu* and the current observation are close together the *bmu's* activity will be close to 1. The greater the distance between the observation and its *bmu*, the lower is its activity. After obtaining the *bmu's* activity, it is compared with the activation threshold a_T (In the following, a capital T formatted as a subscript stands for a threshold value). If the activity of the *bmu* is greater a_T, the current observation is sufficiently represented. Otherwise, the current observation differs significantly from its *bmu*.

If the activity is greater a_T, the network will be trained on the current observation. In this process, the weights of the *bmu* and its neighbors are adjusted towards the observation by two learning rates e_b and e_n:

$$\Delta \boldsymbol{w}_{bmu} = e_b \times h_{bmu} \times (\boldsymbol{\xi} - \boldsymbol{w}_{bmu}), \tag{2.14}$$

$$\Delta \boldsymbol{w}_i = e_n \times h_i \times (\boldsymbol{\xi} - \boldsymbol{w}_i) \tag{2.15}$$

where $0 < e_n < e_b < 1$, e_b is the learning rate for the *bmu* and e_n is the learning rate for its neighbors. The neighborhood is defined as all nodes that are connected to the *bmu*. Further, h_s is the value of the firing counter for the *bmu* and h_i is the value of the firing counter of the *bmus* i-th neighbor. The resulting $\Delta \boldsymbol{w}_{bmu}$ and $\Delta \boldsymbol{w}_i$ are then added to the weight vector of the *bmu*and its neighbors respectively.

If, by contrast, the activity is below a_T, there can be two reasons for this. Either the node was not yet trained enough to sufficiently match the input. Or there are

not enough nodes in the network to cover the required input space. In the first case, it is reasonable to further train the node so that it can adapt to the input. In the second case, a new node must be added to the network to cover the unattended space.

To distinguish between these two cases, the nodes firing counter is consulted, since the extent of training can be equated with how often the node has fired. If the nodes firing counter value is above the firing threshold h_T, the node is seen as not sufficiently trained. Therefore, the node and its neighbors are adjusted like in formulas 2.14 and 2.15. However, if the nodes firing counter is below h_T, a new node is inserted between the *bmu* and the current observation. This is done by finding the mean average between the observation and the *bmu* weight vector:

$$A = A \cup \{r\}, \tag{2.16}$$

$$\boldsymbol{w}_r = (\boldsymbol{w}_{bmu} + \boldsymbol{\xi})/2 \tag{2.17}$$

where r is the new node added to the network and $\boldsymbol{w}_r$ its corresponding weight vector. Additionally, edges from the new node to the *bmu* and *sbmu* are added. The edge between the current *bmu* and *sbmu* is deleted. Regardless of whether a new node has been added or not, in both cases the firing counter of the bmu and its neighbors is updated afterwards:

$$h_s(t) = h_0 - \frac{S(t)}{a_b}(1 - e^{-a_b t/\tau_b}), \tag{2.18}$$

$$h_i(t) = h_0 - \frac{S(t)}{a_n}(1 - e^{-a_n t/\tau_n}) \tag{2.19}$$

where $h_i(t)$ is the size of the firing counter for node i. Further, h_0 is the initial strength and $S(t)$ is the stimulus strength, which both are usually set to 1. The variables a_n, a_b and τ_n, τ_b are constants controlling the behavior of the curve that models the firing counter. Note that a clear distinction is made here between the *bmus* firing counter and that of its neighbors. The rationale behind this is that a neighbor of the *bmu* fires not as strongly as the *bmu* itself. After adjusting the firing counter, the age of the edges between the *bmu* and its neighbors is increased by one. Then all edges with an age greater age_{max} are deleted and all nodes without any edges are removed. This whole procedure is repeated for each observation in the input distribution for several epochs, until a stopping criterion is reached.

In addition to deciding whether a new node must be added, the firing counter also fulfils other functionalities. First, the firing frequency of a node is integrated into the learning rate, such that nodes that fire often are trained less and at some point almost not trained at all. This addresses the problem that even well-trained nodes are still slightly adjusted when firing such that the network does not converge. Second, since the neighborhood of a *bmu* is also adapted while training, the firing counter in the learning rate offers a possibility of adjusting the neighborhood to a lesser extent than the *bmu* itself. Third, the firing counter can implicitly be utilized

for novelty detection. If the node never fired before or fires infrequently, the input can be considered as novel.

Both the firing counter and the insertion threshold a_T significantly influence the training and the resulting network. The firing counter has a significant impact on the network converging to a stable state. However, according to Marsland [22], the exact value of the threshold h_T does not greatly affect the behavior of the network.

The insertion threshold a_T controls when a new node is added to the network and has therefore a major influence on the network topology. The closer a_T comes to 1, the more accurately the network maps its input. This is because with a high insertion threshold, new nodes are generated even with small deviations between current observation and *bmu*. Conversely, smaller values for a_T lead to fewer generated nodes which results in the network generalizing more between similar perceptions.

2.2.6.3 Extend the GWR with Labels

The GWR is initially an unsupervised learning model. In order to predict the pointing position of previously unseen observations, we need to extend the GWR to classify incoming feature vectors. For this, we modify an approach by Parisi et al. [29], who realized action recognition with hierarchical GWRs, by equipping each node with a label after the training process. They extended the GWR algorithm with two labelling functions, one for the training phase and one for the classification of unseen observations. During training each node is provided with a histogram in which each cell represents a certain action class. Then the incidents in which a node becomes the *bmu* of a specific class are counted in that histogram. After the training phase, the class with the highest histogram value is assigned to the label of the respective node. With using this histogram, they address the alterations of the nodes during training and compensate for possible mislabellings in early training epochs. In the prediction process, the *bmu* of the current observation returns its label, which classifies the current observation.

In contrast to our approach, Parisi et al. [29] classify a finite set of discrete symbolic actions. Since we have a set of continuous pointing positions as labels, our approach faces different constraints and requirements, as these pointing positions are incompatible with a discrete histogram representation. Therefore, similar to Parisi et al. [29], we need to define a way for the label to reflect the changes in the weight of the node so that the mapping between weight and label is not distorted during training.

To achieve this, we exploit the fact that the weights and label are both continuous *points* in a n-dimensional space. The weights are 5-dimensional vectors as they inherit their shape from the training data. The label is an annotated bounding box, represented by 2-dimensional position vectors. This allows us to apply methods

altering the weight vectors one-to-one to the vectors that represent the target position of the deictic gesture. Thereby, the alterations in the labels will follow the alterations in the weights and therefore map their topology preserving properties. This means, that as long the initial weight to node mapping is correct, similar pointing positions in the input distribution will be represented by the labels of nodes that are close together in the network.

To implement the changes in the node's labels, we distinguish between the creation and training of a node. When a node is created, in addition to the calculating the mean average between the observation and the $bmu's$ weight (see 2.17), we also calculate the mean average of both labels and assign it to the new node. Since the label is represented by a bounding box, we define the mean of two bounding boxes as the mean of their centroids, heights and widths, respectively:

$$lc_r = (lc_{bmu} + lc_\xi)/2 \tag{2.20}$$

where lc_r is the centroid of the new node's label, lc_{bmu} is the centroid of the $bmu's$ label and lc_ξ is the centroid of the label of the current observation. Afterwards, we create a new bounding box around the calculated centroid lc_r with the new width and height. Next, we must also address the adjustments of the nodes weights when no new node is added. For this purpose, we incorporate the learning rates of the bmu and its neighbors to the adjusting of the nodes labels. This is done exactly as in the case of adjusting the weights, only that we exchange the 5-dimensional weight vectors with the 2-dimensional bounding box centroids.

$$\Delta lc_{bmu} = e_b \times h_{bmu} \times (lc_\xi - lc_{bmu}), \tag{2.21}$$

$$\Delta lc_i = e_n \times h_i \times (\boldsymbol{l}c_\xi - lc_i) \tag{2.22}$$

where lc_{bmu} is the centroid of the $bmu's$ label, lc_i is the label's centroid of the $bmu's$ i-th neighbor and lc_ξ is the centroid of the label of the current observation. Similar to the adjusting of the weights, the resulting Δlc_{bmu} and Δlc_i are added to the centroids of the bmu and its neighbors respectively. Like when a new node is added, the average of the height and width of the bounding boxes are calculated and applied to the trained bounding boxes. This adjusting of the labels has the benefit that it exactly matches the alterations of the labels. So, if a node is already trained well, neither its weight nor its label will change significantly.

2.2.6.4 Predicting with the GWR

After equipping its nodes with labels, the GWR gains the ability to predict the pointing position of yet unseen observations. Based on this information, we aim to identify the object that the experimenter is targeting. To this end, we first check if the current object constellation is ambiguous. Depending on that, we adapt the procedure for identifying the gesture's target. This distinction allows us to model the ambiguities present in the scene, which benefits both the accuracy

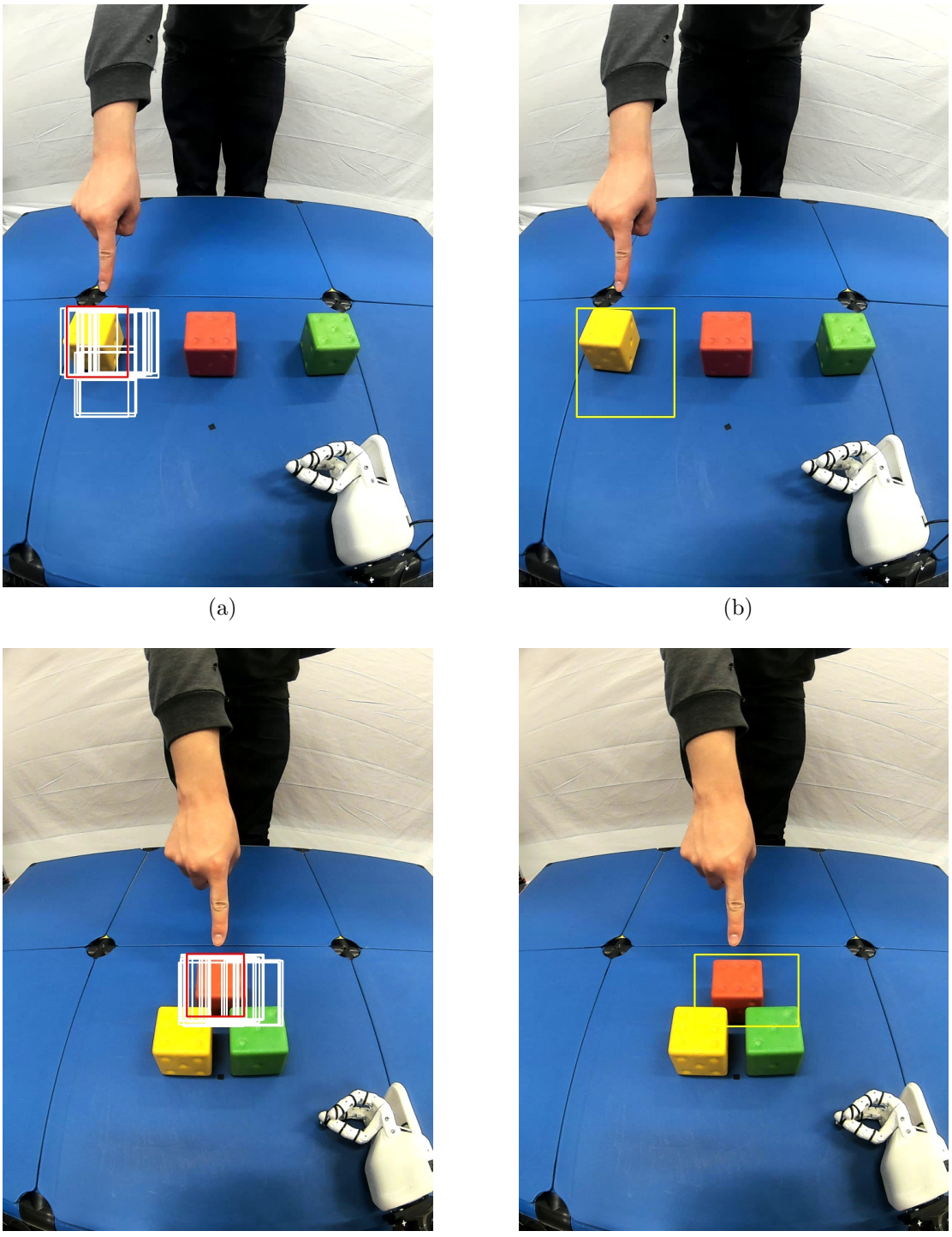
of our approach and the fidelity of the subsequent evaluation. In the following, we explain how we detect ambiguities and identify targeted objects based on the predicted pointing positions from the GWR.

The process of predicting the pointing position of an observation itself is very straightforward. We take the trained GWR and calculate the *bmu* of the current observation. If the activation of the *bmu* is below a noise threshold n_T, the observation is considered noise and filtered out. Otherwise, the *bmu* returns its label, which is the prediction for the pointing position. With implementing the noise threshold n_T, we address one drawback of the computer vision-based approach described in section 2.2.5, where misidentified pointing intentions are immediately converted into a pointing array. Since such a pointing is not intended by the experimenter, it will most likely not be part of the distribution of the gestures from the training data. Therefore, the nodes of the GWR will not represent this pointing and the activation of the corresponding input will be comparatively low. We empirically determined n_T to be 0.5, as the activation of most noisy input ranges from 0.1 to 0.35 on average.

For detecting whether the current object constellation is ambiguous, we need to examine more of the scene than one single pointing position offers us. This is because each predicted pointing position is approximately the same size as an object itself, as they originate from the annotated bounding boxes from the training data. In order to increase the area for detecting ambiguities, we get a set of "*best matching units*" from the GWR and calculate the union of their labels, that we call A. These labels, represent the respective positions the GWR regards as most similar to the pointing position of the current observation. As illustrated in figure 2.24, these positions cluster around the label of the GWR's *bmu* and thus cover a wider range of positions. Calculating the union of these positions enlarges the area for detecting ambiguities as desired. For calculating A, we get the minimum and maximum value of the x- and y-coordinates across all bounding boxes' corner *points* in the set of the received labels. We define the top left corner of A as (min_{x1}, min_{y1}) and its top right corner as (max_{x2}, max_{y2}) (see figure 2.25). Thus, A represents the area including all pointing positions. To determine the exact number of labels we get from the GWR to calculate A, we introduce an empirically set-up function F_A

$$F_A(\#nodes) = \lceil \#nodes * 0.01 + 5 \rceil \tag{2.23}$$

where $\#nodes$ is the number of nodes of the GWR and the values 0.01 and 5 are empirical constants. With introducing F_A, we vary the number of labels we receive from the GWR in relation to its size. This addresses the issue that if this number remains constant, the size of area A correlates negatively with the network size. This mean, area A becomes smaller as the GWR grows and vice versa. This is due to the fact that the distribution of nodes becomes denser as the network grows. Therefore, the variation in the obtained pointing positions decreases, which negatively affects the size of area A. In order to keep the size of A as constant as possible, we consider the size of the GWR.



(a) (b)

(c) (d)

Figure 2.24: Illustration of our ambiguity detection process. (a) and (b) show cases where no ambiguity is detected, (c) and (d), by contrast, show object constellations that are recognized as ambiguous.

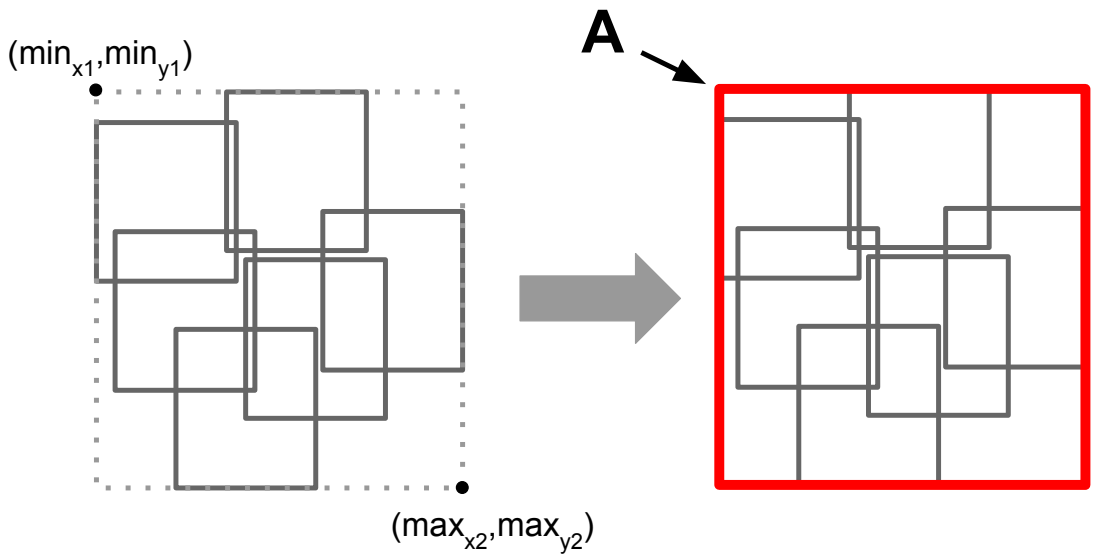


Figure 2.25: Illustration of the estimation of area A for detecting ambiguities. (a) presents the bounding boxes estimated by function F_A (see function 2.23) and the two corner *points* (min_{x1}, min_{y1}) and (max_{x2}, max_{y2}) determined by them. (b) shows the area of union of these bounding boxes, which we call A.

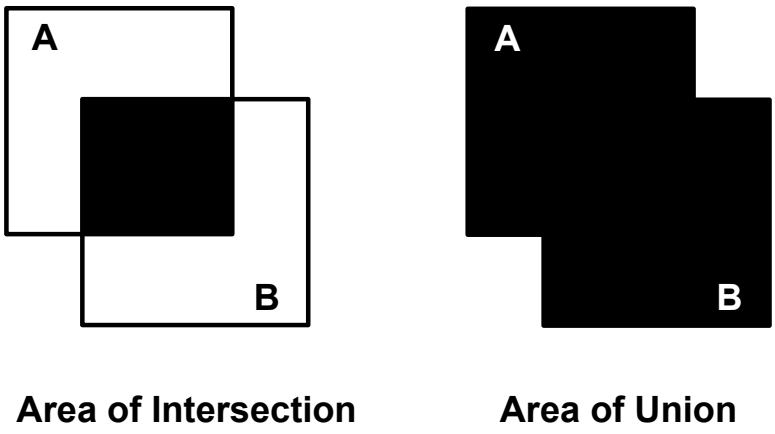


Figure 2.26: Example of the *area of intersection* and *area of union* for calculating the *Intersection Over Union* (see formula 2.24) of two bounding boxes **A** and **B**.

After obtaining A, we detect all objects in this area, using the object detection described in section 2.2.3. If we recognize more than one object in A, we consider these objects to be close to each other and therefore ambiguous. Otherwise, we regard the constellation as unambiguous. With introducing the ambiguity detection we address two further drawbacks of the computer vision based approach, namely the requirement of detecting all objects a priori and the modeling of pointings

without a goal. By determining A, the ambiguity detection puts the gesture in relation to a cluster of possible pointing positions. This allows us to significantly reduce the area we have to consider for detecting objects, as we assume the object to be in A. This makes object detection more controlled and efficient, as we get an informed estimate of the gesture's target before we detect objects. Thus, situations that contain a lot of objects have less influence on the object detection's performance. In addition, modeling the gesture's target and the concomitant reduction in search space gives us the necessary control to address pointings which seem to have no target at all. This may include to further enlarge the area A or to allow NICO to respond to the gesture. Some of these ideas will be discussed in the *Future Work* section (see section 4.1).

After checking for an ambiguity, we are faced with the task of identifying the targeted object. We adapt this procedure depending on the results of the ambiguity detection. If we do not detect an ambiguity, we either found exactly one or no object in A. If we find no objects in the scene, we consider this a miss. In the case of detecting exactly one object, we save its color and position and define it as the target of the gesture. By contrast, if we detect an ambiguity, we need to resolve the object constellation to determine the targeted object. For this, we recourse on the GWR's actual *bmu*, as this is the most precise estimation we have. In this situation, we have two to three detected objects and one predicted pointing position in A. Now we need to apply a metric that decides which object is best matched by the predicted pointing position. Since we compare different positions represented by bounding boxes, we employ a metric from the field of object detection called *Intersection Over Union* (IoU). The IoU is illustrated in figure 2.26 and divides the area of intersection of two shapes by their area of union.

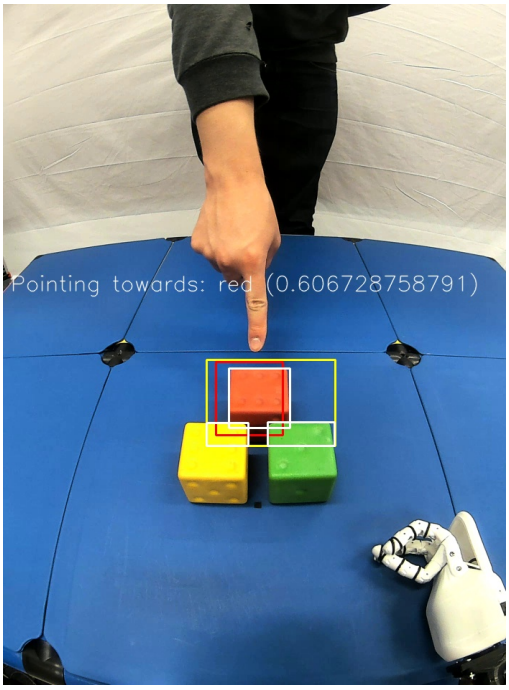
$$IoU = \frac{Area\,of\,Intersection}{Area\,of\,Union} \tag{2.24}$$

The IoU estimates the similarity of two shapes and penalizes when the union of both shapes is larger than their intersection. Note that this definition of union differs from our definition above. In the case of calculating the IoU, the area of union is defined as the area of both shapes minus their intersection.

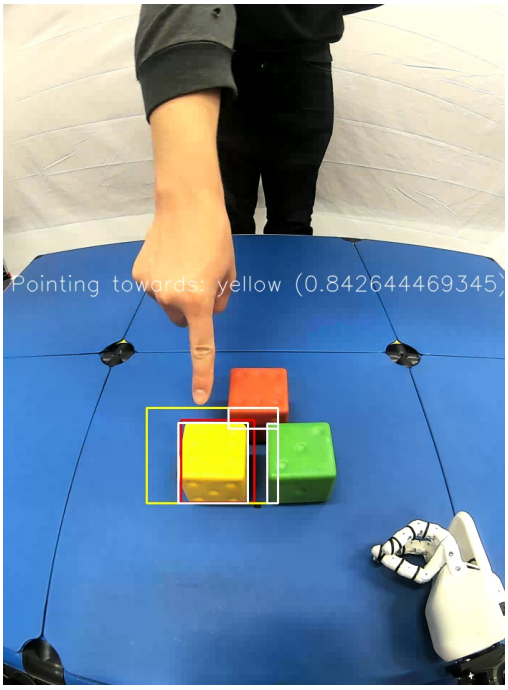
We now calculate the IoU between the GWR's prediction and every detected object in A. If the IoU between the prediction and an object is ≥ 0.5, we consider the object as the target. In the unlikely case that two objects have an $IoU \geq 0.5$, we regard the object with the highest IoU the target of the gesture. If no object has an $IoU \geq 0.5$, we consider this a miss. Again, the color and bounding box of the object identified as the target is saved. Here it quickly becomes apparent why the ambiguity detection, besides modeling ambiguities in the scenario, also improves the prediction accuracy in lower ambiguity classes. The area A prevents the bmu from being used in situations where it is not necessary and thus reduces the risk of the target object being missed.

The reason why the IoU threshold is surprisingly low at 0.5, is that inaccuracies

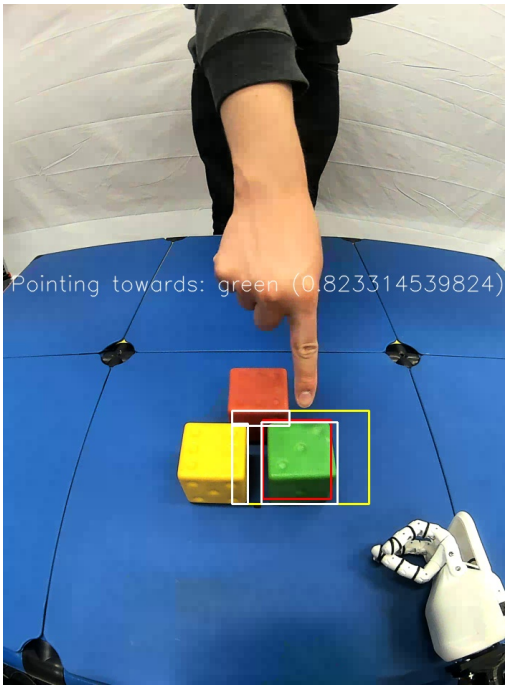
in both the object detection and the pointing position are taken into account. In addition, the IoU threshold of 0.5 is widely used in the literature for evaluating object detection and object recognition approaches [8].



(a)



(b)



(c)

Figure 2.27: Exemplification of the predicting with the GWR when an ambiguity is detected. (a), (b) and (c) show a pointing to one object respectively. The yellow bounding box represents area A, the white boxes the detected objects and the red bounding box is the label of the observation's *bmu*. The number behind the color of the target object is the respective *Intersection Over Union*.

Chapter 3

Experiments, Results and Evaluation

In the last chapter, we introduced two approaches for identifying the target of a deictic gesture. First, we approximated a pointing array from the segmented shape of the hand (see section 2.2.5). Second, we learned the target position of deictic gestures by quantifying certain hand features and training a Growing When Required network (GWR)(see section 2.2.6). In this chapter we will evaluate these two approaches against each other and present our results. In the following, we first describe and motivate the evaluation metrics we use and then successively detail on the evaluation of both approaches.

3.1 Evaluation Metrics

For being able to compare the models behind the two approaches, we need to find evaluation metrics that can be applied to both. Although the two models differ in their structure, they underlie the same baseline scenario in which the experimenter performs a deictic gesture to an object. The task of the models is to identify this *gesture-object* mapping and to predict the target object. This prediction can result in three possible outcomes:

1. The model returns the object the experimenter is targeting
2. The model returns an object the experimenter is not targeting
3. The model returns no object at all

For being able to quantify our results, we represent these outcomes as *true positive* (*TPs*), *false positive* (*FPs*) and *false negative* (*FNs*). These categories compare the

actual target of the deictic gesture with the prediction of the model. We consider the first case a *TP*, as the targeted object and the model's prediction coincide. The second case is a *FP*, since the model falsely predicts an object that is not targeted. However, we also regard the second case as FN, because the actual target object was falsely not predicted. This makes the third case a *FN* as well. These three categories provide the foundation for our evaluation.

3.1.1 Precision and Recall

To get a better overview, over the different predictions, we further summarize the *TPs*, *FPs* and *FNs* by introducing *precision* and *recall*. *Precision* is defined as the number of *TPs* divided by the sum of *TPs* and *FPs*.

$$precision = \frac{TP}{TP + FP} \tag{3.1}$$

Precision describes how many *gesture-target* mappings are correctly identified, with the prerequisite that a target is identified at all (case 1 & 2). By contrast, *recall* is defined as the *TPs* divided by the sum of *TPs* and *FNs*.

$$recall = \frac{TP}{TP + FN} \tag{3.2}$$

Recall describes how many gestures are correctly recognized with respect to all gestures and, thus, also considers cases where no target is detected at all. *Precision* and *recall* in combination possess instructive information, as they reflect how precise the model is when a target is identified and how many targets are identified from the complete population.

3.1.2 F_1-Score

Since we want to evaluate different models against each other, comparability is an important criterion for the selection of evaluation metrics. Unfortunately, *precision* and *recall* alone are cumbersome to interpret as the information of the model's performance is divided on two values. Especially when comparing multiple models, the constant pondering between different *precision-recall* pairs hinders the decision making process and may lead to erroneous conclusions. Therefore, we strive for a metric that allows the intuitive comparison of different models by representing their performance with a single value. To this end, we consult the f_1-*score*, which is the harmonic mean of *precision* and *recall*.

$$f_1\text{-}score = 2 * \frac{precision * recall}{precision + recall} \tag{3.3}$$

Besides being more intuitive to interpret, the f_1-*score* summarizes both values more balanced and meaningful as it punishes low values of either precision or

recall. This prevents models that perform very well in one metric and very poorly in the other from obtaining a satisfactory result on average. In the following, the *precision*, *recall* and f_1-*score* are the metrics we emphasize the most during the evaluation of our approaches.

3.2 Evaluation of the Computer Vision based Approach

In the following we describe the evaluation of the computer vision based approach. To this end, we first take a closer look at how the requisite data is collected. Afterwards, we outline the set-up and steps of the evaluation itself. Finally, we present and analyze our results.

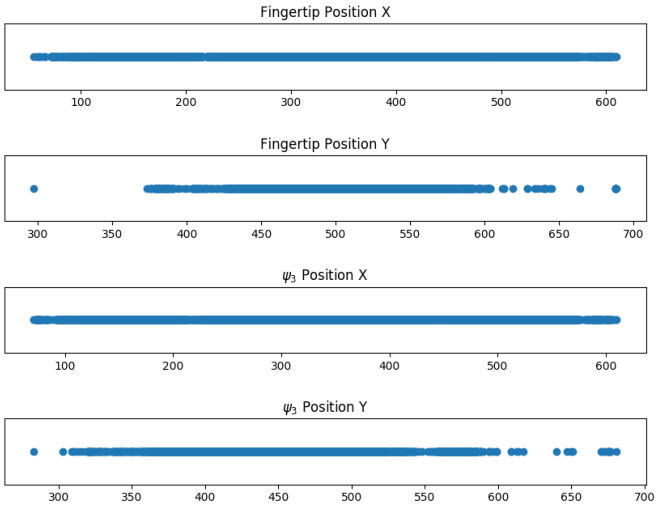
3.2.1 Data acquisition

Regarding the acquisition of data, we use the same annotated data and extraction procedure as described in section 2.2.6.1 with the exception that the content of the resulting feature vector changes. We fill the vector with the hand properties required for calculating the pointing array. These include the x- and y-coordinates of the fingertip position ρ and the x- and y-coordinates of the medial point on the index finger ψ_{13} (see figure 2.20). For a better overview, we refer to ψ_{13} as ψ from now on. For each extracted frame from the recordings we obtain a feature vector Cv_i

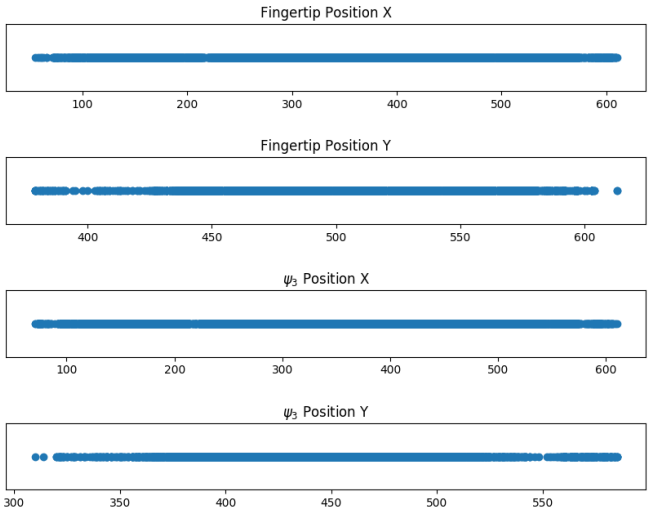
$$Cv_i = (\rho_{xi}, \rho_{yi}, \psi_{xi}, \psi_{yi}) \tag{3.4}$$

where ρ_{xi} and ρ_{yi} are the x- and y-coordinates of ρ and ψ_{xi} and ψ_{yi} the x- and y-coordinates of ψ, of the i-th observation. This leaves us with a *Numpy* array of 7680 observations which we refer to as D_c. Like in section 2.2.6.1, we also obtain the filename of the corresponding recording, the ambiguity class of the object constellation and the color of the targeted object. The latter is used as the observation's label because we evaluate the observation by comparing the color of the targeted and predicted object.

After collecting the data, we detect and remove outliers in the same way as we did with equation 2.7. For each dimension, we remove each value that is more than 3.0 standard deviations from the the dimensions mean. This results in 7628 observations. The data before and after the filtering are shown in figure 3.1. Since this data is saved in pixel values, we convert them into percentages so that the positions are independent of the resolution of the frames to which the model is applied.



(a)



(b)

Figure 3.1: Illustration of the results of filtering outliers in the data for the computer vision based approach. Every observation containing a value with a standard deviation of more than 3.0 is removed. (a) shows the distributions of the individual data dimensions before and (b) after filtering noise.

3.2.2 Evaluation Set-up

In the final evaluation process, we iterate through all 7628 observations and predict the object the experimenter is targeting by following the steps described in section 2.2.5. For each observation, we get its corresponding filename, ambiguity class and color of the target object. The filename gives us the recording from which the

observation originated. We take the recording's 40-th frame and detect all objects by using the object detection described in section 2.2.3. We choose the 40-th frame as the experimenter reaches a resting position around that time and color irritations that occur in the beginning of the recording become less significant. This frame provides us with the position and color of each object present in the scene. Then, we take the values of the current observation and calculate the pointing array as described in section 2.2.5.1. Equipped with the pointing array and detected objects, we follow the steps of section 2.2.5.2 to predict the target object. Then we compare the color of this object with target object's color and classify the observation accordingly in *TPs*, *FPs* or *FNs*. If the color of the predicted object is equal to the color of the label we consider the observation a *TP*. If the colors are different, we count the observation both as a *FP* and *FN*. In the case where no object is predicted, we classify the observation as a *FN* and count it as a *miss*. These results are saved with regard to all observations and the applicable ambiguity class. After all observations are processed, precision, recall, f_1-*score* and percentage of the *misses* are calculated for the total count and each ambiguity class individually.

3.2.3 Results and Evaluation

After completing the evaluation process, we receive the results presented in tables 3.1 and 3.2. While table 3.1 contains the raw measurements, table 3.2 presents the final evaluation metrics described in section 3.1.

Counts of Evaluation Process					
	TP	*FP*	*FN*	*Misses count*	*Total*
a_1	1248	0	20	20	1268
a_2	1428	4	119	115	1547
a_3	1356	6	55	49	1411
a_4	2414	203	984	781	3398
Total	6446	213	1178	965	7624

Table 3.1: Counts of *TPs*, *FPs*, *FNs* and *misses* with respect to all observations and each ambiguity class.

Calculated Evaluation Metrics				
	Precision (%)	*Recall (%)*	F_1*-score (%)*	*Misses (%)*
a_1	100.0	98.42	99.21	1.58
a_2	99.72	92.31	95.87	7.43
a_3	99.56	96.1	97.8	3.47
a_4	92.24	71.04	80.27	22.96
Total	96.8	84.55	90.26	12.65

Table 3.2: Calculated *precision*, *recall*, f_1*-score* and percentage of *misses* with respect to all observations and each ambiguity class.

The first noticeable aspect is the high *precision* with an overall value of 96.8%. So, when a target object is estimated, it is a *TP* with high probability. This is especially the case for the ambiguity classes a_1 to a_3, since the precision always approaches 100%. This applies similarly to the *recall* and *misses*, which also show positive results in the first three ambiguity classes. This is reflected by a high f_1*-score* that ranges from 95.87% to 99.21%. However, in ambiguity class a_4 this picture suddenly changes, as the *misses* escalate from its prior maximum of 7.43% to 22.96%. This also affects the *recall*, which drastically drops to 71.04%. Besides the *misses*, also the *FPs* increase from 6 incidents in a_3 to 213 in a_4, which brings the precision down to 92.24%.

This particular behavior can be explained by two factors. First, the objects in a_4 are very close together, such that variations in the pointing array are more likely to cause a non-targeted object to be *hit*. This most likely accounts for the increase in *FPs*. Second, we observed a moderate yellow cast in the recordings from ambiguity class a_4. Since the hand detection solely relies on a learned color distribution, it is very susceptible to alterations of the colors of the recordings. Therefore, in the frames where the yellow cast appears more heavily, the segmentation performance is negatively affected. This introduces additional variation to the pointing array, which causes it to strongly deviate from its intended direction. Therefore, the probability of the pointing array to intersect with the targeted object significantly decreases.

In conclusion, the computer vision-based approach has produced positive results and is capable to recognize the target of deictic gestures in most cases. Here especially the excellent performance in the ambiguity classes a_1 to a_3 stands out. However, the results from a_4 show how dependent the approach is on a well-functioning hand segmentation.

3.3 Evaluating the GWR based Approach

To tackle some shortcomings of the computer vision based approach we learned the distribution of deictic gestures by quantifying certain features of the hand and training a GWR (see section 2.2.6). To evaluate this approach, we first provide details on the training process, including different GWR parameter configurations we consider and the structure of the resulting networks. After describing the training, we outline the procedure for evaluating the approach quantitatively. To this end, we first evaluate the performance of the GWR alone and thereafter the performance of the GWR based prediction process described in section 2.2.6.4. Finally, we evaluate the prediction process with the GWR in a qualitative manner, by using recordings that are not part of the acquired data.

3.3.1 Training Process

Before describing the evaluation set-up, we first outline the process behind the training of the GWR and the parameters we use. According to Marsland et. al [22], the most significant parameter for the GWR's performance is the value of the insertion threshold a_T. Since a_T governs how many nodes are added to the network, it has a considerable influence on how well the input is represented. To determine the number of nodes that are necessary for a good performance, we test three different values for a_T, namely 0.85, 0.90 and 0.95. Thereupon, it is interesting to investigate how long a network with the corresponding value for a_T has to train in order to achieve a certain performance. For this reason, we train the networks over different numbers of *epochs*. An *epoch* is all the observations in the training data cycling through the network once. To cover a broad range, we observe a GWR's behavior over 30, 50 and 100 *epochs*. For keeping the network configurations combinatorially manageable, we set the other parameters required by the GWR to constant values that we do not change. In table 3.3, the values of these parameters are shown.

Fixed training parameters									
e_b	e_n	h_T	τ_b	τ_n	a_b	a_n	h_0	age_{max}	$neigh_{max}$
0.1	0.01	0.1	0.3	0.1	1.05	1.05	1.0	200	6

Table 3.3: Parameters for training the GWR that are fixed among the different networks.

Here, e_b is the learning rate for the *best matching unit* (*bmu*), e_n is the learning rate for the *bmu's* neighbors, h_T is the firing threshold and τ_b, τ_n, a_b, a_n, h_0 are the parameters for habituating the *bmu* and its neighbors. The parameters age_{max} and

$neigh_{max}$ specify the maximum age of an edge before deletion and the maximum number of neighbors a node can have.

For training and testing the GWR, we use the data described in section 2.2.6.1. In order to obtain resilient results, we employ the k-fold cross-validation technique. This technique divides the data into k complementary subsets of equal size, called *folds*. During the process, $k-1$ of these *folds* are used for training and one *fold* for testing. The model is then trained k-times, with using each subset for testing exactly once. This creates k results for each model. Subsequently, the mean average and standard deviation for the k-results of the model's test *folds* are calculated. This provides insights on the stability of the model's overall performance.

In this manner, we train and test each a_T and *epoch* permutation by dividing our data into three folds, which results in three trained networks per permutation. Since we have nine different permutations, we obtain 27 differently trained networks after the training process. To provide an intuition on the resulting networks after training, we show one network of each parameter configuration in table 3.4. Besides the number of nodes and edges, the table also contains an *error* metric that indicates how well the network represents the training observations. The *error* is calculated every *epoch* and is the mean average of the current distances of all observations to its *bmu*. The better the GWR covers the input space, the lower is the overall *error*. The final *error* of a GWR is obtained after the last *epoch*.

GWR Networks				
a_T	*Epochs*	# Nodes	# Edges	Error
0.85	30	263	633	0.0597
	50	332	861	0.0557
	100	448	1217	0.0522
0.90	30	541	1422	0.0461
	50	677	1797	0.0422
	100	890	2409	0.0394
0.95	30	1543	4206	0.0268
	50	1930	5222	0.0261
	100	2606	7263	0.0250

Table 3.4: Number of nodes and edges generated with our GWR network for different values of a_T and number of *epochs*. The resulting *error* is also presented.

The resulting networks in table 3.4 reflect the properties of the GWR as expected, since with an increasing value for a_T, more nodes and edges are added during training. Additionally, it is noticeable that the *error* decreases as the number of nodes increases. In our example, however, the GWRs do not stop growing at a

certain point. Therefore, their size also correlates with the number of *epochs* over which the network was trained. This contrasts with the results of Marsland et al. [22], whose GWRs converged to a stable state, where no further nodes and edges are added. However, Marsland only used low-dimensional data in his experiments and thus one explanation for this behavior is that our data is more complex.

Nevertheless, as our GWRs do not stop growing, we cannot use a converged network as an indication for a completed training. We must, therefore, determine by other means when the network has grown to an appropriate size. Appropriate in this context means compromising between a sufficient performance without adding unnecessary nodes, that cause the network to lose its ability to generalize. To approach this trade-off, we need to assess the GWR with its evaluation metrics (see section 3.1), as the information available during training provide little or no information on its final performance. This is due to the fact that the metrics that reflect the training process, such as the *error*, are entirely determined by the weights of the nodes. However, the performance of our GWRs are not determined by the nodes' weights, but by their labels. Consequently, the error of the network keeps decreasing as new nodes are added, without providing any insight on the current performance. Without evaluating the GWR, it remains inconclusive at which network size a certain performance is reached and at which point the following added nodes can be regarded as redundant. Therefore, we will address the question of an appropriate network size later during the evaluation of the GWR.

3.3.2 Quantitative Evaluation

After training the different GWRs, we evaluate and compare their performance. To provide a comprehensive picture of both the GWR individually and its prediction procedure, the evaluation process is twofold. First, we evaluate the trained GWRs with the annotated pointing positions provided by the data introduced in section 2.2.6.1. Here, we particularly emphasize the question of how many nodes a GWR requires for a good performance. Afterwards, we exactly follow the process described in section 2.2.6.4 by incorporating the object, ambiguity and noise detection. In this way, we aim to compare the influence of the ambiguity detection with the sole performance of the GWR. In addition, we focus on evaluating which levels of ambiguity the GWR is capable to resolve. After describing both evaluation processes we present and analyze our results.

3.3.2.1 Evaluating the GWR's Performance

When evaluating the GWR's performance individually, we solely use the data described in section 2.2.6.1, without further methods being employed. For representing the positions of the objects in the scene, we use the annotated pointing positions. Note that the pointing positions from the data also represent object

positions.

For each observation in the current test *fold*, we obtain the position of the target object as well as the positions of the other objects in the scene. Subsequently, we predict the pointing position of the current observation, by receiving the label of its *bmu* in the GWR. The following procedure for predicting the target object is similar to that described in section 2.2.6.4, in the case that an ambiguity is detected. We calculate the *Intersection Over Union* (IoU) (see figure 2.26) of the prediction and each annotated object position. If the IoU of both is ≥ 0.5, we consider the object *hit*. Then we compare the *hit* object with the observation's target and classify the result in *TPs*, *FPs*, *FNs* and *misses*.

Based on these values, we then calculate *precision*, *recall*, f_1-*score* and the percentage of *misses* for all observations and each ambiguity class respectively. In addition, we store the average IoU between the prediction and target positions with respect to all observations. We do not divide the $IoUs$ further into ambiguity classes, as they are not influenced by the particular object constellation. Since we evaluate the networks by using 3-fold cross validation, we calculate the mean average and standard deviation of the different results after all three test *folds* are processed. This leaves us with mean average and standard deviation pairs of IoU, *precision*, *recall*, f_1-*score* and *miss* percentage.

3.3.2.2 Evaluation of the Predicition Process

After individually evaluating the GWR, we integrate it with the object, ambiguity and noise detection to assess the prediction procedure described in section 2.2.6.4. In the beginning, we follow the same procedure as described in section 3.2.2, until we obtain the 40-th frame of the recording and the color of the target object. From there we calculate the *bmu* of the current observation and examine its activation. If the activation is below the noise threshold n_T, we regard the observation as noise and do not process it further. If the activation is above n_T, we check if the object constellation is ambiguous.

For this, we use the obtained frame of the scene to detect all objects in the area A. If we do not detect an ambiguity, we have either detected one object or no object at all. In the first case, we get the color of the object and compare it with the target color from the data. If both colors coincide, we consider the observation a *TP*. Otherwise we count the observation as both a FP and FN. If no object is detected, we consider the observation a *FN* and a *miss*. By contrast, if we detect an ambiguity, we determine the target object by examining the IoU of the observation's *bmu* label and each object position in A. Then we classify the observation in *TP*, *FP* and *FN* the same way as when no ambiguity is detected. After processing all observations, we calculate *precision*, *recall*, f_1-*score* and the percentage of *misses*. Furthermore, we calculate the mean average over all IoUs between the prediction from the GWR and the position of the detected target

object.

In addition, we evaluate the accuracy of the ambiguity detection by comparing its result with the actual ambiguity class assigned to each observation. For this purpose, we classify the ambiguity classes a_1 and a_2 as unambiguous and the classes a_3 and a_4 as ambiguous. If the result of the ambiguity detection and the actual class coincide, the ambiguity detection is regarded successful. Finally, we evaluate the performance of the noise detection. Since every observation in the data represents a correct pointing, we consider every filtered noise as falsely identified. As in the last section, we save these values with respect to all observations and each corresponding ambiguity class. Except for the IoU, which we only calculate for all observations. Additionally, we also calculate the mean average and standard deviation of the results, since we also use 3-fold cross validation.

3.3.2.3 Results and Evaluation

In the following, we present, analyze and compare the results we obtain from both evaluation set-ups described in the two previous sections. First, we discuss the performance of the GWR alone, by referring to its results presented in the tables 3.5, 3.6 and 3.7. Here, we attach particular importance to determining the minimum number of nodes that our GWR requires to achieve a good performance. We then concentrate on the evaluation of the entire prediction process, whose results can be found in the tables 3.8 to 3.13. We investigate to what extent the ambiguity detection increases the performance of the GWR in the lower ambiguity classes. In general, we also focus on the performance in higher ambiguity classes to assess what levels of ambiguity our approach is capable to solve.

The tables in this section present the results achieved with the different parameter configurations described in the training process (see section 3.3.1). Each table shows three GWRs, which are trained with the same value for the activation threshold a_T, over 30, 50 and 100 *epochs*. The results of the individual metrics are described both with regard to the applicable ambiguity class ($a_1, \ldots, a_4$) and to the overall observations. Since we use 3-fold cross validation for evaluation, each table cell displays both the mean average and the standard deviation across the test *folds*.

a_T : 0.85, Individual GWR Evaluation				
30 Epochs, IoU:0.677 ± 0.004				
	Precision (%)	*Recall (%)*	F_1-*score (%)*	*Misses (%)*
a_1	100.0 ± 0.0	74.43 ± 2.2	85.32 ± 1.4	25.57 ± 2.2
a_2	100.0 ± 0.0	87.23 ± 1.5	93.17 ± 0.8	12.77 ± 1.5
a_3	99.76 ± 0.0	88.6 ± 3.1	93.82 ± 1.8	11.18 ± 3.1
a_4	100.0 ± 0.0	90.69 ± 1.0	95.11 ± 0.6	9.31 ± 1.0
Total	99.95 ± 0.0	86.89 ± 0.2	92.97 ± 0.1	13.07 ± 0.2
50 Epochs, IoU:0.685 ± 0.005				
a_1	100.0 ± 0.0	80.73 ± 8.0	89.12 ± 4.9	19.27 ± 8.0
a_2	100.0 ± 0.0	89.2 ± 1.6	94.28 ± 0.9	10.8 ± 1.6
a_3	99.92 ± 0.1	84.26 ± 3.8	91.38 ± 2.3	15.68 ± 3.8
a_4	100.0 ± 0.0	90.82 ± 0.5	95.19 ± 0.3	9.18 ± 0.5
Total	99.98 ± 0.0	87.62 ± 2.3	93.38 ± 1.3	12.37 ± 2.3
100 Epochs, IoU:0.713 ± 0.001				
a_1	100.0 ± 0.0	85.12 ± 5.2	91.88 ± 3.0	14.88 ± 5.2
a_2	100.0 ± 0.0	90.87 ± 1.0	95.21 ± 0.5	9.13 ± 1.0
a_3	99.91 ± 0.1	87.57 ± 1.3	93.33 ± 0.7	12.35 ± 1.4
a_4	100.0 ± 0.0	94.23 ± 1.8	97.02 ± 1.0	5.77 ± 1.8
Total	99.99 ± 0.0	90.79 ± 1.1	95.17 ± 0.6	9.19 ± 1.0

Table 3.5: Results of GWRs with a_T : 0.85.

a_T : 0.90, Individual GWR Evaluation				
30 Epochs, IoU:0.725 ± 0.0				
	Precision (%)	*Recall (%)*	F_1-*score (%)*	*Misses (%)*
a_1	100.0 ± 0.0	91.03 ± 2.2	95.29 ± 1.2	8.97 ± 2.2
a_2	100.0 ± 0.0	94.46 ± 1.1	97.15 ± 0.6	5.54 ± 1.1
a_3	100.0 ± 0.0	96.67 ± 0.5	98.31 ± 0.2	3.33 ± 0.5
a_4	100.0 ± 0.0	95.34 ± 0.1	97.62 ± 0.0	4.66 ± 0.1
Total	100.0 ± 0.0	94.68 ± 0.2	97.27 ± 0.1	5.32 ± 0.2
50 Epochs, IoU:0.736 ± 0.002				
a_1	100.0 ± 0.0	92.77 ± 1.7	96.24 ± 0.9	7.23 ± 1.7
a_2	100.0 ± 0.0	95.37 ± 1.1	97.63 ± 0.6	4.63 ± 1.1
a_3	100.0 ± 0.0	95.92 ± 1.1	97.92 ± 0.6	4.08 ± 1.1
a_4	100.0 ± 0.0	97.53 ± 0.2	98.75 ± 0.1	2.47 ± 0.2
Total	100.0 ± 0.0	95.99 ± 0.4	97.95 ± 0.2	4.01 ± 0.4
100 Epochs, IoU:0.76 ± 0.004				
a_1	100.0 ± 0.0	93.75 ± 2.4	96.76 ± 1.3	6.25 ± 2.4
a_2	100.0 ± 0.0	95.08 ± 2.0	97.47 ± 1.0	4.92 ± 2.0
a_3	100.0 ± 0.0	98.18 ± 0.7	99.08 ± 0.3	1.82 ± 0.7
a_4	100.0 ± 0.0	97.32 ± 0.4	98.64 ± 0.2	2.68 ± 0.4
Total	100.0 ± 0.0	96.42 ± 0.4	98.17 ± 0.2	3.58 ± 0.4

Table 3.6: Results of GWRs with a_T : 0.90.

a_T : 0.95, Individual GWR Evaluation				
30 Epochs, IoU:0.808 ± 0.001				
	Precision (%)	*Recall (%)*	F_1-*score (%)*	*Misses (%)*
a_1	100.0 ± 0.0	97.63 ± 0.6	98.8 ± 0.3	2.37 ± 0.6
a_2	100.0 ± 0.0	97.8 ± 0.3	98.89 ± 0.2	2.2 ± 0.3
a_3	100.0 ± 0.0	98.62 ± 0.3	99.31 ± 0.1	1.38 ± 0.3
a_4	100.0 ± 0.0	99.17 ± 0.2	99.59 ± 0.1	0.83 ± 0.2
Total	100.0 ± 0.0	98.55 ± 0.2	99.27 ± 0.1	1.45 ± 0.2
50 Epochs, IoU:0.813 ± 0.001				
a_1	100.0 ± 0.0	97.03 ± 1.3	98.49 ± 0.7	2.97 ± 1.3
a_2	100.0 ± 0.0	97.61 ± 0.4	98.79 ± 0.2	2.39 ± 0.4
a_3	100.0 ± 0.0	98.47 ± 0.8	99.23 ± 0.4	1.53 ± 0.8
a_4	100.0 ± 0.0	99.19 ± 0.5	99.59 ± 0.2	0.81 ± 0.5
Total	100.0 ± 0.0	98.36 ± 0.3	99.17 ± 0.1	1.64 ± 0.3
100 Epochs, IoU:0.824 ± 0.001				
a_1	100.0 ± 0.0	97.24 ± 0.6	98.6 ± 0.3	2.76 ± 0.6
a_2	100.0 ± 0.0	98.53 ± 1.0	99.26 ± 0.5	1.47 ± 1.0
a_3	100.0 ± 0.0	99.44 ± 0.3	99.72 ± 0.2	0.56 ± 0.3
a_4	100.0 ± 0.0	99.38 ± 0.3	99.69 ± 0.2	0.62 ± 0.3
Total	100.0 ± 0.0	98.86 ± 0.3	99.43 ± 0.1	1.14 ± 0.3

Table 3.7: Results of GWRs with a_T : 0.95.

The first aspect that immediately attracts attention when observing the individual performance of the GWR is that the *precision* is always close to 100% across all parameter constellations. This means, when a target object is predicted, it is almost guaranteed to be a *TP*. In fact, *FPs* barely exist in our resuls. *Misses*, on the other hand, occur more frequently and constitute the vast majority of the *FNs*. This is because a prediction that produces a *FP* has an $IoU \geq 0.5$ with a non-target object, which only happens when a gesture is severely misinterpreted. A *miss*, in contrast, already occurs when the prediction's IoU with the target object is < 0.5. Therefore, a *FP* is harder to achieve than a *miss* and consequently occurs less often.

Another aspect that immediately becomes apparent is that the metrics consistently show a small standard deviation across the different parameter configurations. Only the networks with a a_T of 0.85, trained over 30 *epochs*, show a significant increase in the ambiguity classes a_1 and a_3. In general, the standard deviation ranges from 0.0% to 2.3% with regard to the total observations. However, most metrics show a value between 0.0 and 0.6. This shows that the results of the networks are stable and resilient.

When considering the *recall*, an interesting observation is that it is positively affected by an increasing activation threshold a_T. This is reasonable, as the higher the value for a_T is, the more nodes are created and the better the input is represented. This results in a better fit between the predicted pointing position and

the actual target, but reduces the network's ability to generalize. This observation is in line with a decreasing percentage of *misses* and an increasing overall IoU. Since a_T significantly influences these values, it is inevitably also a strong predictor of the overall performance of the networks. This is comprehensible when comparing the networks f_1-*score* across the respective tables, that steadily increments as a_T increases. For instance, the networks trained over 50 *epochs*, show an average f_1-*score* of 93% with $a_T = 0.85$, then reach 97% when $a_T = 0.90$ and rise to 99% when $a_T = 0.95$. However, as outlined in figure 3.4, networks that are trained with a high value for a_T generate significantly more nodes than networks with a lower a_T. At a certain point, the amount of additionally generated nodes does not reflect the network's increment in performance anymore. This becomes obvious when comparing the networks trained with an $a_T = 0.90$ and $a_T = 0.95$ over 30 *epochs*, where the first network generates 541 nodes and the latter 1543 while only performing 2% better. Therefore, the additional accumulated nodes of the latter network can be regarded as unnecessary as they do not contribute to a better modeling of the input distribution and moreover abolish any generalization abilities of the GWR. This raises the question of how many nodes a network needs to perform well without growing to an unreasonable extent.

To answer this question, we first analyze how long a network needs to be trained by examining how the number of training *epochs* affects the overall performance. When considering the f_1-*scores*, the number of *epochs* has only a small influence, as these increase only by a comparatively small amount. This indicates that the GWR already generated most of the required nodes before the 30th *epoch* and the remaining nodes are only added sporadically until the 50th or 100th *epoch* is reached. Here the question of the network's convergence from section 3.3.1 is addressed, as it can be assumed that the network reaches a stable state even before the 30th *epoch*. Therefore we regard 30 *epochs* as the optimal training duration among the available parameter configurations.

Now we compare the performance and number of nodes of these networks across the different values for a_T. Since we already found out that the network with an a_T of 0.95 produces too many nodes, we focus on the a_T values 0.85 and 0.90. Here it is noticeable that the network with an $a_T = 0.90$ produces less than half of the *misses* (13.07% vs. 5.32%) and shows a substantial better recall (86.89% vs. 94.68%). This is reflected in a f_1-*score* of 97.27% for the network with an $a_T = 0.90$ compared to 92.97% for an $a_T = 0.85$. In addition, the IoU of an a_T of 0.90 is increased by 0.048 compared to that of an a_T of 0.85 (from 0.677 to 0.725). Although the network with an a_T of 0.90 generates slightly more than twice as many nodes as the other one (263 vs. 541), considering the convincing improvements, we argue that these additional nodes are added meaningfully. We therefore consider the network with an a_T that equals 0.90 and that is trained over 30 *epochs* as the overall best performing network, since it embodies the best trade-off between performance and generalization. Based on these findings, we focus our evaluation of the entire prediction process on the networks trained over 30 *epochs* and especially

emphasize the parameter configuration we regard as best performing. Additionally, we will not consider the networks with an a_T of 0.95 anymore, as they are no longer interesting to evaluate due to the amount of unreasonably generated nodes.

When evaluating the entire prediction process, we aim to assess the influence of the newly introduced methods with particular emphasis on the ambiguity detection. For this purpose, the tables representing the evaluation metrics are divided into two parts. The first part presents the familiar performance metrics of the prediction process, while the second part displays metrics concerning the ambiguity and noise detection. The metrics of the second part display the percentages of the correctly detected ambiguities and falsely detected noise. As already described in the prediction process (see section 3.3.2.2), we regard the ambiguity classes a_1, a_2 as unambiguous and a_3, a_4 as ambiguous. Every time the result of the ambiguity detection and the actual ambiguity class coincide, the ambiguity detection is regarded correct. This set-up allows us to compare the performance of the prediction process with that of the GWR alone while assessing the influence of the ambiguity detection. In terms of the noise detection, we observe if any FPs are produced, as we have no noisy observations in our data to be filtered out.

Before considering the performance metrics, we first look at the noise detection and find that it did not produce any *FP*, since no observation was considered as noise across the networks. We also notice that, compared to the GWR alone, the prediction results also show a small standard deviation. In fact, the standard deviations of the metrics became considerably more stable as they now range from 0.0% to 0.7%.

a_T : 0.85, Evaluate Prediction with GWR				
30 Epochs, IoU:0.691 ± 0.003				
	Precision (%)	*Recall (%)*	*F_1-score (%)*	*Misses (%)*
a_1	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	0.0 ± 0.0
a_2	100.0 ± 0.0	94.6 ± 0.8	97.22 ± 0.4	5.4 ± 0.8
a_3	98.54 ± 1.0	85.0 ± 0.5	91.27 ± 0.3	13.72 ± 1.3
a_4	100.0 ± 0.0	85.95 ± 1.6	92.44 ± 0.9	14.05 ± 1.6
Total	99.75 ± 0.2	89.84 ± 0.5	94.54 ± 0.3	9.93 ± 0.4
50 Epochs, IoU:0.7 ± 0.002				
a_1	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	0.0 ± 0.0
a_2	100.0 ± 0.0	95.39 ± 0.7	97.64 ± 0.4	4.61 ± 0.7
a_3	99.43 ± 0.6	82.7 ± 3.8	90.26 ± 2.5	16.84 ± 3.3
a_4	99.97 ± 0.0	85.41 ± 2.0	92.11 ± 1.2	14.56 ± 2.0
Total	99.9 ± 0.1	89.37 ± 0.9	94.34 ± 0.6	10.54 ± 0.8
100 Epochs, IoU:0.715 ± 0.004				
a_1	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	0.0 ± 0.0
a_2	100.0 ± 0.0	96.33 ± 0.3	98.13 ± 0.1	3.67 ± 0.3
a_3	99.85 ± 0.2	85.6 ± 3.4	92.14 ± 2.0	14.27 ± 3.5
a_4	100.0 ± 0.0	87.92 ± 3.1	93.55 ± 1.7	12.08 ± 3.1
Total	99.97 ± 0.0	91.2 ± 1.6	95.38 ± 0.9	8.77 ± 1.7

Table 3.8: Results of the GWR's prediction process with $a_T : 0.85$.

a_T : 0.85, Ambiguity and Noise Detection		
30 Epochs		
	CDA (%)	*FDN(%)*
a_1	100.0 ± 0.0	0.0 ± 0.0
a_2	70.88 ± 1.8	0.0 ± 0.0
a_3	97.12 ± 1.4	0.0 ± 0.0
a_4	93.89 ± 0.8	0.0 ± 0.0
Total	90.93 ± 0.2	0.0 ± 0.0
50 Epochs		
a_1	99.59 ± 0.6	0.0 ± 0.0
a_2	70.51 ± 3.3	0.0 ± 0.0
a_3	97.69 ± 0.7	0.0 ± 0.0
a_4	93.53 ± 1.6	0.0 ± 0.0
Total	90.73 ± 1.0	0.0 ± 0.0
100 Epochs		
a_1	99.38 ± 0.9	0.0 ± 0.0
a_2	69.39 ± 0.4	0.0 ± 0.0
a_3	98.85 ± 1.3	0.0 ± 0.0
a_4	95.05 ± 0.2	0.0 ± 0.0
Total	91.36 ± 0.2	0.0 ± 0.0

Table 3.9: Results of the ambiguity and noise detection with $a_T : 0.85$.

a_T : 0.90, Evaluate Prediction with GWR				
30 Epochs, IoU:0.713 ± 0.0				
	Precision (%)	*Recall (%)*	*F_1-score (%)*	*Misses (%)*
a_1	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	0.0 ± 0.0
a_2	100.0 ± 0.0	98.33 ± 0.4	99.16 ± 0.2	1.67 ± 0.4
a_3	100.0 ± 0.0	91.5 ± 1.3	95.56 ± 0.7	8.5 ± 1.3
a_4	100.0 ± 0.0	90.77 ± 1.9	95.15 ± 1.0	9.23 ± 1.9
Total	100.0 ± 0.0	93.94 ± 0.6	96.88 ± 0.3	6.06 ± 0.6
50 Epochs, IoU:0.719 ± 0.002				
a_1	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	0.0 ± 0.0
a_2	100.0 ± 0.0	98.55 ± 0.5	99.27 ± 0.2	1.45 ± 0.5
a_3	100.0 ± 0.0	89.63 ± 0.1	94.53 ± 0.1	10.37 ± 0.1
a_4	99.97 ± 0.0	93.16 ± 0.9	96.44 ± 0.5	6.81 ± 0.9
Total	99.99 ± 0.0	94.71 ± 0.3	97.28 ± 0.2	5.28 ± 0.3
100 Epochs, IoU:0.735 ± 0.002				
a_1	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	0.0 ± 0.0
a_2	100.0 ± 0.0	98.44 ± 1.2	99.21 ± 0.6	1.56 ± 1.2
a_3	99.92 ± 0.1	90.25 ± 2.5	94.82 ± 1.4	9.68 ± 2.5
a_4	100.0 ± 0.0	93.76 ± 0.8	96.78 ± 0.4	6.24 ± 0.8
Total	99.99 ± 0.0	95.09 ± 0.7	97.48 ± 0.4	4.89 ± 0.7

Table 3.10: Results of the GWR's prediction process with a_T : 0.90.

a_T : 0.90, Ambiguity and Noise Detection		
30 Epochs		
	CDA (%)	*FDN(%)*
a_1	100.0 ± 0.0	0.0 ± 0.0
a_2	72.8 ± 1.5	0.0 ± 0.0
a_3	99.41 ± 0.5	0.0 ± 0.0
a_4	94.63 ± 1.1	0.0 ± 0.0
Total	92.06 ± 0.8	0.0 ± 0.0
50 Epochs		
a_1	100.0 ± 0.0	0.0 ± 0.0
a_2	72.52 ± 0.3	0.0 ± 0.0
a_3	98.65 ± 1.0	0.0 ± 0.0
a_4	96.48 ± 0.9	0.0 ± 0.0
Total	92.71 ± 0.4	0.0 ± 0.0
100 Epochs		
a_1	100.0 ± 0.0	0.0 ± 0.0
a_2	72.36 ± 1.6	0.0 ± 0.0
a_3	98.68 ± 0.4	0.0 ± 0.0
a_4	95.47 ± 1.5	0.0 ± 0.0
Total	92.22 ± 0.8	0.0 ± 0.0

Table 3.11: Results of the ambiguity and noise detection with a_T : 0.90.

a_T : 0.95, Evaluate Prediction with GWR				
30 Epochs, IoU:0.754 ± 0.002				
	Precision (%)	*Recall (%)*	*F_1-score (%)*	*Misses (%)*
a_1	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	0.0 ± 0.0
a_2	100.0 ± 0.0	99.19 ± 0.3	99.6 ± 0.1	0.81 ± 0.3
a_3	99.79 ± 0.2	97.33 ± 0.4	98.54 ± 0.1	2.47 ± 0.5
a_4	100.0 ± 0.0	98.0 ± 0.4	98.99 ± 0.2	2.0 ± 0.4
Total	99.96 ± 0.0	98.45 ± 0.2	99.2 ± 0.1	1.51 ± 0.3
50 Epochs, IoU:0.756 ± 0.001				
a_1	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	0.0 ± 0.0
a_2	100.0 ± 0.0	99.14 ± 0.6	99.57 ± 0.3	0.86 ± 0.6
a_3	100.0 ± 0.0	96.46 ± 0.7	98.2 ± 0.4	3.54 ± 0.7
a_4	100.0 ± 0.0	98.16 ± 0.7	99.07 ± 0.4	1.84 ± 0.7
Total	100.0 ± 0.0	98.33 ± 0.3	99.16 ± 0.1	1.67 ± 0.3
100 Epochs, IoU:0.762 ± 0.001				
a_1	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	0.0 ± 0.0
a_2	100.0 ± 0.0	99.34 ± 0.3	99.67 ± 0.2	0.66 ± 0.3
a_3	100.0 ± 0.0	97.08 ± 0.2	98.52 ± 0.1	2.92 ± 0.2
a_4	100.0 ± 0.0	98.08 ± 0.1	99.03 ± 0.1	1.92 ± 0.1
Total	100.0 ± 0.0	98.47 ± 0.1	99.23 ± 0.1	1.53 ± 0.1

Table 3.12: Results of the GWR's prediction process with a_T : 0.95.

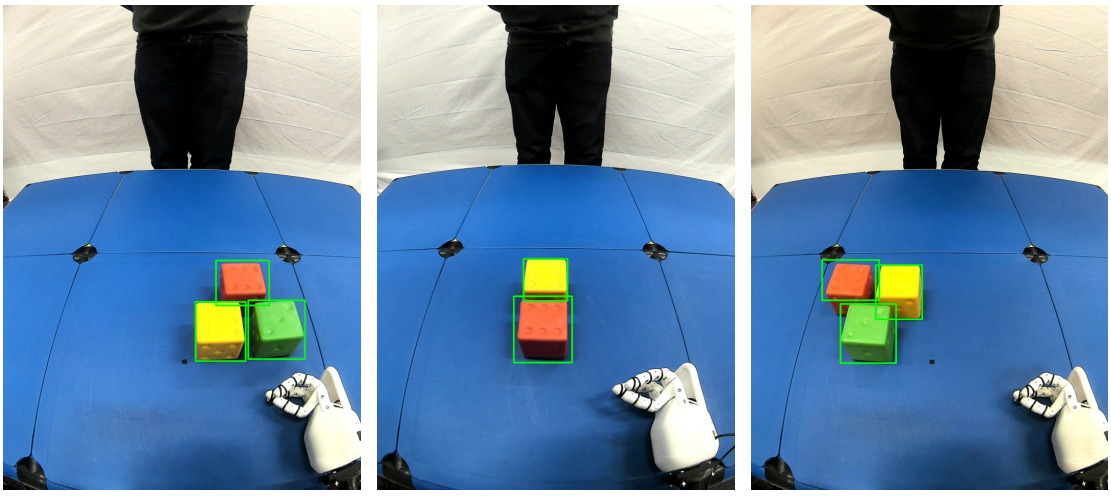
a_T : 0.95, Ambiguity and Noise Detection		
30 Epochs		
	CDA (%)	*FDN(%)*
a_1	100.0 ± 0.0	0.0 ± 0.0
a_2	85.73 ± 1.4	0.0 ± 0.0
a_3	99.8 ± 0.3	0.0 ± 0.0
a_4	94.07 ± 1.2	0.0 ± 0.0
Total	94.45 ± 0.3	0.0 ± 0.0
50 Epochs		
a_1	100.0 ± 0.0	0.0 ± 0.0
a_2	80.82 ± 0.9	0.0 ± 0.0
a_3	100.0 ± 0.0	0.0 ± 0.0
a_4	94.49 ± 1.3	0.0 ± 0.0
Total	93.7 ± 0.7	0.0 ± 0.0
100 Epochs		
a_1	100.0 ± 0.0	0.0 ± 0.0
a_2	78.4 ± 1.0	0.0 ± 0.0
a_3	100.0 ± 0.0	0.0 ± 0.0
a_4	96.83 ± 0.5	0.0 ± 0.0
Total	94.27 ± 0.0	0.0 ± 0.0

Table 3.13: Results of the ambiguity and noise detection with a_T : 0.95.

The first feature that immediately catches the eye when examining the performance of the prediction step is that every observation in ambiguity class a_1, is correctly predicted across all networks. This results in both a precision and recall of 100% for the object constellations of a_1. In addition, the ambiguity detection classifies the ambiguity class a_1 in 100% of the cases correctly. The ambiguity detection thus completely eliminated the *misses* that were the main cause for the GWR's false predictions. With that, the prediction performance in ambiguity class a_1 heavily improved.

Regarding ambiguity class a_2, the performance metrics have also improved considerably. When looking at the networks trained over 30 *epochs*, the *misses* dropped from 12.77% to 5.4% for an a_T of 0.85 and for an a_T of 0.90 from 5.54% to 1.67%. This is further reflected by the *recall* and f_1-*scores* of the networks. The performance of the ambiguity detection reduces to around 70%. This is due to the objects in a_2 being closer together compared to a_1. Thus, the area A is more likely to include more than one object.

Despite the improvements in the ambiguity classes a_1 and a_2, it is noticeable that the predictions in a_3 and a_4 yield considerably worse results than achieved by the GWR alone. Besides the f_1-*score*, this is especially reflected in the the *misses* and IoU. In the case of the network, we regard as best performing, the *misses* increase from 3.33% to 8.5% in a_3 and 4.66% to 9.23% in a_4 and the IoU decreases from 0.725 to 0.713. Since both the IoU and the *misses* are affected, it can be assumed that this drop in performance is due to a less accurate match between the predicted target position of the GWR and the object position. Since the labels of the GWR remained the same, only the object representation has changed due to the introduced object detection. A decline in performance because of the object



(a) (b) (c)

Figure 3.2: Variation in the size of the bounding box of the detected object in ambiguity class a_4.

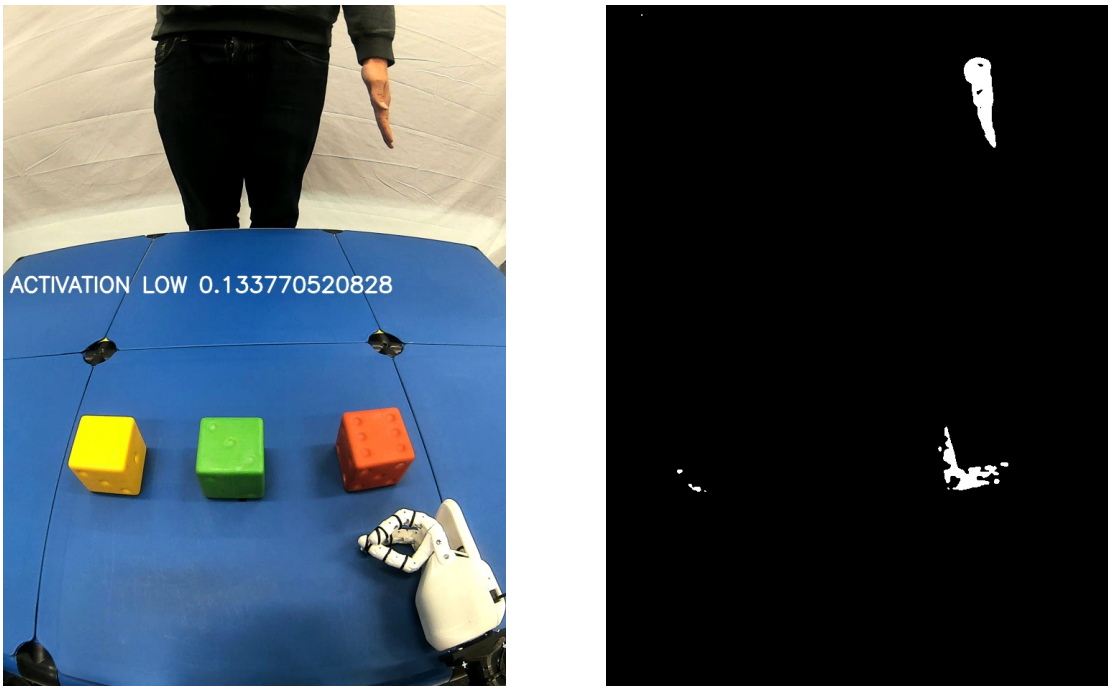
detection is plausible, since the size and position of the estimated bounding boxes are not as accurate as the manually annotated labels. Therefore, the detected object may not necessarily correspond to the predicted target position of the GWR anymore. Furthermore, in a_3 and especially a_4 the objects occlude each other, which leads to smaller visible object areas. This causes smaller bounding boxes around some detected objects, as the object detection solely relies on the color cues visible in the image. This variation in the object size is shown in figure 3.2. Both of these factors have a negative effect on the *IoU*. The lower ambiguity classes are not affected by this, since here not the label of the *bmu*, but the estimated area A is used to determine the target object. Unlike the computer vision based approach, the GWR does not seem to be affected by the yellow cast, as the GWR alone shows a good performance.

After examining the individual ambiguity classes, we now analyze the overall performance of the networks trained over 30 *epochs*. When considering the f_1-*score* and *recall*, we observe mixed results when comparing them with the performance of the GWR alone. While the *recall* and f_1-*scores* both increased in the case of $a_T = 0.85$ (86.89% vs. 89.84%; 92.97% vs. 94.54%), they showed a slight decrease when considering the a_T equal to 0.90 (94.68 vs. 93.94%, 97.27% vs. 96.88%). Here it becomes apparent that despite the considerable improvements in the ambiguity classes a_1 and a_2, the overall performance is not significantly positively affected, due to the negative influence of the object detection. Nevertheless, even after this decline in performance, the GWR still produces reliable results over all ambiguity classes.

In conclusion, both the GWR alone and the prediction process are well capable of recognizing deictic gestures across the ambiguity classes. Especially the ambiguity detection provides excellent results and succeeds in eliminating the vast majority of the *misses* in the lower ambiguity classes. Although the performance in the higher classes declined in some cases, positive results were achieved nevertheless. Furthermore, we found that the best performing GWR has 541 nodes and is trained over 30 *epochs* with an a_T of 0.90. Although the object detection negatively affects the performance of the prediction process due to its naive implementation, the overall performance remains noticeable.

3.3.3 Qualitative Evaluation

After quantitatively evaluating the prediction process, we assess further aspects of the GWR by employing recordings that are not part of our training and testing data. With this, we aim to provide insights into how the GWR behaves in certain edge case scenarios. In particular, we focus on evaluating the noise detection, with which the filtering of incorrectly identified pointing intentions is achieved. In addition, we investigate how the GWR behaves when confronted with deictic gestures that target areas outside the previously learned pointing positions. For



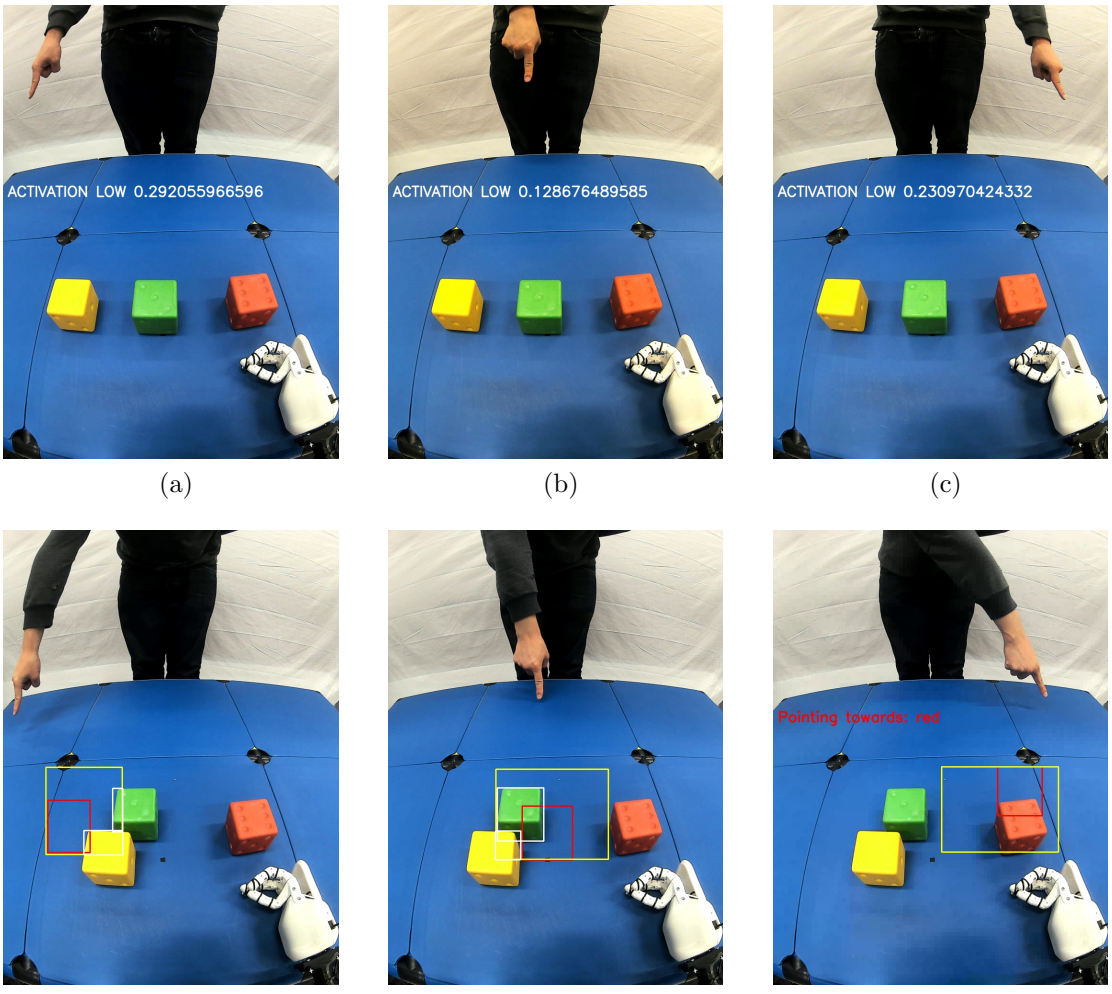
(a) (b)

Figure 3.3: Demonstration of noise detection filtering out misidentified pointing intentions resulting from ambiguous hand poses. (a) shows an example for such a hand pose resulting in a misidentified pointing intention but is filtered out by the noise detection. (b) shows the corresponding binary image in which the hand posture looks similar to one with an extended index finger.

these purposes, we use the GWR we regard as best performing in the last section ($a_T = 0.90$, 30 *epochs*).

During the quantitative evaluation, we already discovered that the noise detection does not generate any *FPs* at all. So, if an observation belongs to the distribution of deictic gestures, it is almost certainly not erroneously filtered out. To evaluate the GWR's ability to filter actual noisy observations, we observe 6 recordings in which the experimenter gesticulates with his hands, while mimicking hand poses that lead to inadvertently detected pointing intentions. These poses resemble a hand with an extended index finger in the resulting binary image after hand detection. Such a hand position is shown in figure 3.3.

When observing the recordings, we notice that when these ambiguous hands shapes occur in the upper part of the frame, the hand posture is correctly considered as noise. The further the hand moves down towards the table, the greater is the probability that it is regarded a deictic gesture. This is because the closer the hand gets to the region where the gestures usually have their *peak*, the higher is the activation of the corresponding observation. Therefore, it is arguable that the y-position of both the fingertip and the centroid of the hand's feature vector significantly influ-



(a) (b) (c)

(d) (e) (f)

Figure 3.4: Illustration of the GWR's learned topology of deictic gestures. The experimenter points at positions on the table that are not reflected by the labels of the GWR data. (a), (b) and (c) show pointings filtered out due to a low activation. (d), (e), and (f) show pointings towards the table areas right in front of the experimenter. Here, the GWR returns the learned positions spatially closest to the gesture's actual target. The GWR is, therefore, capable of generalizing the observation's characteristics in such a way that it returns the nearest known position in space.

ence its activation. Since gesticulation that we regard as natural only takes place at abdomen and waist level, the noise detection was able to detect and filter the vast majority of incorrectly identified pointing intentions. The activation of these hand postures ranges from about 0.08 to 0.35.

With another 10 recordings, we investigate the effects when the experimenter targets positions, that are not yet represented by the GWR. We first analyze gestures aiming at more distant regions on the table, but all of them were considered as

noise due to the position of the hand. This can be observed in the parts (a), (b) and (c) of figure 3.4. In contrast, when the experimenter targets the table areas directly in front of him, the activation exceeds the noise threshold and the prediction step of the GWR is executed. In this way, the *bmus* of the current observation are activated, which naturally lead to the prediction of already learned pointing positions. Interestingly, as illustrated in (d), (e), and (f) of figure 3.4, the GWR does not return a random position but the position closest to the gesture's actual target. Although the GWR has not yet seen this particular *gesture-target* mapping, it is able to abstract from it and returns the next known target position in space. This suggests that the GWR has learned a comprehensive model of the topological structure of deictic gestures in this scenario.

3.4 Comparison of Computer Vision- and GWR based Approach

After the evaluation, we now compare the prediction process of the computer vision- and the GWR based approach. For the computer vision approach, we consider the results of the tables 3.1 and 3.2. In terms of the GWR, we regard the results of our best performing network ($a_T = 0.90$, 30 *epochs*), which can be found in the tables 3.10 and 3.11.

First of all, both approaches share a high value for precision. While the GWR exhibits an overall value of 100%, the computer vision approach shows 96.8%. Thus, they share the property of producing only few *FPs*, although it is notable that the the GWR produces none at all. When regarding the *recall*, the GWR displays a significantly higher overall value (93.94%), than the computer vision approach (84.55%). This deviation is due to the fact that the latter produces more than twice as many *misses* (6.06% vs. 12.65%). These result from a severe performance loss in a_4, where the misses increase to 22.96% causing the *recall* to drop to 71.04%. By contrast, the *recall* of the GWR never falls below 90%. When comparing the overall performance by consulting the f_1-*score*, we see the GWR with 96.88% to perform significantly better than the computer vision based approach with a value of 90.26%. In the first three ambiguity classes, however, when solely considering the average of the f_1-*score* (98.10% vs. 97.63%), the GWR approach performs only slightly better. Thus, when ignoring the results in a_4, the computer vision approach performs similar to the GWR in terms of the sole evaluation metrics.

However, the results of the computer vision approach in a_4 suggest that it is very susceptible to noise and confounding factors. In contrast, the GWR, whose training data originate from the same recordings, shows no such substantial decline in performance. The ambiguity detection introduces notable stability to the prediction of target objects in unambiguous object constellations. This is reflected in the fact that the GWR perfectly predicts all observations in ambiguity class a_1. In contrast

to the computer vision approach, the GWR actually addresses the ambiguity of the respective object constellation. Hence, in contrast to the computer vision approach, the GWR actually addresses the ambiguity of the respective object constellation. Furthermore, when considering the results of the qualitative evaluation in section 3.3.3, the GWR is capable to detect and filter misidentified pointing intentions with a conspicuous accuracy. While the computer vision approach immediately extrapolates a pointing array, the prediction with the GWR is significantly more controlled.

Chapter 4

Discussion

The aim of this thesis is to implement and evaluate a natural deictic gesture interface for the NICO robot. Beside modeling a dynamic *gesture-target* mapping, we especially emphasize on addressing the ambiguities in deictic gestures. In order to achieve this, we design a scenario with adaptable object positions in which an experimenter performs deictic gestures towards objects arranged in front of the NICO robot. To implement, test and evaluate the insterface, we take several recordings in that scenario with varying object constellations of different levels of ambiguity. In terms of implementing the gesture interface, we first detect and track the experimenter's hand by modeling the distribution of skin color in the scene. We then detect the referenceable objects on the basis of their color information. Subsequently, we process the shape of the detected hand and model its current state to determine when the experimenter intends to perform a deictic gesture. Finally, we introduce two approaches for estimating the deictic gesture's target. In the first approach, we approximate a pointing array based on the hand shape's geometrical properties. In the second approach, we learn the distribution of deictic gestures by quantifying expressive features of the hand shape while pointing and training a Growing When Required network (GWR). After evaluating our results, we are confident that both approaches are well suited for the recognition of deictic gestures. However, it becomes apparent that the computer vision-based approach is very susceptible to confounding factors. By contrast, the GWR proves to be more robust and significantly more capable of resolving ambiguous object constellations. Due to the implemented ambiguity detection, the GWR also achieves a high performance in lower ambiguity classes. In the following, we discuss several aspects of our approach for recognizing deictic gestures and put its capabilities into context with related work.

Towards the end of our introduction (see section 1.3) we raise the question whether a stable deictic gesture interface that fulfills the mentioned goals can be realized without the use of depth sensors. This question emerged, since the majority of other approaches, with the exception of Richarz et al. [36], utilize depth information.

After evaluating our approaches, we can confidently answer this question with yes. We substantiate this by regarding the results of our best performing GWR ($a_T = 0.90$, 30 *epochs*), that predicted the target of 93.94% of all gestures correctly. Thus, we show that a lightweight computer vision-based approach can achieve high performance without employing expensive geometric calculations of a Kinect™ v2, for example. However, it should be noted that this performance is accomplished in a controlled laboratory environment. Particularly, since our hand and object detection are solely based on color information, they are highly susceptible to noise and cannot immediately be applied to other environments.

During the scenario design, one goal we particularly emphasize is the creation of a natural interaction. Naturalness is especially facilitated if the interaction reflects human interaction patterns as accurately as possible. Canal et. al. [6] detect a deictic gesture if the angles between the detected body joints remain within predefined threshold ranges for a certain amount of time. This is suboptimal, since the experimenter has to adapt his behavior to these artificial restrictions. In contrast, we define our pointing intention as pointing with the index finger to model an interaction pattern people regard as familiar. If we look at Stiefelhagen's approach [26], we find that a magnetic sensor attached to the head is used to determine the head rotation. We regard this as unnatural, since the requirement of mounting the sensor hinders an accessible interaction. In contrast, we solely rely on computer vision techniques, as we want to realize an interaction that does not require external devices. Nonetheless, it can be argued that our approach also restricts naturalness to a certain extent, since the position of the experimenter is fixed throughout the entire interaction and the objects are arranged in a predefined space.

Another goal of this thesis is to implement an adaptable interface that allows variable object positions to realize a dynamic *gesture-target* mapping. Stiefelhagen et al. [26] and Park et al. [30] both model the direction of the deictic gesture and the object positions with angle values. Each object present in the scene is assigned a fixed angle interval. If the estimated angle that represents the pointing direction falls within the interval of a specific object, the object is considered targeted. Since the angle intervals must not overlap with this method, the corresponding approaches cannot process ambiguous object constellations. They are not able to realize a dynamic *gesture-target* mapping because the objects can not be positioned variably in the scene. In contrast, we do not only model the direction or position of the gesture, but also employ a measure of quality to determine which object is *hit*. This puts the gesture into relation to the targeted object.

Having adaptable object positions constitutes the basis for realizing the thesis' overarching goal of addressing the ambiguities of deictic gestures. Different levels of ambiguities can be modeled and evaluated. Canal et al. [6] also address the ambiguities of deictic gestures in their approach. However, they do not resolve an ambiguity by means of their gesture recognition, but with the robot verbally asking for further instructions. Although asking for feedback is reasonable, our approach differs in the sense that we seek to resolve ambiguities with our estimated

pointing position. Furthermore, Canal [6] does not cover a large variety of object constellations, which results in a less detailed evaluation. By contrast, we create 95 different object constellations that divide themselves into four different levels of ambiguity. This allows us to cover a wide range of possible arrangements and enables a particularized evaluation of our approach.

Upon completion of this evaluation, we can determine that the GWR is able to predict the correct target object across all level of ambiguities in most cases. Here we again look at the GWR we regard as best performing ($a_T = 0.90, 30$ *epochs*) and observe the f_1-*scores* of the ambiguity classes ($a_1 : 100\%$, $a_2 : 97.22\%$, $a_3 : 91.27\%$, $a_4 : 92.44\%$). Since the standard deviation of the results is also small (ranges from 0.0% to 1.0%), we can state that we achieved robust and reliable results even in object constellations with high ambiguity. Further it is noticeable that integrating the ambiguity detection into our prediction process achieves to almost completely eliminate the *misses* in the lower ambiguity classes. This demonstrates that by considering ambiguity, we can resolve the clear constellations more effectively. Good performance is also achieved in the high ambiguity classes, but here the detected ambiguity is no longer taken into account since the GWR plainly returns the label of the observation's *bmu*. This results in the prediction decoupling from the object constellation such that the ambiguity no longer affects the overall performance. The modeling of ambiguity therefore only benefits the unambiguous object constellations. On the one hand, this is reasonable since ambiguous object constellations are more difficult to resolve, but at the same time it is unsatisfying as our modeling of ambiguity reaches its limit here.

The major reason for the GWR performing that well is probably that it achieved to learn a comprehensive model of the distribution of deictic gestures in the scenario. We therefore assume that our labeling function, including the adjustment of the labels during the training, performs as desired. Nevertheless, equipping the GWR's nodes with labels should be treated with caution, since the GWR quickly acquires the character of a look-up table when generating an unreasonable amount of nodes. Determining the point where no additional nodes should be added, however, remains fuzzy as it is strongly associated with the network's performance. For finding this optimal point, one would have to put the f_1-*scores* of different networks in relation to their generated nodes. Although a strong dissonance between performance and number of nodes can certainly be recognized, with finer differences the choice of the best network is based on one's own heuristics.

An essential advantage of recognizing deictic gestures with the GWR is its ability to detect and filter misidentified pointing intentions that result from ambiguous hand postures. However, the need to filter these is only that significant, since we rely exclusively on the static properties of the segmented hand shape to identify the pointing intention. Since the geometric properties of the hand shape are sometimes ambiguous, this leads to misidentified pointing intentions. One way to address this problem would be to consider the temporal structure of a deictic gesture and only identify a pointing intention when the gesture's *peak* is detected. However,

as described in section 2.2.4.3, we found the method considered at the time to be incompatible with our approach. Nonetheless, Park [30] and Stiefelhagen [26] model this temporal structure by training a Hidden Markov Model for each temporal stage (*preperation*, *peak*, *retraction*) and estimate the pointing direction when the gesture reaches its *peak*.

Another substantial benefit of the prediction with the GWR is that, due to the estimated pointing positions, we have considerably more control over the further behavior of the prediction process than with a pointing array. This was already evident in the ambiguity detection, where the determined area A is used to decide whether the current object position is ambiguous. However, what we did not yet cover is the interface's reaction to a deictic gesture that the GWR regards as a *miss*. In this thesis, we state that there are no deicitc gestures without a goal, only that our model has not succeeded in identifying it. Especially in the case where an ambiguity is detected and the regular pointing position is used for predicting the target, a *miss* is more likely to occur. The absence of a strategy to deal with *misses*, especially in the higher ambiguity classes, is one of the more serious drawbacks of this approach so far.

4.1 Future Work

Regarding future work, we propose to provide the deictic gesture interface with additional tools to address the case in which the GWR produces an *miss*. Since we argue that there are no deictic gestures without a target, we assume that the experimenter targets an object when a *miss* is identified. Consequently, it is reasonable to enable NICO to react to this *miss* by giving the experimenter a certain feedback. In fact, it is a natural human behavior to give feedback to another person in an interaction when something is unclear. We consider the use of NICO's speech capabilities to be most practical to provide that feedback. A potential reaction procedure would be to first consult the area A or an enlarged version of it and detect all objects in this area. Then the objects closest to the pointing position are determined and NICO asks the experimenter for the targeted object ("*Did you mean the red or yellow object?*"). Ideally, the experimenter can then also react verbally to this question. After determining the target object the respective observation could be incorporated into the GWR in an online manner.

With this in mind, it would further be useful to employ a more sophisticated object recognition, such as the *RetinaNet* [19] instead of the currently implemented detection approach. This would initially contribute to a more accurate and robust estimation of the object position and thus improve the performance of the GWR. Moreover, since a object recognition is capable of predicting the actual type of the object, it would further enable the realization of more complex interaction scenarios in which deictic gestures are combined with other tasks such as object

grasping.

4.2 Conclusion

This thesis contributes a computer vision based deictic gesture interface with the NICO robot. We design a scenario with adjustable object positions to model different levels of ambiguities. Based on this we build up a large set of recordings with an experimenter performing deictic gestures towards referenceable objects in various constellations. We use these recordings to create an extensive data set of deictic gestures in our scenario. Based on this data, we implement two different approaches for recognizing deictic gestures. Particularity, the approach using a Growing When Required network achieves noticeable results in recognizing deictic gestures, even in decidedly ambiguous object constellations. Furthermore, we achieve our overarching goal of addressing the ambiguities of deictic gestures, by realizing an ambiguity detection that significantly improves the prediction performance in lower levels of ambiguities. Finally, we conclude that we achieve to fulfill all of our goals outlined in section 1.3.

Appendix A

Nomenclature

Appendix B

Bibliography

[1] T. Behne, U. Liszkowski, M. Carpenter, and M. Tomasello. Twelve-month-olds' comprehension and production of pointing. *British Journal of Developmental Psychology*, 30(3):359–375, 2012.

[2] J.-Y. Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker. *Intel Corporation, Microprocessor Research Labs*, 2000.

[3] G. Bradski and A. Kaehler. OpenCV. *Dr. Dobbs journal of software tools*, 2000.

[4] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* O'Reilly Media, Inc., 2008.

[5] G. Butterworth. Pointing is the royal road to language for babies. In *Pointing: Where Language, Culture, and Cognition Meet*, pages 9–33. 2003.

[6] G. Canal, S. Escalera, and C. Angulo. A real-time Human-Robot Interaction system based on gestures for assistive scenarios. *Computer Vision and Image Understanding*, 2016.

[7] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* John Wiley & Sons, 2012.

[8] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[9] F. Franco and G. Butterworth. Pointing and social awareness: declaring and requesting in the second year. *Journal of child language*, 23(2):307–336, 1996.

[10] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[11] E. Jones, T. Oliphant, and P. Peterson. SciPy: Open source scientific tools for Python, 2014.

[12] M. J. Jones and J. M. Rehg. Statistical color models with application to skin detection. *International Journal of Computer Vision*, 46(1):81–96, 2002.

[13] M. Kaâniche. *Gesture recognition from video sequences*. Doctoral dissertation, Université Nice Sophia Antipolis, 2009.

[14] P. Kakumanu, S. Makrogiannis, and N. Bourbakis. A survey of skin-color modeling and detection methods. *Pattern Recognition*, 40(3):1106–1122, 2007.

[15] M. Kerzel, E. Strahl, S. Magg, N. Navarro-Guerrero, S. Heinrich, and S. Wermter. NICO Neuro-Inspired COmpanion: A Developmental Humanoid Robot Platform for Multimodal Interaction. In *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 113–120, 2017.

[16] S. Kita. *Pointing: Where Language, Culture, and Cognition Meet*. Psychology Press, 2003.

[17] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982.

[18] W. J. M. Levelt, G. Richardson, and W. La Heij. Pointing and voicing in deictic expressions. *Journal of Memory and Language*, 24(2):133–164, 1985.

[19] T.-Y. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[20] U. Liszkowski. Infant pointing at twelve months: Communicative goals, motives, and social-cognitive abilities. In *Roots of human sociality : culture, cognition and interaction*, pages 153–178. 2006.

[21] M. M. Louwerse and A. Bangerter. Focusing attention with deictic gestures and linguistic expressions. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, pages 1331–1336, 2005.

[22] S. Marsland, J. Shapiro, and U. Nehmzow. A self-organising network that grows when required. *Neural networks*, 15:1041–58, 2002.

[23] T. Martinetz. Competitive hebbian learning rule forms perfectly topology preserving maps. In *ICANN'93*, pages 427–434. Springer London, 1993.

[24] David McNeill. *Hand and mind: What gestures reveal about thought*. University of Chicago Press, 1992.

[25] M. M. McQueen and G. T. Toussaint. On the ultimate convex hull algorithm in practice. *Pattern Recognition Letters*, 3(1):29–34, 1985.

[26] K. Nickel and R. Stiefelhagen. Visual recognition of pointing gestures for human-robot interaction. *Image and Vision Computing*, 25(12):1875–1884, 2007.

[27] T. E. Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[28] H. Ottenheimer. *The Anthropology of Language: An Introduction to Linguistic Anthropology*. Thomson, Wadsworth, 2006.

[29] G. I. Parisi, C. Weber, and S. Wermter. Self-organizing neural integration of pose-motion features for human action recognition. *Frontiers in Neurorobotics*, 9, 2015.

[30] C. B. Park and S. W. Lee. Real-time 3D pointing gesture recognition for mobile robots with cascade HMM and particle filter. *Image and Vision Computing*, 29(1):51–63, 2011.

[31] V. I. Pavlovic, R. Sharma, and T. S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):677–695, 1997.

[32] S. L. Phung, A. Bouzerdoum, and D. Chai. Skin segmentation using color pixel classification: analysis and comparison. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(1):148–154, 2005.

[33] F. H. Rauscher, R. M. Krauss, and Y. Chen. Gesture, speech, and lexical access: The role of lexical movements in speech production. *Psychological Science*, 7(4):226–231, 1996.

[34] S. S. Rautaray and A. Agrawal. Vision based hand gesture recognition for human computer interaction: a survey. *Artificial Intelligence Review*, 43(1):1–54, 2015.

[35] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 779–788, 2016.

[36] J. Richarz, A. Scheidig, C. Martin, S. Müller, and H. M. Gross. A monocular pointing pose estimator for gestural instruction of a mobile robot. *International Journal of Advanced Robotic Systems*, 4(1), 2007.

[37] J. Sklansky. Finding the convex hull of a simple polygon. *Pattern Recognition Letters*, 1(2):79–83, 1982.

[38] M. Tomasello, M. Carpenter, and U. Liszkowski. A new look at infant pointing. *Child Development*, 78(3):705–722, 2007.

[39] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I–I, 2001.

[40] Y. Wen, C. Hu, G. Yu, and C. Wang. A robust method of detecting hand gestures using depth sensors. In *2012 IEEE International Workshop on Haptic Audio Visual Environments and Games (HAVE)*, pages 72–77, 2012.

[41] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

[42] J. Yang, W. Lu, and A. Waibel. Skin-color modeling and adaptation. In *Asian Conference on Computer Vision*, pages 687–694, 1998.

[43] H.-S. Yeo, B.-G. Lee, and H. Lim. Hand Tracking and Gesture Recognition System for Human-computer Interaction Using Low-cost Hardware. *Multimedia Tools and Applications*, 74(8):2687–2715, 2015.

[44] X. Zabulis, H. Baltzakis, and A. Argyros. Vision-based hand gesture recognition for human-computer interaction. *The Universal Access Handbook*, 34, 2009.

Erklärung der Urheberschaft

Hiermit versichere ich an Eides statt, dass ich die vorliegende Bachelorthesis im Studiengang Human-Computer-Interaction selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum Unterschrift

Erklärung zur Veröffentlichung

Ich stimme der Einstellung der Bachelorthesis in die Bibliothek des Fachbereichs Informatik zu.

Ort, Datum Unterschrift