Universität Stuttgart

Institut für Visualisierung und Interaktive Systeme (VIS)
Abteilung Computational Cognitive Science

Universitätsstraße 32
70569 Stuttgart

Bachelorarbeit

Augmented Reality to Improve Electroencephalography (EEG) Cap Preparation

Yannik Bocksch

Studiengang:	B.Sc. Softwaretechnik
1. Prüfer:	Jun. Prof. Benedikt Ehinger
2. Prüfer:	
Betreuer:	M.Sc. Vladimir Mikheev
begonnen am:	09.02.2024
beendet am:	09.08.2024

Augmented Reality to Improve Electroencephalography (EEG) Cap Preparation

Yannik Bocksch

Abstract

EEG (Electroencephalography) cap preparation requires a significant amount of time before actual measurements can be performed. It takes even longer for EEG caps that need some form of conductive gel or saltwater to decrease contact impedance between the electrodes and the scalp. In this thesis, several methods for detecting electrodes in a camera image and identifying detected electrodes are developed and evaluated for performance and reliability. Additionally, an application is developed that leverages augmented reality to visualize important values, like the impedance of electrodes, directly in the view of the operator. The proposed application runs on the Microsoft Hololens 2 that is used by the operator to overlay a visualization of impedance values of the electrodes over the electrodes themselves, which lets the operator see those values immediately. This reduces the time required to look at a separate device that shows impedance values and locate electrodes on the cap that need additional work. However, the identification of electrodes from the camera image does not work reliably enough with the proposed methods, and several ArUco markers, placed in predefined locations on the EEG cap, are used for aligning the virtual representation of the electrodes with the real electrodes.

- *Y. Bocksch, B.Sc. Software Engineering*
  E-Mail: st161614@stud.uni-stuttgart.de
  Student Number: 3400571

CONTENTS

1 Introduction

EEG (Electroencephalography) [6] is a method of recording brain activity by measuring small amounts of voltages on the scalp of the head. These voltages can be measured by electrodes that either directly attach to the scalp or form a contact by using some conductive interface material like conductive gel. In some cases, the electrodes or electrode holders, where electrodes will plug in, are integrated into an EEG cap. The caps can preserve a specific layout of electrodes, like a grid-like pattern, that allows them to be placed on the head all at once. The electrodes record brain activity by measuring voltages emitted from electrochemical processes during brain cell communication.

The layout of electrodes on an EEG cap can differ greatly. There are caps with as few as 24 channels or less, and caps with over 400 channels. But also, the positioning of the electrodes on the cap can differ. On most EEG caps, the electrodes are placed symmetrically on two axes, front to back and left to right. The EEG cap used in this thesis is the waveguard ^{TM} original [1] by ANT Neuro [2] providing 128 channels, which uses the extended 10-20 system as its layout.

Besides different layouts, there are also several types of electrodes. There are electrodes that form contact with the scalp by pressing against it with metal combs or conductive rubber materials. Those electrodes are called dry electrodes and do not depend on any additional conductive material. But there are so-called wet electrodes, which need a conductive material to form the contact between the electrode and scalp. Those could be water-based caps that require saline water, or gel-based caps that use a conductive gel. All types of electrodes have their advantages and disadvantages, but the EEG cap used for the experimental work of this thesis is gel-based and requires a conductive gel between the electrodes and the scalp.

With all types of electrodes, the contact between the electrodes and the scalp has to be good for the voltage measurements to be accurate. A bad contact between electrodes and the scalp can lead to noisy measurements. Some reasons for bad contact can be having not enough gel between an electrode and the scalp or too much hair in between the electrodes and the scalp. The quality of a contact can be measured in impedance. Impedance is similar to resistance but in AC circuits. Like resistance, the impedance has to be low for good contact and good measurements.

Most setups require the recorded data to be sent to different computers over a network, as the person wearing the EEG cap (the patient) sits in a separate room from the examiners. This allows the patient to not be distracted by anything else than the experiment itself and is required for experiments that, for example, examine brain activity depending on different types of stimuli. The distribution of the EEG data to other computers in the same network can be done with the Lab Streaming Layer (LSL) [8]. The Lab Streaming Layer is a system that handles the networking and the time synchronization for streaming measurement

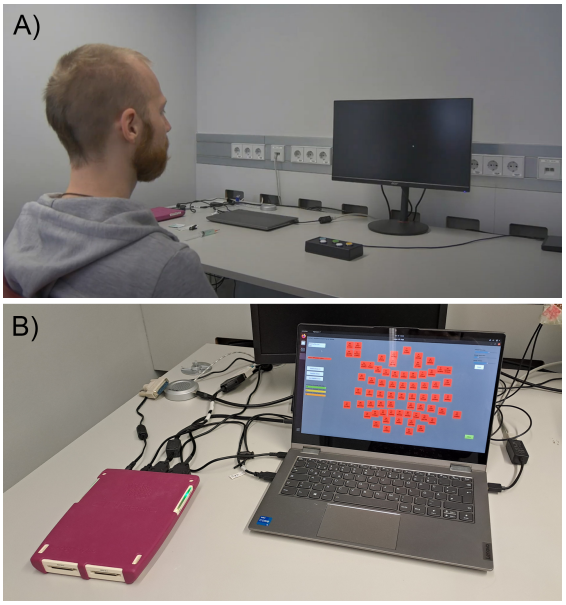

Fig. 1: Images of the current lab setup. A) shows the overall setup with the sitting position of the patient before the EEG cap is put on. B) A more detailed image showing the laptop, connected to the amplifier, displaying the impedance values.
Source: A) Screenshot captured from YouTube Video [17] made by Martin Geiger. B) Image taken by Maanik Marathe

time series over a network and is designed for neural, physiological, and behavioral data.

The topic of this bachelor's thesis is to use augmented reality (AR) to visualize data from the EEG cap in the operator's view. Augmented reality is the concept of displaying computer user interfaces on top of the real world. This is achieved primarily by wearing a headset, such as the Microsoft HoloLens [10], which has two displays built into it, sitting in front of the eyes of the user. With these two displays, applications can display objects or user interfaces in 3D space around the user. In this thesis, augmented reality is used to visualize important data required for the preparation of EEG caps on top of the EEG cap itself. This allows the operator to see the impedance of electrodes directly in front of them without the need to look at another device or monitor, which should result in high-quality EEG measurements and faster preparation times.

2 Motivation

2.1 Problem

The setup of a gel-based EEG cap includes several steps that need to be performed before any measurements can be taken. At first, the cap has to be placed onto the head of the patient while all electrodes have to be at their correct positions on the head. This is achieved by using anatomical landmarks for guidance. Then, the external electrodes, which are attached to the cap by wires, need to be placed at their designated positions. These are typically called EOGs and can include some

Fig. 2: Image showing the author wearing the Microsoft HoloLens 2 and looking at the EEG cap.

around the eyes and at the ear. After everything has been placed at the correct positions, the gel needs to be inserted into the center holes of the electrodes. The conductive gel helps to form a good electrical connection between the scalp and the electrodes.

The quality of the electrical contacts between the electrodes and the scalp can be measured as impedance. The amplifier of the EEG cap can measure the impedance of electrodes. Some amplifiers use a reference electrode for that, which means it is important to prepare the reference electrode first to get its impedance low. After that, the impedance values for all other electrodes can be used to determine which electrodes have a high impedance. The impedance of electrodes can be lowered by pushing the hair under the electrode away, scrubbing off the top layer of skin, or adding more gel if needed. It is possible to add too much gel into electrodes, which can result in electrodes bridging together. The impedance values can be seen on a separate device nearby, for example, a laptop connected to the amplifier of the cap.

This means that the person preparing the cap (the operator) has to look repeatedly at a separate device to see the impedance values, which are required for the preparation. After the gel has been inserted into all electrodes, the impedance values have to be checked for electrodes that still need better contact. This includes looking at the separate device, finding impedance values that are too high, locating the associated electrode on the EEG cap, and improving the contact between the electrode and the scalp. The operator has to go through these steps several times until all electrodes have low enough impedance values. This process is very time-consuming, depending on the number of electrodes on the EEG cap and the skill level of the operator. The time required to prepare an EEG cap has to be taken into account when performing experiments, as it is a substantial amount of the total time for the experiment.

2.2 Key novelty and contributions

Different detection and identification methods described in this thesis allow the detection of electrodes from camera images using the HoloLens' camera and the identification of detected electrodes. The proposed application of this thesis uses this information and

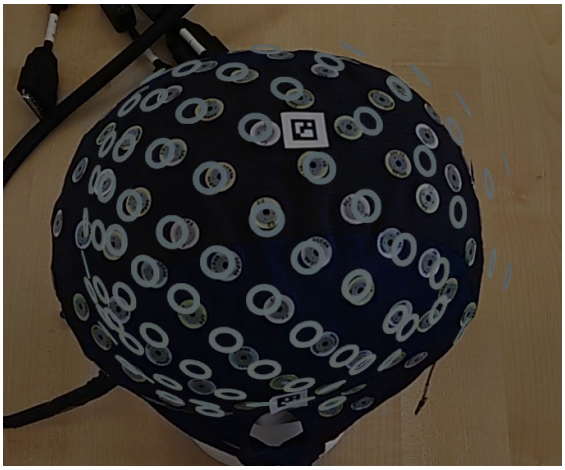

Fig. 3: Image taken with the HoloLens that shows how visualizations are overlaid in augmented reality over real electrodes in the final version of the application. The alignment of the virtual electrodes to the real electrodes is not perfect.

augmented reality (AR) to help the operator in the preparation steps. Augmented reality allows AR headsets to show objects and user interfaces directly in the view of the operator wearing the headset by using two displays in front of the eyes. The two displays allow the headset to show objects that appear to the operator as 3D objects around them. Additionally, AR headsets track themselves and, therefore, know where they are in the environment, which allows the developer to place visualizations in one place and keep them in that position. The AR headset used in this thesis is a Microsoft HoloLens 2 [10], which provides all the necessary hardware and features, like the camera and hand-tracking, for this project.

This application aims to help an operator in the preparation step by showing important indicators, like the impedance values of electrodes, directly in their view. It visualizes the impedance value of each electrode directly on top of the electrode on the EEG cap. Figure 3 shows how the overlay would look for the operator. This means that the operator can see the impedance values while looking at each electrode and does not have to look at a separate device and search for electrodes. The visualizations on the EEG cap also remove the need to locate electrodes with high impedance on the EEG cap, as the values can be directly associated with their respective electrodes. Overall, this application should help decrease the time needed to prepare a gel-based EEG cap, and it can also help in improving the overall quality of the contacts between electrodes and the scalp.

3 Related Work

The company Brain Products [5] provides hardware and software solutions for EEG research. One of their products is the actiCAP [4], a gel-based EEG cap that uses active electrodes. These active actiCAP electrodes

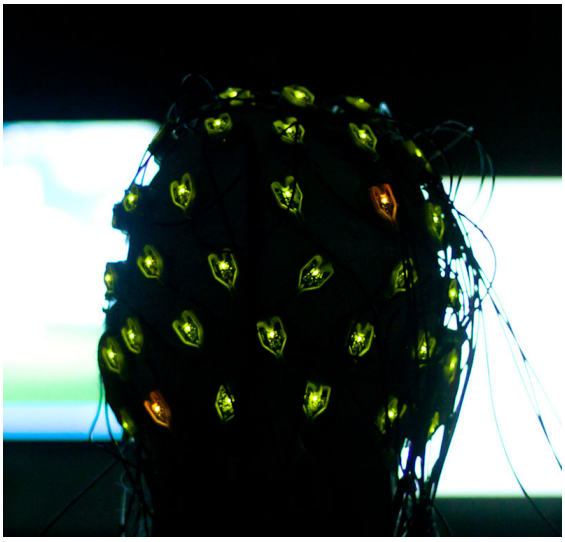

Fig. 4: Image from the active electrodes of the actiCAP EEG cap. All, except for two electrodes, light up green indicating their impedance is low enough. Two electrodes light up orange, indicating their impedance is higher.
Source: Image used with permissions from Benedikt Ehinger

include a multicolor LED (Light Emitting Diode) on each electrode [4], which can be used to visually indicate impedance values by lighting up in three colors (red, yellow, green) [3] while the cap is being prepared. Figure 4 shows the LEDs indicating their impedance value. This allows the operator to see the impedance values directly on the electrode. This approach decreases the preparation time by reducing the number of times the operator has to look at a computer monitor and locate electrodes on the cap.

Malmivuo et al. [22] talk about smart electrodes in their paper about high-resolution EEG recordings. Their smart electrodes also have LEDs built in. They use them to provide feedback on the impedance of the electrode to the operator in a similar way as the actiCAP does. Additionally, they use the LEDs on the cap to scan the positions of the electrodes on the head. They use multiple cameras that can see the electrodes on the cap and by turning on specific LEDs and capturing the electrode with the LED from multiple cameras, they can triangulate the position of each electrode on the head. The positions can be used as additional spatial information for analyzing EEG recordings.

Both of the approaches help to improve the preparation time in a similar way to the proposed application of this thesis. They show impedance values directly on the cap, but instead of using augmented reality (AR) with an AR headset, they use integrated LEDs to show the values in the form of different colors. An advantage of electrodes that have LEDs built into them is that they do not need additional hardware and software, except for the electrodes themselves, to show impedance values directly on the patient's head. However, LEDs only work with specific caps that have the necessary hardware built into them. The proposed application in this thesis can be modified to work with other caps and could be used with all kinds of EEG caps, which cannot show impedance values themselves. Additionally, it could still help in the preparation of EEG caps with built-in LEDs by showing additional information like absolute values, which are still necessary despite LEDs. The application can also show other values than impedance values, which the EEG caps with LEDs cannot. On the other hand, the proposed application in this thesis requires a HoloLens, or similar AR headset, and additional software to work properly, which adds additional setup time and possibly cost.

Fiedler et al. [18] proposed an EEG cap consisting of 256 dry electrodes. Dry electrodes do not need any additional material to interface with the scalp and instead use some sort of pins that touch the scalp directly to be able to measure voltages. Fiedler et al. [18] have found that in comparison to gel-based EEG caps, the dry electrode cap is much faster in terms of preparation time as there is no need to insert a conductive gel into each electrode. It allows for "rapid applications by not medically trained persons" [18]. They found that such dry electrode EEG caps have the advantage that there is no gel residue left in the patient's hair afterward that needs to be washed out. In addition, there is no need to wash the gel residue off the electrodes. On the other hand, the reported comfort of wearing the dry electrode EEG cap is lower than with gel-based EEG caps [18] due to the electrodes having metal combs that have to press against the skin of the head. However, the dry electrode EEG cap was still preferred over gel-based EEG caps because of the advantages it provides.

Also, Lee et al. [19] have proposed dry electrodes for use in EEG. Their electrodes consist of multiple spring-loaded metal pins, which press against the skin to improve contact with it. Those metal pins have a small finger-like structure at the tip, making it easier for the electrodes to touch the skin under the hair. They compared their dry electrodes against another type of dry electrodes by a manufacturer called G.tec and conventional gel-based electrodes. Lee et al. [19] have found in their testing, that the impedance values of their proposed dry electrodes are only slightly higher than the gel-based electrodes. They also mention advantages like "convenient installation" [19] and "feasibility of long-term monitoring" [19].

As Fiedler et al. [18] and Lee et al. [19] have shown, there are alternatives to gel-based electrodes that do not require the application of conductive gel, which results in faster preparation times and less complex preparation procedures. However, the quality of measurements recorded with dry electrodes is heavily dependent on the application and the type of the electrodes.

4 Approach

4.1 Architecture

The proposed application requires several components to work together. The application itself needs to work with Augmented Reality (AR) headsets to display 3D

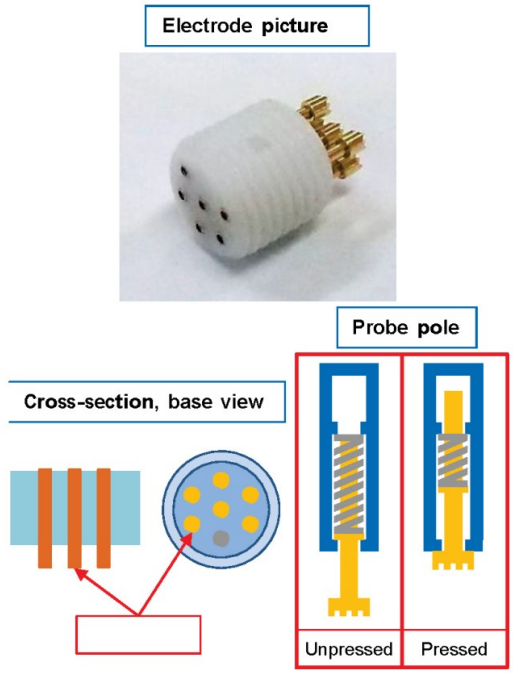

Fig. 5: Image showing the structure of an electrode using spring-loaded fingers.
Source: Lee et al. [19]

objects in front of the operator. It also needs to detect electrodes from camera images reliably and identify detected electrodes by assigning them their correct name. For the application to show impedance values over the electrodes of the EEG cap, it also needs to receive the EEG LSL stream.

The use of a 3D game engine for this kind of AR application has several advantages. Game engines already include the necessary rendering pipelines that are needed to render 3D objects. They also include various utilities and tools to work with 3D objects and scenes. Game engines usually include utilities for coordinate calculations with 3D coordinate systems and 2D coordinate systems and handling calculations with camera parameters. Additionally, game engines provide user interfaces that allow the user to edit 3D objects and scenes using the game engine's UI, which helps to picture the content of a scene and how it will look. Furthermore, they also provide numerous tools to debug and analyze the behavior of code in the application. The use of game engines for AR applications decreases the necessary work that would have been needed to develop these tools and utilities from scratch. There are many game engines available. The biggest of them are Unreal Engine [16], Unity [15], and Godot [7].

Microsoft provides a Mixed Reality Toolkit (MRTK) [12] that helps applications interface with augmented reality headsets. It provides several features from AR headsets, like the HoloLens 2, to Unity and the developer. One feature, that is necessary for this application, is capturing images and videos from the camera on the HoloLens, since it depends on computer vision tasks. MRTK also provides some utilities in the engine, like pre-made components, which can be placed in a Unity scene and handle the virtual camera that is used to render the output for both eyes. Those utilities also handle hand tracking and, if needed, eye tracking. Additionally, MRTK provides various pre-made UI components that can be used to create user interfaces like windows or menus. The interaction between the hands or the eyes and the UI is already handled by MRTK. It is very simple to assemble components into a window that can be used to allow the user to interact with the application by visualizing information and providing actions. MRTK helps developers build AR applications by providing several features and ready-to-use components, but it also allows developers to build custom components that use features from AR headsets. The Microsoft HoloLens 2 is not the only supported device by the MRTK. It also supports other headsets, which use the OpenXR specifications, like the Meta Quest. The MRTK is available for Unreal Engine, Unity, JavaScript for WebXR applications, and native applications, which do not use a framework or game engine.

MRTK is compatible with the game engines Unity and Unreal Engine. Unity was selected for this application mainly because Unity uses C# as the programming language for scripts, and prior experience in Unity and C# exists. It integrates nicely with MRTK and in combination, these are a good base for the application.

The application developed in this thesis is supposed to visualize impedance values from the EEG cap in a way that allows the operator wearing the AR headset to see them in their view and understand the values. It should help the user prepare an EEG cap by providing important metrics directly in the view of the person. In this case, the impedance values for each electrode are displayed on top of the electrode in the real world by overlaying a visualization over it. Therefore, the application needs to know where the electrodes are relative to the location of the headset. To obtain the positions of the electrodes, the application uses computer vision to detect the electrodes on the cap and calculate their positions in 3D space. Unity and the MRTK are used to capture camera frames of the HoloLens camera, which are passed to OpenCV [14][13], an open-source utility, that has everything built in that can be used for computer vision tasks. OpenCV is then used to process the image and detect electrodes in the camera frame. The detected positions of electrodes can be passed back to Unity, which can use them to overlay visualizations over the electrodes in AR.

However, in addition to the positions of the detected electrodes, the application needs to know which detected electrodes correspond to which real electrodes. It needs to identify detected electrodes and assign them their correct name, which is required to calculate their 3D positions on the EEG cap relative to the camera. The initial goal was to detect and identify them without any additional markers, like QR codes or ArUco markers, that had to be placed on the EEG cap and would have been used to help in this process. The application could use the positions of detected electrodes and their names

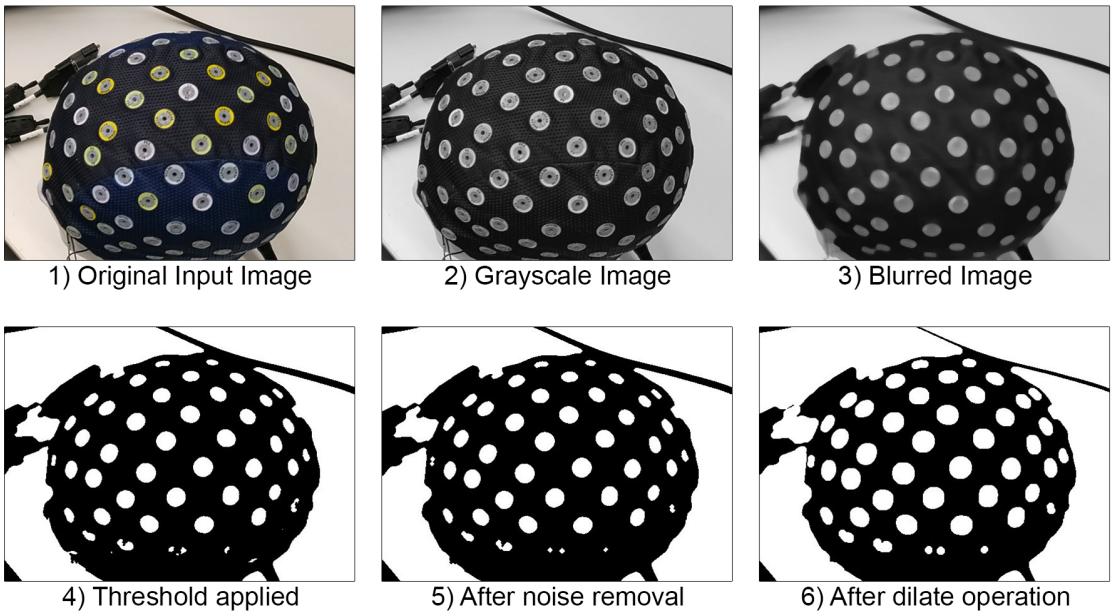

Fig. 6: Results of the 6 steps of the image preprocessing. 1) Original image taken by the camera. 2) Image converted into a grayscale image. 3) Grayscale image blurred. 4) Result of applying a threshold to the blurred image. 5) Result of removing smaller noise by slightly reducing white areas. 6) Result of increasing white areas by a small amount.

to extrapolate the 3D position of every other electrode that isn't detected on its own by using the positions of the detected electrodes. The positions could then be used to overlay visualizations over the real electrodes in the view of the operator wearing the headset, as shown in Figure 3.

4.2 Detection of Electrodes

For the application, developed in this thesis, to be able to visualize data on top of the electrodes, it needs to know their location. Specifically, it needs to know where the real electrodes are located relative to the camera of the HoloLens. It can use this information to calculate where, on the screens, it has to show the visualization so that they overlay over the real electrodes from the operator's perspective.

To get the positions of the electrodes on the EEG cap, the application has to find them in the camera image. The Mixed Reality Toolkit (MRTK) provides the necessary APIs for the application to use the camera and take images or videos with it. In this case, it needs to take images every set amount of time and process those images to detect the electrodes and their positions of the EEG cap in front of the user. The camera image is passed to OpenCV [13] for this task, which has all the necessary utilities and functions to process images and detect features in them. After the electrodes have been detected, the application can use those positions and transform them to include the position of the camera on the HoloLens, which is offset from the eyes and the displays. It has to do this reliably over several consecutive frames.

4.2.1 Image Preprocessing

The camera of the HoloLens provides RGB images that consist of a channel for each of the three colors (red, green, and blue). However, the detection process doesn't benefit from multiple color channels. A grayscale image with only one channel is enough for the detection methods to detect electrodes. Converting the image into a grayscale image improves processing power requirements and performance. To remove some noise from the image, the grayscale image gets blurred by a small amount. The image is then thresholded with a threshold of 100 on the scale between 0 and 255 to highlight brighter parts of the image, like the electrodes on the cap. Some smaller noise in the black and white image, that remains, will be removed with an erode operation that decreases white areas slightly. This smaller noise is a result of the fabric of the EEG cap used in this thesis, which can appear brighter in some small spots due to it reflecting light. The last step of the image processing is a dilate operation, which increases the highlighted areas by a small amount after they have been decreased by the preceding erode operation. Additionally, this operation helps to close some smaller holes in the highlighted areas. Figure 6 shows the result of the different preprocessing steps. The resulting image, after all of these operations, is better suited for detection methods than the original version as it highlights the electrodes, which are brighter than the rest of the cap, as white dots on a black background.

4.2.2 Contour Shape Detection

The first failed attempt at detecting electrodes in the preprocessed camera image uses contours. It uses OpenCV [13] and its `findContours(...)` method.

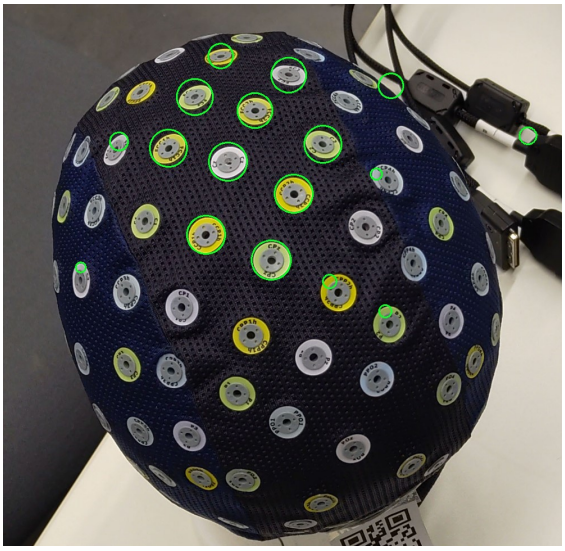

Fig. 7: Detection of electrodes by using contour detection and minimum enclosing circles around contours to detect circles in the preprocessed image. Detected circles are marked green.

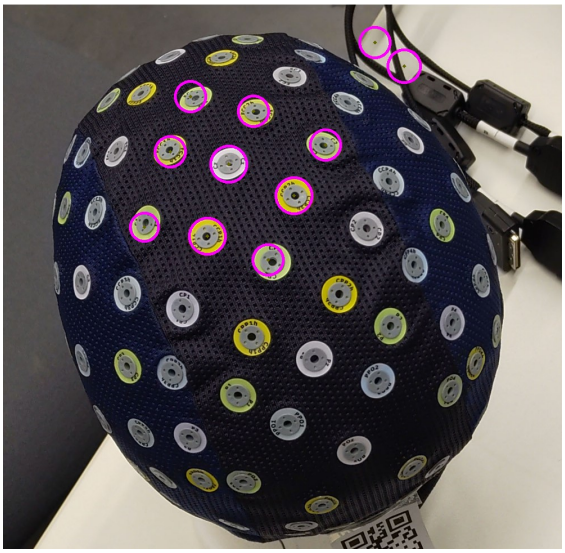

Fig. 8: Detection of electrodes by using HoughCircles method of OpenCV to detect circles in the preprocessed image. Detected circles are marked pink.

A contour is a set of points that define a shape, like the key points of a polygon. The `findContours(...)` method of OpenCV tries to find shapes that set themselves apart from the rest of the image. In this case, the input is a black and white image, which makes it easy for OpenCV to find shapes in the image. Some of the contours that OpenCV returns are the white circles that represent electrodes on the cap and are highlighted in the preprocessing of the camera image. The extracted contours are then simplified by approximating the shape of the contour with fewer points. The approximated shapes can then be used to filter out shapes, that are not circles, by checking the number of points for each contour. If a contour, for example, has three points, it resembles a triangle and not a circle. A contour is considered a circle if it consists of more than five points and the area of the contour is in between two specified limits. After all contours have been filtered and only circles remain, these contours are passed to OpenCV again, to calculate the minimum enclosing circle for each contour. The result is a list of circles with position and radius attributes. Figure 7 shows an example image with the result of this method. The detected circles and therefore detected electrodes can then be used in the identification of electrodes later.

Although this method detects circles well enough, it does not work well enough in this specific use case. The electrodes on the EEG cap appear as circles in the middle of the view, but they appear as ellipses the more they are toward the edges of the EEG cap due to the perspective warp and the round shape of the head. The electrodes that appear as ellipses are still detected as circles, which means that the sizes, areas, and shapes are not always accurate enough. It is also not possible to filter shapes based on the moment of inertia of the ellipses, which would be helpful to filter out false detections. The moment of inertia of an ellipse describes how close the shape of an ellipse is to a perfect circle, and is calculated by dividing the smaller length of the bounding box of the ellipse by the larger length. Figure 13 shows an example of two ellipses and their moment of inertia.

4.2.3 *Hough Circles Detection*

Another attempt at solving the electrode detection problem is to use the `HoughCircles(...)` method of OpenCV [13]. The `HoughCircles(...)` method from OpenCV uses the Hough Gradient Method to detect circles. It also gets the preprocessed black and white image as its input and detects shapes resembling circles. It can detect electrodes on the EEG cap and also manages to detect some of the electrodes that are more toward the edge of the cap and therefore more squished due to the perspective warp. The result of the `HoughCircles(...)` method is a list of circles with their position and radius attributes. Figure 8 shows an example of the result of this method. Like the previous method, it does not contain any additional information about the shape, since it is designed to detect pure circles.

The problem with this method is that it has several parameters, which are not documented very well. It takes several attempts to fine-tune the parameters so that the electrodes get detected correctly and stable enough. The `HoughCircles(...)` method has two parameters for the minimum radius and the maximum radius of circles it should detect. These parameters have to be set to a quite small range for the method to only detect the correct electrodes and have the least amount of false detections. However, this means that the slightest difference in the size of the electrodes can lead to them not being detected. If the camera moves further away or towards the EEG cap and the size of

the electrodes changes in the input image, the detection of the circles won't work reliably. This method is good for a camera with a static position and a scenario that requires the detection of flat circles.

4.2.4 Template Matching

The third attempt at detecting electrodes uses template-matching methods. Template matching works by providing a template image, which is then searched in the input image. There are multiple ways to achieve this method. One way is to use feature-matching algorithms. They use features from the template image and another image and try to find as many matches between features from both images. Features are points in an image with some additional context information and are created by finding key points in an image and taking the position of those pixels and information about the surrounding pixels. Key points can be parts of an image that stand out in color or brightness, like edges of objects or smaller dots or specs of dust. If it finds enough matches between the feature points of both the template and the search image, it can calculate where the template image is inside the search image. There are several ways to extract features from an image, but in this case, ORB (Oriented FAST and Rotated BRIEF) [23] features were used. OpenCV [13] provides methods for extracting ORB features from images out of the box.

In this case, the template image was created by isolating an electrode from an image of the cap, so that the template image only contained the electrode itself and nothing around it. OpenCV is then used to extract features from the template image and the camera image from the EEG cap. A simple `BFMatcher` is used to match features from both sets. The `BFMatcher` algorithm got its name from brute force, which means it tries different matches and if it finds one that is good enough, it returns it. Another attempt at this uses SIFT (Scale Invariant Feature Transform) [21] features instead of ORB features. The new features are then used with a `FLANNBasedMatcher`, another feature-matching algorithm that uses a k-nearest neighbor search.

But in this case, both algorithms did not detect any electrode in the camera image. The reason for this is probably that the electrodes in the camera image are much smaller and shaped differently. The electrodes in the center of the camera view represent the template image the most. However, further toward the edge of the cap, the electrodes are more shaped like ellipses due to the perspective warp. Also, the labels of the electrodes have different colors, which also adds up to the difference. This means that there are not enough features that match up in both images, and the algorithms can not detect a good match between both feature point sets. The only time in which the feature matching algorithms were able to detect an electrode was, as shown in Figure 9 when using the image of a single electrode from the same image as it was searched in. In this case, it was able to detect that single electrode since the template was an exact part of the image.

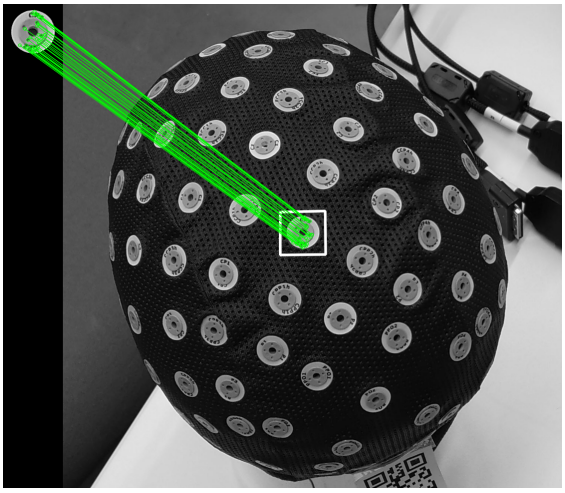

Fig. 9: Detection of an electrode using the OpenCV `FlannBasedMatcher` method. The image shows features that were matched successfully between the template and the search image.

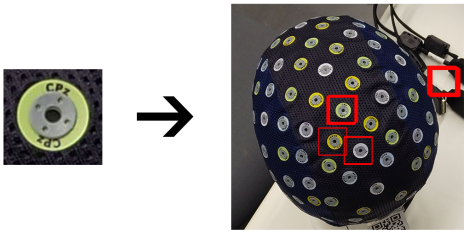

Fig. 10: Detection of electrodes using OpenCV's `matchTemplate(...)` method. A) shows the reference image of a single electrode used. B) shows the result of the `matchTemplate(...)` method with a low threshold.

OpenCV also provides a function called `matchTemplate(...)`, which works by sliding the template over the search image and calculating the pixel difference of the template to this part of the search image for each sliding position. It then can take the position with the least difference as the position where it found the template in the search image. However, for this algorithm to work, the template image has to be very similar to the part in the search image that the algorithm is searching for. A perfect use case would be to find a 2D template image in another 2D image, and not to find a 2D image of an electrode in an image with a 3D EEG cap. If there are too many differences due to perspective warp, different colors, or different lighting, like in this case, this algorithm won't work. Figure 10 shows that it detects a few electrodes correctly and some false positives if the required threshold is set low enough.

4.2.5 Ellipsis Detection

The last attempt at detecting electrodes from the camera image was detecting ellipses directly instead of circles.

Fig. 11: Detection of electrodes with contour detection and `fitEllipse(...)` method of OpenCV. Detected electrodes are marked with their detected color. Letters near each electrode indicate classified color.

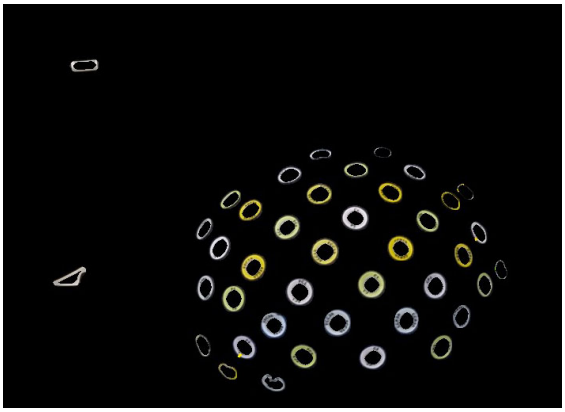

Fig. 12: All masks combined that were used to detect the color of each electrode. Used for visualization only, the color is detected with a single electrode mask each.

This method also uses contours like in an earlier attempt. It uses the Canny Edge Detection feature from OpenCV [13] to highlight edges in the black and white image of the preprocessing step. This algorithm is quite good at detecting edges and in this case, with a black and white image, it just outlines the black and white blobs in the image. However, this already helps the `findContours(...)` function of OpenCV to calculate the contours for each of the blobs. The resulting contours are being filtered by their shape, like in an earlier attempt, by filtering out contours that contain less than six points. This way, it only considers shapes that are more circle-like and not shapes that have fewer edges, like triangles, for example. After having a list of contours that represent circle-like shapes, OpenCV can be used to approximate the ellipse attributes for each shape. OpenCV provides a function called `fitEllipse(...)`, which takes a contour as input and tries to construct a minimal ellipse that represents the detected shape as closely as possible.

The resulting list of ellipses still contains false positives that have to be filtered out. One criterion is the minimum and maximum area. The area of the ellipse

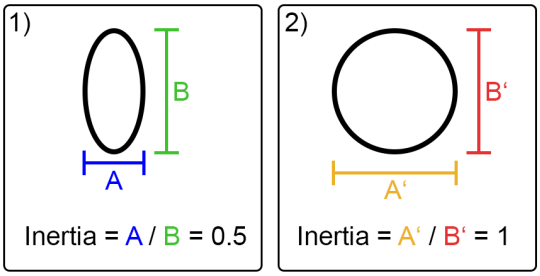

Fig. 13: Graphic explaining what the moment of inertia of an ellipse is. 1) shows a squished ellipse, while 2) shows a circle.

is calculated, and the ellipse is only considered going forward if its area value is in between the minimum and maximum limits. Additionally, the shape of the ellipse itself is checked by using the moment of inertia. Figure 13 shows examples of ellipses and their moment of inertia. It is calculated for each ellipse, and ellipses that have a moment of inertia outside a specified range are ignored. This filters out false positives, which are too wide or flat, like some electrodes that are way too far toward the edge of the EEG cap and therefore not usable in the following detection. The remaining ellipses are a good selection of electrodes that can be reliably detected and used in later steps. Figure 11 shows an example result using this method.

The next step is to determine the color of the label of the electrode. Each electrode on the cap has a plastic ring around it, which has one of four colors (green, yellow, blue, and gray) and has the label of the electrode on it. The text on the label is too small and sometimes obstructed by the electrode itself, and therefore it isn't possible to be used in the computer vision part to identify electrodes. To detect the color of an electrode, a mask is created, which just contains a single electrode including its label. The core of the electrode, which is the electrode itself, and is always gray, is also cut out so that the only remaining part in the mask is the label of the electrode with the color. Figure 12 shows what the masks include. This mask can then be used to mask out the electrode from the HSV (Hue, Saturation, Value) version of the original camera image. OpenCV provides a `mean` function, which can be used to calculate the mean color in HSV space of an image, and it allows for a mask input. The resulting hue value from the mean HSV color can be used in conjunction with several thresholds to classify the color into one of the four possible colors. Each color has a valid range of hue values, which is used to classify them. These threshold values were tuned to provide the most accurate color classification in this specific environment. The list of detected electrodes and their colors can be used in different approaches to identifying electrodes later.

This approach detects electrodes that are more in the center of the camera image reliably enough to be used in the application. Only some electrodes far toward the edge of the cap are not detected reliably over multiple consecutive frames. Additionally, the detection of color

works quite well with the camera of the HoloLens and the lights in our environment. The combination of the lights and the camera lets the colors of the labels appear vibrant, which makes it easy to detect them. However, it is not perfect as well. In some cases, it misidentifies green and yellow, depending on the perspective and shadows on the EEG cap. This may be fixed by fine-tuning the color thresholds used to classify a hue value into the four colors and the sizes of the mask used to calculate the mean color of the label.

4.3 Identifying Electrodes

The second part of getting the positions of the electrodes on the EEG cap, relative to the camera of the HoloLens, is to identify which detected electrode corresponds with which real electrode. The application requires, in addition to the positions of electrodes, the names of electrodes to calculate their 3D positions relative to the camera of the HoloLens. Several attributes of the detected electrodes can help in the identification step. Some of the following methods use the color of the label around each electrode in the identification process. Additionally, the neighbors of each electrode and their positions and colors are also used in some methods. In general, the identification methods should provide a reliable result over many consecutive frames and should not identify electrodes incorrectly, as this could end up in a misaligned visualization that may show wrong values for the electrodes.

4.3.1 Feature Matching

The first attempt at identifying electrodes used feature-matching algorithms. Instead of using them to find a template image in a search image, like in section 4.2.4, they are used to match two sets of arbitrary points. If there are enough corresponding points from both sets, the position, scale, and rotation of the object in the image can be determined.

This attempt to identify electrodes used the positions of electrodes that were detected in the camera frame as one set of features. The other set of features represents the positions of electrodes as shown in AR. The manufacturer of the EEG cap provides a file containing the 3D positions of the electrodes. Those were imported into Unity and virtual electrodes were placed at the 3D positions of the electrodes in the file. The application could then project the positions of the virtual electrodes onto the camera view and calculate the 2D positions of where the virtual electrodes are on the virtual camera frame of the HoloLens, which is displayed to the operator. A feature-matching algorithm could then use both sets of features and return a match between the features of both sets.

One of the feature-matching algorithms used was the `BFMatcher` of OpenCV. It works by calculating the distance between features of both sets and trying to use the closest one for each feature. The problem with this feature-matcher is that perspective warping of the features leads to the feature-matcher not being able to find matches. And because the positions of electrodes on the real EEG cap are slightly different from the positions of electrodes in the 3D positions file does not help. Another attempt was to use the `FLANNBasedMatcher`. Both matching algorithms did not work properly in this case. One reason for that might be that the features used in this case only consist of the positions and don't contain any additional information about neighboring pixels, which might not be enough for the matching algorithms.

4.3.2 Iterative Closest Point

The iterative closest point (ICP) algorithm is designed to align two point clouds with one another. Point clouds are sets of points, normally with 3 dimensions. They are very common in 3D scans and are the result of a 3D scanner scanning an object or area. The ICP algorithm can be used to stitch together multiple point clouds by calculating the transformation and rotation needed to align two point clouds together. The two point clouds have to be roughly aligned beforehand, and it does not work if the difference between the two point clouds is too big. It works by estimating the transformation parameters roughly and then iteratively selects for each point of set A the nearest point of set B. In each iteration, it calculates new transformation parameters using the point matches between set A and set B by calculating the difference of all points. After having calculated new transformation parameters, the algorithm starts over until a set amount of iterations has been reached and the algorithm stops. The result of the algorithm is the last calculated transformation parameters, which can be used to align the two point clouds, and possibly an error value.

In this application, the ICP algorithm was used in an attempt to identify detected electrodes by trying to align them with the reference positions. The same reference positions of the virtual electrodes were used as in the previous attempt with feature-matchers. The ICP algorithm only requires positions as input and does not need any additional information. Additionally, it would be possible to roughly align the virtual electrodes in the AR application with the real electrodes beforehand, which is required for the ICP algorithm. The problem with this algorithm is that it often returns the wrong positions of electrodes. The resulting transformations don't align the two point sets correctly and only apply a small amount of position difference. This might be because the point clouds differ too much. In the typical use-case of the ICP algorithm, the point clouds line up pretty well and all points differ just in their position and rotation and the difference is the same for all points. But in this case, the electrodes are transformed differently for each point. Each point from set A needs a separate transformation to match up with the correct point from set B. The reference 3D positions for the electrodes from the 3D positions file are not shaped like a real head and are roughly placed on a sphere. This means that the detected electrodes can not be aligned to the reference positions by a single position and rotation transformation.

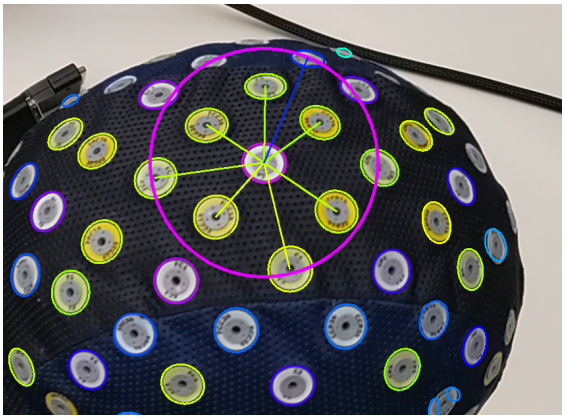

Fig. 14: Visualization of determined neighbors for a single detected electrode by using the distances in the image. The pink circle visualizes the radius in which electrodes are considered neighbors. Lines going between electrodes indicate selected neighbors.

4.3.3 Color Sequences

Another attempt to solve the identification of electrodes is to use the color of the labels of electrodes in addition to the positions of electrodes in the camera image. As mentioned earlier, each electrode has a plastic ring around it with the name of the electrode on it, which has one of four colors (green, yellow, blue, and gray). This additional information should help in identifying electrodes. The first version of the algorithm, which used the colors of electrodes, worked based on the neighbors of each electrode. The detection step also detects the color of each electrode. Then, the neighbors of each electrode are determined by using the distance to other electrodes, which has to be lower than a specified threshold. Figure 14 shows an example of determined neighbors for an electrode. Those neighboring electrodes are then sorted in a circular order, resulting in a list of neighboring electrodes for each electrode. This list can be used to create a color sequence. The color sequence is just a string consisting of the letters ”G” for green, ”Y” for yellow, ”P” for purple, since the gray electrodes appeared purple to the camera of the HoloLens, and ”X” for blue. This color sequence can then be compared to a list of reference color sequences, and therefore used to identify electrodes by their neighbors.

This method needs a list of reference color sequences to work. A separate Python script was used to generate these reference sequences by using the predefined 3D position file of the electrodes, with additional color information for each electrode added. First, this tool also just calculates the distance between each electrode to determine neighbors and, thus, the color sequences for each electrode. This information then gets passed into the identification step. When comparing two color sequences, it is important to consider the fact that the sequences might be shifted by several neighbors due to the starting point of the circular sorting. It is possible, for example, that the reference sequences start from point A, but the detected sequence starts at point B due to the camera being pointed at the EEG cap from different angles. This means that the comparison of sequences has to consider this rotation of the circular sequences. To combat this, we concatenate the reference sequence with itself, resulting in a sequence that contains the color of each neighbor of one electrode two times. We can then just check whether the color sequence of a detected electrode is a part of the concatenated reference sequence. If the test sequence is part of the reference sequence and has a minimum length depending on the length of the reference sequence, both sequences are considered a match and the detected electrode can be named accordingly. Figure 15 explains the concatenation of the reference sequence and shows the check whether the detected sequence is part of it.

The first version of this method has the issue that the simple substring test for checking whether two color sequences match up is too strict. It doesn’t allow falsely detected color sequences or small errors in the determination of neighbors. The second version of the code instead uses the Levenshtein string distance [20] in combination with a threshold that has to be reached for two sequences to match up, as shown in Figure 16 in an example. The Levenshtein distance between two strings is calculated by adding the amount of characters that have to be removed, changed, or added to the source string to reach the target string. To be able to still handle different starting positions in the circular color sequence, this version also concatenates the reference sequence to itself. Then, it iteratively takes a part of the concatenated sequence with the same length as the reference sequence and compares it to the detected sequence. The same can be achieved by moving the first letter in the reference sequence at the end of it in each iteration. This means it has to be checked several times for one sequence test. For each comparison, it uses the standard Levenshtein distance and stores the minimum distance over all iterations. This distance then is compared to a threshold and if the distance of the sequence is below the threshold, the sequences are considered to match.

The approach of determining the neighbors by using a simple distance metric has some problems with electrodes that appear more toward the edge of the cap in the camera frame. The distances between those electrodes appear smaller in the image due to the perspective warp, which also results in the list of neighbors of electrodes changing depending on the perspective of the camera. This means that the detected color sequence for some electrodes is not stable and can not be compared to the reference sequence anymore. It still works for some electrodes, if they are directly in the middle of the camera view, but not for electrodes more toward the edges of the camera frame.

To try to solve this issue, the detected electrode points are projected onto a hemisphere. This means that all points are now placed on a sphere, which represents the head shape to an extent. All positions are in 3D space now instead of just being 2D positions. Figure 17 shows a graph containing the projected positions. The

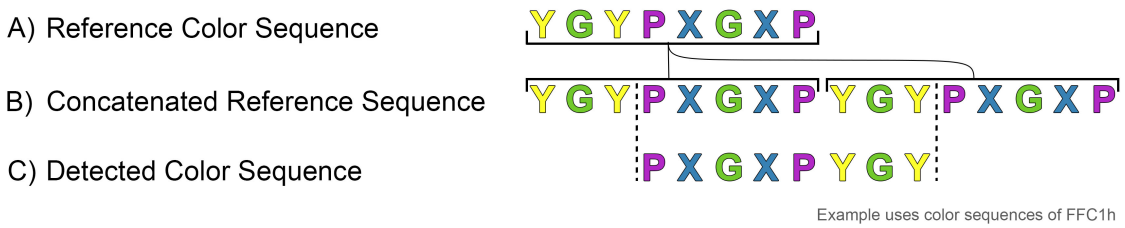

Fig. 15: Method of checking color sequences with a simple substring method. A) represents the reference color string stored for each electrode. B) represents the concatenated version of the reference string. C) represents the color sequence of the detected electrode that is checked against the reference.

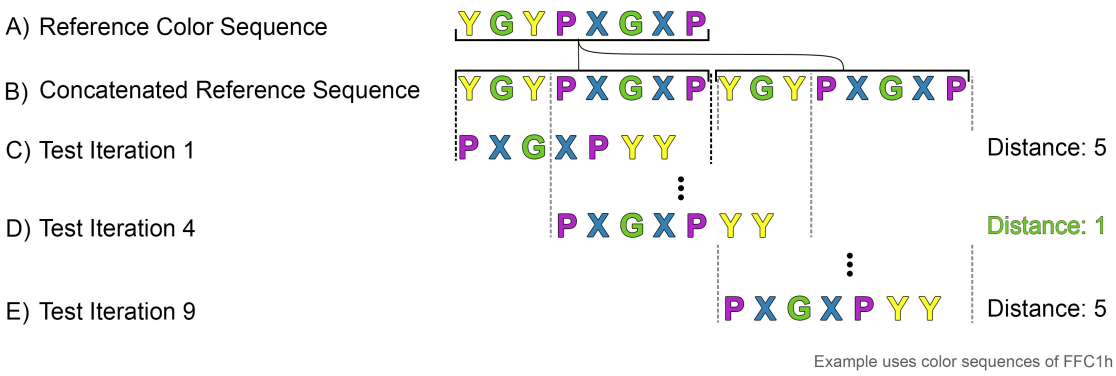

Fig. 16: Method of checking color sequences using the Levenshtein distance. The color sequence of the detected electrode contains a single error. A) represents the reference color string stored for each electrode. B) represents the concatenated version of the reference string. C) represents the first iteration, resulting in a distance of 5. D) is the 4th iteration, which results in the lowest distance.

placement can be fine-tuned by changing the size of the hemisphere. This allows the application to determine the neighbors of an electrode by using the 3D positions resulting from the projection and checking the distance to other electrodes. This way of determining the neighbors of an electrode works better than earlier attempts because it considers the sphere-like shape of the head in the distance calculations between electrodes. It also isn't affected that much by the perspective warp of electrodes more toward the edge of the cap in the camera frame.

But to use this method, the determination of the reference sequences has to be changed as well. Instead of using a simple distance to calculate the reference color sequences, the sequence generator script has to consider the sphere-like shape as well. In the new version, the generator script converts the 3D positions of the reference position file into a spherical coordinate system. Spherical coordinate systems use a distance from the origin and two angles to specify point locations. Refer to Figure 19 for an explanation of both angles and the distance. To determine the neighbors of one electrode, the script has to rotate all electrodes such that the electrode it is working on, currently, is on top of the origin. It will have X and Y coordinates that are zero and only the Z coordinate has a value greater than zero. The rotated positions can be converted into a spherical coordinate system again. One of the angles of the spherical coordinates specifies how much the point is angled from the Z-axis. This angle can be used in combination with a specified threshold to determine the neighbors of an electrode. The other angle can then be used as the sorting metric to sort the neighboring electrodes in a circular order around the current electrode. Figure 20 shows this process with example points. The circular order of neighbors can be used to determine the color sequence of the current electrode. With this algorithm, it is possible to get correct neighbors and, thus, color sequences for each electrode.

The combination of the new reference color sequences and the hemisphere positions for determining the detected neighbors works better at identifying electrodes, but still has some problems. One problem is that the determination of neighbors for each detected electrode is still not perfect, and the set of neighbors for each electrode is not as stable as it should be. It still includes different neighbors depending on the perspective of the camera. This means that the small number of electrodes that can be identified is also not reliable enough. But the biggest issue is that the colors of the labels of the electrodes are placed symmetrical front to back and left to right. That results in electrodes on the left side of the EEG cap having the same color sequence as electrodes on the right side. Therefore, there can be multiple possible electrode names for one sequence and there are only a few electrodes that have a unique color

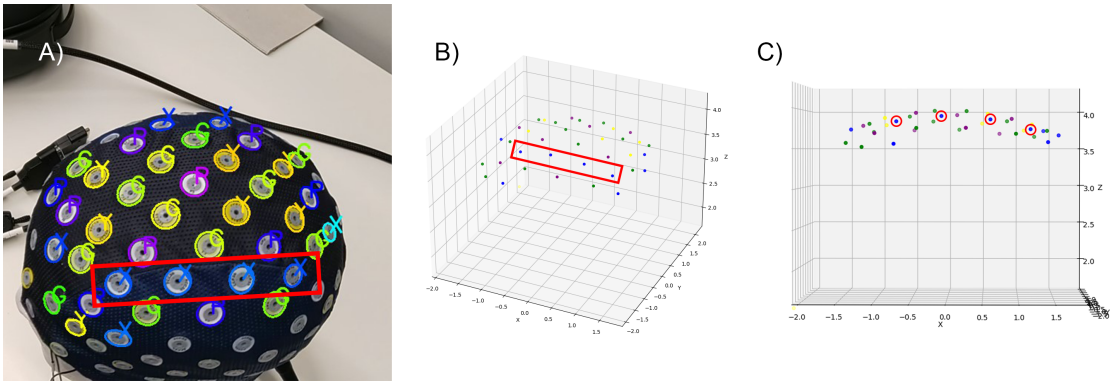

Fig. 17: Visualization of detected electrode positions projected onto a hemisphere. Electrodes marked in red bounding boxes or circles are the same 4 electrodes marked for reference. A) shows the detected electrodes from the Ellipse Detection method. The fine lines between electrodes show, which electrodes were determined as neighbors to the electrode named ”CZ”. B) shows the positions of the detected electrodes projected onto a hemisphere. C) shows projected positions from the side to show the curvature of positions.

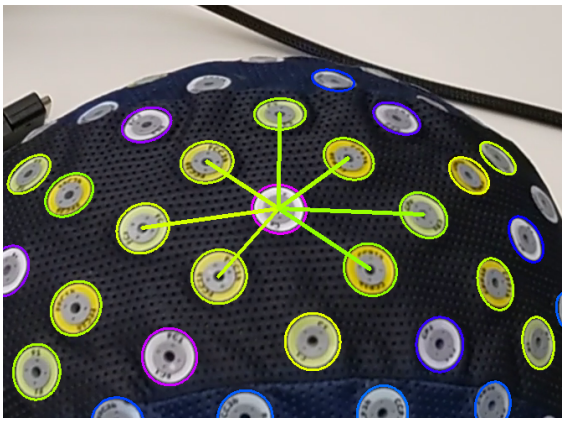

Fig. 18: Visualization of which neighbors are determined for each detected electrode by using the distances between the positions of electrodes that got projected onto a hemisphere. Lines going between electrodes indicate selected neighbors.

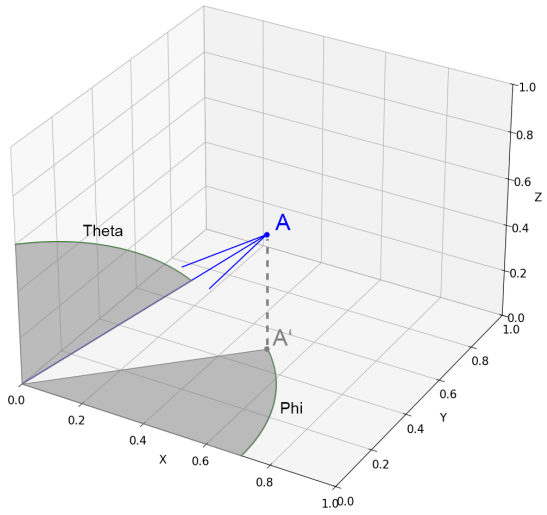

Fig. 19: A 3D graph containing a point A. The angles theta and phi describe the direction of the origin to the point A. Both angles and the distance from the origin to the point A are the three parts of a coordinate in a spherical coordinate system.

sequence. This can be fixed by increasing the number of neighboring electrodes that are considered in the color sequence. However, the more neighbors that are considered means that it has to detect more neighbors of an electrode to identify it.

One attempt to solve the assignment of names to detected electrodes was by using an algorithm to solve the assignment problem. The classical assignment problem is formulated with workers and jobs. Each worker has a different cost for each job. The solution to the problem is an assignment, which assigns each worker a job so that the total cost is minimal. The assignment problem can be adapted to this scenario by using the electrode names as jobs and the detected electrodes as workers. This means a solution to the problem, in this case, tries to assign the electrode names to the detected electrodes depending on the cost of doing so. In this case, the cost is calculated by using the color differences of neighbors and the difference in distance to neighbors compared to the reference neighbors. That means if, for one possible assignment of a name to a detected electrode, the distance to all the neighbors of that electrode is greatly different from what the reference specifies, the cost of that assignment is higher. Additionally, the difference between the color sequence of a detected electrode to the color sequence of a reference electrode is also added to the cost of an assignment. This attempt to apply the assignment problem in this scenario did not work out very well, and it wasn’t able to correctly identify detected electrodes. This probably could have been improved by modifying the cost calculations for each possible assignment.

4.3.4 *Recursive Algorithm*

The next attempt tries to solve the identification of electrodes by brute-forcing an assignment of names

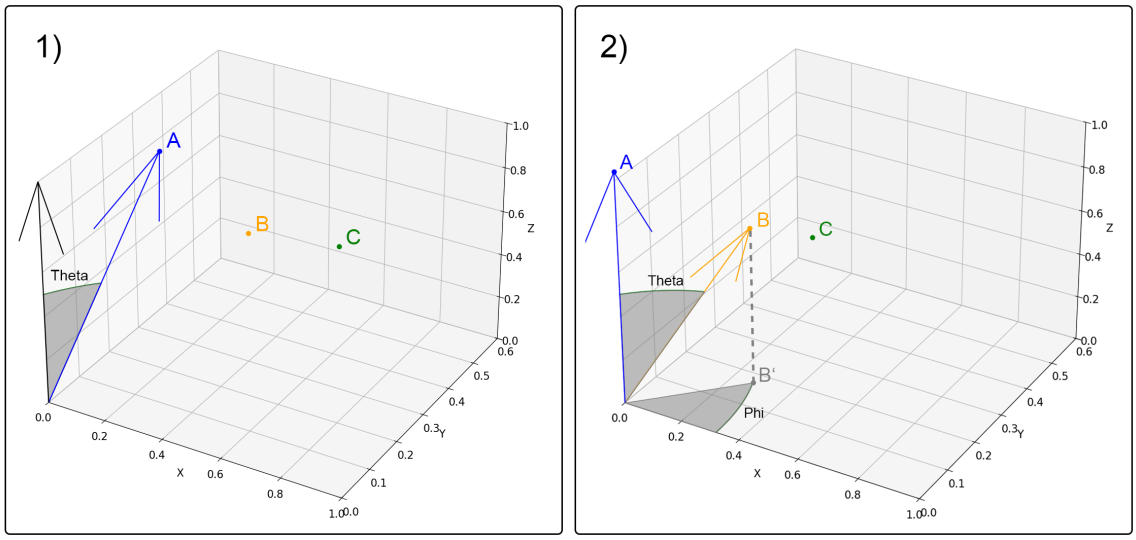

Fig. 20: 1) shows a 3D graph with three points. The angle Theta is one part of the spherical coordinates of point A. 2) shows the result of rotating the set of points such that Theta of point A equals zero. Theta of point B can now be used to determine the distance between point A and point B. The angle Phi can be used to sort neighbors of A in a circular order.

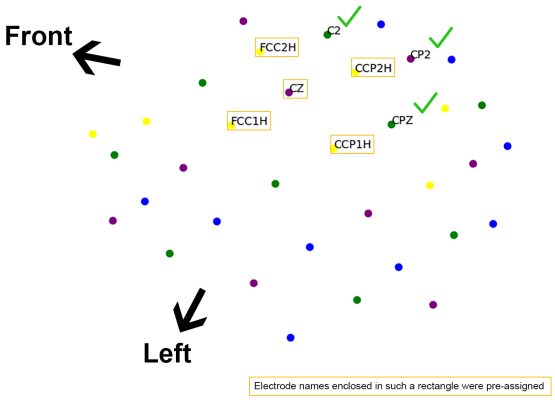

Fig. 21: Visualization of detected electrode positions and the result of the recursive algorithm. Electrodes marked with an orange bounding box were pre-assigned before running the algorithm to provide it with a starting point. All three electrodes identified by the algorithm are correct.

and detected electrodes. This algorithm consists of a recursive backtracking algorithm, which tries out every possible combination of detected electrodes and names and checks whether this combination is correct. A recursive backtracking algorithm assigns a name to a detected electrode in every step and checks whether this assignment is even possible. If it is, the algorithm calls itself recursively with the partial solution it has added to and continues by assigning more names to detected electrodes. If one partial solution is incorrect, it goes back one step, called backtracking, and tries another partial solution going forward. The algorithm runs until it has found a complete solution or no solution can be found.

In this case, the algorithm is used to construct an assignment, which assigns names to each detected electrode. In every step of the algorithm, it calculates all possible names for the next unassigned detected electrode. It then tries out every possible name for the next unassigned electrode by adding it to the partial solution and calling itself with that as input. If one solution is incorrect, it returns to the previous step and tries another partial solution. In each step, the possible names for detected electrodes are determined by using several filters. If a name has already been given to a detected electrode in the current partial solution, it ignores that name because each name can only occur once. Furthermore, it also checks whether an electrode can have a specific name depending on the color of the detected electrode and the color of the electrode with the specific name. Additionally, it also checks whether a name can be given to a detected electrode by checking the neighbors of that electrode. If there are neighbors that already have a name assigned to them in the partial solution, it can filter for its neighboring names and only check those.

This algorithm takes a very long time to finish due to its nature of brute forcing every possible assignment. The number of partial solutions it checks and, therefore, the runtime of the algorithm can be decreased by filtering out more names in each step. The algorithm also has to consider errors in the detection of colors and neighbors for each electrode, as there may be some mistakes in the detection part. It works a lot faster if some detected electrodes already have pre-assigned names as a starting point. For example, the electrode CZ, on the EEG cap used in this thesis, has a unique color sequence and can therefore be detected quite easily. If the algorithm knows this assignment from the beginning, it can filter out much more partial solutions

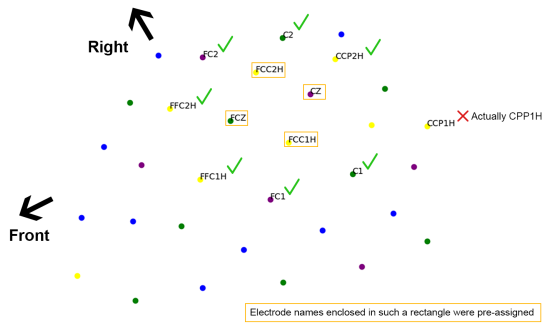

Fig. 22: Visualization of detected electrode positions and the result of the iterative algorithm. Electrodes marked with an orange bounding box were pre-assigned before running the algorithm to provide it with a starting point. Only one out of 8 electrodes identified by the algorithm is incorrect.

and finish faster. However, it still does not assign all names correctly and has the same problem as earlier methods, which is the symmetry of the colors on the EEG caps. Occasionally, it assigns the wrong name to a detected electrode even though the sequences and neighbors match up because multiple electrodes can have the same color sequence and neighbors. Figure 21 shows the result of the recursive algorithm with a predefined starting point.

4.3.5 *Iterative Algorithm*

The last attempt to solve the identification problem of assigning detected electrodes their correct names is to use an iterative algorithm. It was inspired by an algorithm used to solve Sudoku puzzles. One possible algorithm to solve Sudokus is to iteratively go over each field and check what possible numbers can be assigned to that field. In the beginning, there are some pre-assigned fields, which already have a number specified. All other fields have a set of possible numbers they can have. In each iteration, this set of possible numbers is checked, and several criteria are used to exclude possible numbers from that set. If a field only has one possible number left, it can be assigned with that number and then used to exclude more numbers from other fields. After some iterations, each field will have a number assigned to it and the Sudoku is solved. Depending on the criteria used to exclude numbers, this algorithm can exclude more and more possible numbers from each field.

The iterative algorithm used to identify electrodes works similarly but with different criteria than for a Sudoku. It also has numerous fields, in this case, detected electrodes, each having a set of possible electrode names. In each iteration, it checks every electrode and excludes possible names depending on various criteria. One criterion is the color sequence of an electrode. A detected electrode can only be one electrode if their color sequences are not too different. If the difference between the color sequence of a detected electrode and a possible reference electrode is too high, this name is excluded from the possible names of the detected electrodes. Depending on already assigned neighbors or the possible names of neighbors, it is also possible to exclude names for an electrode. Also, if there are, for example, two electrodes that only have the same two names as possibilities, these two names can not be assigned to any other electrode other than these two. Before the first iteration, each electrode gets a set of possible names it can have, depending on the color of the electrode.

This algorithm, like the recursive algorithm earlier, also takes a very long time without any pre-assigned electrodes. If there are some electrodes pre-assigned, it can exclude possible names for each electrode much more efficiently and will finish faster. But like the recursive algorithm, this algorithm won't work very well if there are errors in the detection of colors or neighbors of detected electrodes. It takes some margin of error into account, but it won't work perfectly. Additionally, it also struggles with the symmetry of colors on the EEG cap. But if it has some electrodes pre-assigned as an input it runs fast enough to be used and the results are correct enough in some scenarios. The output is still not stable and will change depending on the perspective of the camera. Figure 22 shows the result of the iterative algorithm with some electrodes already pre-identified.

5 Implementation

Although there are many attempts to solve the problem of detecting and identifying electrodes from the camera feed of the HoloLens, none of them worked reliably and consistently enough to be used in the final implementation [1]. The detection of electrodes alone works quite well with the ellipse detection algorithm. Additionally, the detection of the label color and the neighbors of electrodes works well enough and is also used in some versions. However, the methods for identifying electrodes by assigning them their correct name don't work reliably enough. Some methods can't identify electrodes at all, and some are too unstable or identify too few electrodes to be useful. The recursive algorithm and the iterative algorithm can identify electrodes correctly with some help, but both take a long time, depending on how many electrodes are pre-assigned for them. It would have been possible to use a marker between four electrodes, which could assist those algorithms by defining the names of the four surrounding electrodes beforehand. However, this meant that a marker had to be placed on the EEG cap.

Instead of putting together a method that combines the detection of electrodes and their color and markers that have to be placed on the cap, the final implementation only uses markers. It uses ArUco markers to track points on the cap, which have to be placed at predefined positions on the cap, as shown in Figure 24. The application can then use the detected ArUco

1. GitHub repository of this project: https://github.com/s-ccs/BSc2024_AR-EEG
Zenodo URL to a fork of the final state of the repository: https://doi.org/10.5281/zenodo.13208943

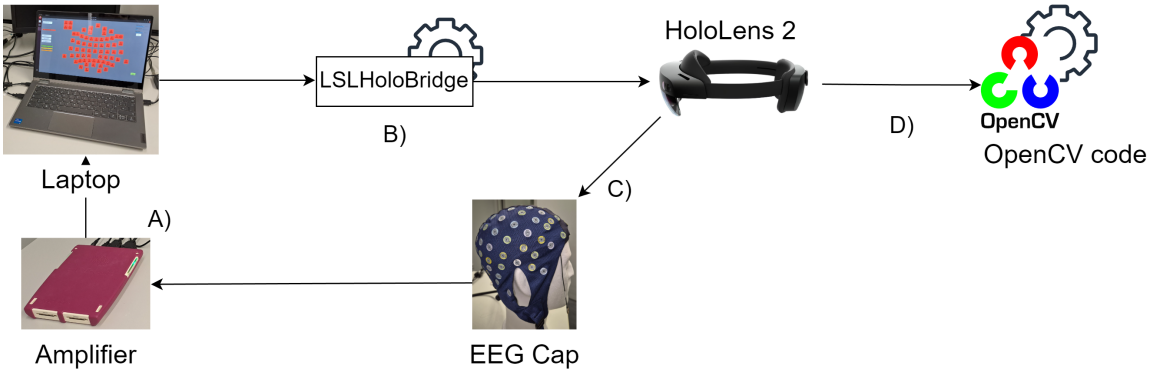

Fig. 23: Diagram containing all necessary components of the application and how they interact with each other. A) The laptop uses the amplifier to measure impedance values of electrodes on the EEG cap. B) The LSLHoloBridge script receives the LSL stream and retransmits the LSL data to the HoloLens. C) The HoloLens captures camera frames of the EEG cap and visualizes the impedance values over the electrodes of the cap. D) The HoloLens uses OpenCV to detect ArUco markers in the camera frames.

Fig. 24: Image of an ArUco marker placed at the predefined position between electrodes on the EEG cap.

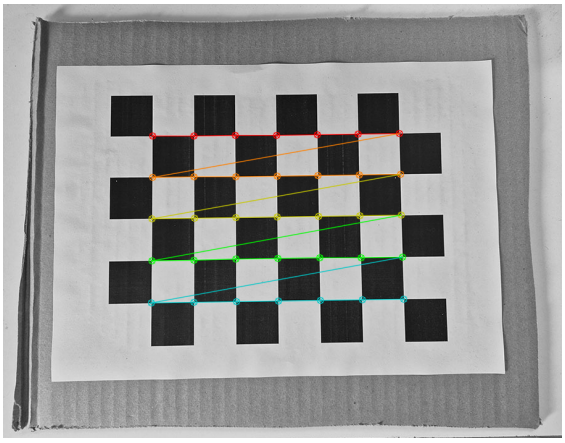

Fig. 25: Image of a checkerboard pattern glued to a piece of cardboard. Image is processed by OpenCV to detect the checkerboard pattern edges.

markers and their predefined positions on the EEG cap to align the virtual representation of the cap to the real cap accordingly. One of the earliest tests to align the virtual representation of the cap with the real one used a QR code and the built-in QR tracking feature of the HoloLens to achieve the same goal.

The detection and tracking of ArUco markers use features of OpenCV [13], which already provides several functions to track ArUco markers. The implementation of ArUco tracking in OpenCV is rather trivial and by knowing the size of the markers, it can calculate the 3D position of markers relative to the camera by using a PNP algorithm. A Perspective-n-Point algorithm uses predefined 3D points and corresponding 2D points to estimate the pose of the camera. In this case, the predefined 3D points are specified by the size of the marker and represent the edges of a marker. The 2D positions are the edges of the detected marker in an image. The PNP algorithm can now estimate the pose of the camera with both point sets, which results in a transformation and a rotation vector. Those vectors represent where the camera is relative to the marker and can be used to get the position of the marker relative to the camera.

The accuracy of the calculated 3D positions depends on the calibration of the camera. Without calibrating the camera, the image stays slightly distorted due to the lens of the camera. By calibrating the camera, OpenCV can undistort camera frames, which improves the accuracy of calculations. Camera calibration works by taking pictures of a checkerboard pattern from different angles, rotations, and distances. The checkerboard pattern should be captured in different parts of the images because distortions are more extreme on the edges of the image and OpenCV needs to have samples for all places on the image. Additionally, it is important to make sure that the checkerboard pattern is on a flat surface and can't bend around and that the images are sharp and not blurred. The checkerboard pattern has

to be as clear as possible for the calibration to be as accurate as possible. The actual calibration searches the checkerboard pattern in the images. Figure 25 shows a checkerboard pattern that has been found by OpenCV. It then uses known values like the size of each block and the number of blocks in both directions to calculate the distortions in different parts of the camera frame. This camera calibration depends on the camera and lens and, ideally, has to be done for each combination separately.

When the application is run, it starts capturing camera frames by using the PhotoCapture API of the Mixed Reality Toolkit (MRTK) [12]. Each frame is passed to the OpenCV code, which handles the detection and pose estimation of ArUco markers. The resulting transformation- and rotation-vectors are then passed back to the application. However, before the application can use those values, it needs to convert them into the correct format and unit for Unity, and it needs to transform pose data to incorporate the camera's position accordingly. The PhotoCapture API provides a camera-to-world matrix for every captured frame, which contains information about the camera's position at the time of capturing the frame. This matrix is used to transform the OpenCV vectors. All marker positions of the detected markers are being stored for later use.

The application waits until the operator has looked at each marker and the application detected all five markers, which are needed to align the virtual representation of the cap to the real cap. If all markers have been found, the application can use the positions to scale, rotate, and transform the virtual representation of the cap so that it overlays the real cap. It uses the predefined marker positions to calculate the scale in each direction and the rotation in each direction to align the virtual cap as close as possible to the real cap. In case the automatic alignment is not good enough, the position, scale, and rotation of the representation can be fine-tuned by sliders in the user interface of the application.

The virtual representation of the EEG cap can now be used to visualize values over the real electrodes. However, to get the appropriate values, the EEG LSL stream has to reach the HoloLens. Due to the incompatibility of the LSL library with ARM64 systems, this has to go through a companion computer. The companion computer uses the *LSLHoloBridge* project [9], which provides the necessary scripts to stream LSL data to the HoloLens. It uses the LSL library itself to discover available LSL streams and to receive those streams. It then repacks the received values in a format that the HoloLens can understand. The HoloLens has to connect to the companion computer, and the bridge script running on it. The script will send the repacked LSL data to the HoloLens, which can unpack the values and visualize them.

6 Discussion

6.1 Results

The topic of this thesis is to develop a method for detecting electrodes from a camera image and identifying them by assigning names to the detected electrodes. This information is used by an application that leverages augmented reality (AR) using the Microsoft HoloLens 2, an AR headset, to visualize data from the preparation step of EEG caps. Some EEG caps rely on good impedance values for good measurements, like gel-based EEG caps. It visualizes impedance values measured by the EEG amplifier in AR at the positions of the electrodes of the cap so that the user wearing the HoloLens can see those values directly overlaid over the electrodes of the cap. An example of the overlay is shown in Figure 3. This should help the operator in finding electrodes that have a high impedance. It makes it easier to see impedance values compared to the traditional lab setup with a separate device displaying those values. This reduces the time required in the preparation step and could also improve the overall quality of the contacts between electrodes and the scalp and, thus, the quality of the EEG measurements.

The application is based on the Unity game engine, which already provides many features like a working rendering pipeline for 3D objects and scenes and several tools around it to help in development. In addition to that, the Mixed Reality Toolkit (MRTK) from Microsoft [12] adds various necessary tools to develop AR applications for AR headsets like the HoloLens and helps integrate features provided by those AR headsets. It provides pre-made components that help set up the project for AR purposes and make the interaction between the user and the AR application possible. With those tools, it is possible to create visualizations that are displayed in the user's view and with which the user can interact.

The Lab Streaming Layer (LSL) [8] is used to stream EEG measurements between different computers over a network. This allows multiple computers to work with the data recorded with the EEG amplifier. However, since the LSL library does not support the HoloLens 2 [10], the LSLHoloBridge project [9] was used to stream the EEG LSL Data from the LSL stream to the HoloLens using a computer in between. This allows the HoloLens to receive the necessary data from the EEG LSL stream without needing the LSL library itself.

The positions of the electrodes on the cap are needed to be able to overlay visualizations over them. The first step to determining the positions is to detect the electrodes on the EEG cap. The camera of the HoloLens is used to capture frames, which then get passed to OpenCV [13]. OpenCV is used to process the camera frames and to detect electrodes in the camera frame. Several methods have been tried to achieve the required detection, like detecting circles in different ways.

A second step is required to gather the necessary positions of the electrodes. The application needs to know which detected electrode has which label to be able to determine the correct value that has to be

visualized over each electrode. It needs to identify detected electrodes and assign them their respective name, which can be used to get the correct value for each electrode. There are several attempts at solving this problem, including feature-matching algorithms and multiple methods that use the color information of the electrodes and their neighbors to try to identify them, like the color sequence matching, the recursive backtracking algorithm, and an iterative approach to solve the problem.

The final version of the application uses five ArUco markers to determine predefined positions on the EEG cap. The application can use those detected markers and their positions to stretch, rotate, and move a virtual representation of the EEG cap so that the electrodes of the virtual representation overlay over the real electrodes on the EEG cap.

6.2 Interpretation

The use of Unity as the base for the application in combination with the MRTK turned out to be very helpful. Unity already provides several features for displaying 3D objects and scenes, which helps a lot in creating visualizations and user interfaces (UI). The integration of MRTK and the HoloLens in Unity provides several additional features that help in developing and debugging an application. It is possible to run the application on a computer, but stream the screen to the HoloLens. With this, you can use the HoloLens to see the 3D scene and interact with it. Additionally, this allows for easy debugging of errors on the computer. One problem with that is that the application is technically not running on the HoloLens and several features like the camera of the HoloLens are not accessible this way. To get access to all features of the HoloLens in the application, it has to be compiled in Unity and then run directly on the HoloLens. It is still possible to debug the application, but this process takes much longer in each iteration. If the network connection from the computer to the HoloLens isn't fast enough, this process can take very long.

Additionally, MRTK provides several UI components that can be used to create user interfaces. I have used components like windows, buttons, and sliders to assemble the user interface of this application. It was quite easy to integrate the UI components into my code and even create my own components that used premade scripts from MRTK. This allowed me to enable the interaction between the user and the application quite easily and fast.

At the beginning of the project I did not anticipate that, depending on where you want to run the application, you need to consider the architecture of this system. If I wanted to test the application on the local computer, I needed libraries to be compiled for x64 systems. But if I wanted to test the application on the HoloLens, all libraries need to be compiled for ARM64 and the Universal Windows Platform (UWP) SDK. This meant that I had to compile OpenCV itself and my own OpenCV code for both architectures.

There were several attempts at solving the problem of detecting electrodes of the EEG cap from a camera image without using markers. Some of them are too unreliable or don't detect electrodes at all. The feature-matching method uses feature-matching algorithms to find electrodes on the camera image with the help of features extracted from a reference image. The reference image only contains a single electrode. The feature-matching algorithms were not able to detect any electrode, except if the reference image originated from the search image directly. This is probably due to the large difference in the appearance of electrodes in the camera frame and the different perspectives of the camera depending on the angle it has to the EEG cap. Similarly, the template-matching, which also uses a reference image of a single electrode to find electrodes in the camera frame, did not work for the same reasons. Both algorithms are designed for another use case and don't work properly for this scenario.

The different types of circle detection methods also had some problems in detecting electrodes reliably. Because they are designed to find circles and, thus, provide only the position and radius of found circles, they were not able to provide enough information to filter out false detections. Depending on the viewing angle of the camera, electrodes appear more squished toward the edges of the cap. The application uses the attributes of detected ellipses, like the moment of inertia of the ellipses, to filter out falsely detected electrodes.

The best method of detecting electrodes is the ellipse detection. It also uses contours to find shapes in the camera image and then tries to fit an ellipse to match the shape as closely as possible. This allows the application to have additional information about the shapes and can filter false positives quite reliably. There are still some false positives detected in the end. The ellipse detection works reliably enough in the center of the camera frame. It only struggles with electrodes that appear more toward the edge of the EEG cap due to the perspective warp and the visibility of the label of the electrodes. However, this method of detecting electrodes works due to the dark color of the cap and the bright color of the electrodes. It might not work as reliably with another type of EEG cap.

The problem of identifying electrodes turned out to be more of a challenge than I expected. I originally thought it was easier to map names to the detected electrodes based on the layout of the electrodes on the camera frame. But that didn't work at all, and even using additional color information for each electrode did not help. The last attempt with the iterative algorithm worked better in the testing scenarios, but still isn't reliable and fast enough to be used in the application. The issue with most methods was that they could not match the correct names to the detected electrodes.

Methods using the feature-matching algorithms could not identify electrodes at all. The problem with this might be that I created the feature points myself with only position information and without any additional context, which those algorithms need, like pixel values around the key points. Feature-matching

algorithms are designed to work with features extracted from images and, in my case, I tried to use them with positions I extracted from the reference 3D positions file for this specific EEG cap. Those features don't include any additional information and are not made for this kind of algorithm.

The iterative closest point (ICP) algorithm also isn't designed to handle the data I provided. It is designed to work with point clouds, which contain sets of points that are in both sets at the same time and overlap quite accurately. In my case, the detected positions from the camera frame and the positions from the 3D positions file don't overlap at multiple points at the same time. The points need to be moved around individually to match up correctly, since the positions from the 3D positions file don't contain the exact position of the electrodes on a head shape. The ICP is not designed for this, as it only calculates a single transformation vector for all points at the same time and does not transform points individually.

The detection of colors of the electrode labels worked quite well in this situation due to the lighting conditions in our environment and the camera of the HoloLens. An image captured with the camera of the HoloLens in our lighting conditions lets the colors appear vibrant. The vibrant colors are easier to extract from the image. This also means that the detection of colors might not work properly in other environments. Especially the green and yellow values of the cap used in this thesis are quite close depending on the angle the camera has to the electrodes. Some adjustments of the color hue thresholds in the code might be necessary to detect the colors correctly.

However, using the color information didn't help with the identification process as much as I imagined. The distribution of the colors on the cap is quite repetitive and symmetrical. This means that there are several electrodes with the same patterns of color around them. This hasn't helped in the identification of the electrodes. If the colors were distributed more randomly, more of the color sequences would have been unique and better suited to identify electrodes more reliably. The simple color sequence check, which just checked the nearest neighbors of each electrode, was able to detect some electrodes. However, it also often misidentified the electrodes with the wrong label of an electrode from the other side of the cap due to the symmetry of the colors.

The same problem with misidentifying electrodes due to the symmetry of the cap occurred in the recursive and iterative algorithm. I was able to work around it by providing five pre-identified electrodes from the top of the cap before running the iterative algorithm. It could use those five predefined electrode names to work out the correct side of electrodes and assign most of the rest of them correctly. But this also means that the algorithm only works correctly if it has the pre-identified electrodes from the top as an input, which are located at the top of the cap. This wouldn't work if the camera is directed at the back of the cap and the electrodes on top of the cap aren't visible.

Additionally, the algorithms don't handle mistakes in the input very well. If the color of an electrode has been detected wrong or if the neighbors of an electrode got determined differently than in the reference data, the algorithm is unable to detect this electrode or, in some cases, electrodes around it. These errors can occur on electrodes that are more toward the edge of the cap and some neighbors it expects might not have been detected correctly.

The current version uses five ArUco markers on the left, right, top, front, and back so that the application can not only align the cap's position but also its rotation and scale in all three axes. This wouldn't be possible with the QR code since only one marker won't give the application enough data to calculate the scale in each direction individually as the scale is different for each head shape. For the scale calculations, the application needs two markers in all directions to calculate the scale for each direction. In this case, the bottom point is calculated by using the center of the front and back marker, as it is impossible to place a marker in the patient's neck.

6.3 Limitations

Even though Unity and MRTK provide helpful tools for (remote) debugging applications, the application still has to be compiled for the HoloLens and then deployed to it. For access to the camera, applications have to be run directly on the HoloLens. If the network connection between the computer and the HoloLens is not perfect, the deployment takes very long. In my case, it had to upload around 200 MB of data to the HoloLens almost every time, which took about 15 minutes. In an attempt to fix the long upload times, I opened up a Wi-Fi hotspot on my computer, which has Wi-Fi capabilities, and connected the HoloLens to it. But this turned out to be a hassle as well because most of the time the HoloLens didn't show the hotspot and didn't want to connect to it in case it had found it. But after the HoloLens was connected to the hotspot of my computer, it was able to deploy the application in about 3 minutes, which let me iterate much faster.

Also, many of the features MRTK provides are not very well documented. The documentation on the internet is mostly for MRTK2 [11] and not completely applicable to the new version, MRTK3 [12]. The documentation of the new version is missing some features I wanted to use. UI components and their scripts are not very well documented as well. I had to play around with the provided scripts and check them out to understand them and how to use them. The documentation of the MRTK3 could be better.

Another hurdle in the development was the architecture of the HoloLens 2. It is based on the ARM64 architecture and only runs Universal Windows Platform (UWP) applications, which means everything has to be compiled specifically for it. I had to compile OpenCV specifically for the HoloLens, which took some time to figure out how to do correctly. Additionally, there are libraries, which don't exist for this architecture and I

couldn't compile myself. The LSL library does not exist for ARM64 architectures, and I wasn't able to compile it for this architecture myself, due to it depending on Windows APIs that are not available on the HoloLens 2. This meant I had to find a way of getting the LSL stream data to the HoloLens without using the official LSL library. The current solution with the LSLHoloBridge works but needs a script running on a separate computer.

Furthermore, the final version of this application is designed to specifically work with the EEG cap used in this thesis. The detection of electrodes relies on the dark color of the cap and the fact that the electrodes stand out from it. The detection of electrodes and their color also depends very heavily on the colors of the labels and the specific lighting conditions in our environment, which lets the colors appear more vibrant for some reason.

Due to the unfinished state of the application and time limits, a study on the effectiveness of the application has not been performed. It wasn't possible to test whether the proposed application helps in improving the preparation time of EEG caps in real-world scenarios. It can visualize impedance values on top of the electrodes, but those visualizations are not necessarily aligned perfectly, which might result in a much lower preparation time difference than anticipated at the beginning of this thesis. Additionally, the extra preparation steps to prepare the HoloLens, the ArUco markers on the cap, the alignment of the virtual representation of the cap, and the preparation of the LSL bridge computer add additional preparation time. Depending on the number of electrodes on the cap and the skill level of the operator, these additional steps might increase the time needed to prepare an EEG cap instead of decreasing it.

The use of additional hardware like the HoloLens and the ArUco markers leads to additional prerequisites that have to be taken care of. The HoloLens needs to be charged as its battery does not last very long. The HoloLens, the computer streaming LSL data to it, and the ArUco markers also have to be prepared for experiments. The required preparation of the additional hardware adds additional failure points and might delay experiments. It could also lead to the operator having to resort to conventional methods for the preparation of the EEG cap without this application and the HoloLens.

6.4 Future Work

Despite the many failed attempts at implementing a method for reliably identifying electrodes from the camera image, I still think it is possible to solve this problem. It seems like it shouldn't be impossible to find a method that can work with this scenario and can identify electrodes correctly based on the layout of the electrodes and how they appear (in which positions) on the camera frame. It seems like a problem that should already have a solution to it.

If a method of identifying electrodes from a camera image can be found in future work, the detection, and identification methods can be used to determine the names and positions of electrodes on the EEG cap. Those can then be used to visualize the impedance values more accurately on the electrodes of the cap, which should improve the helpfulness of this application. If the method can work without any markers, it would also remove one of the extra steps of using this application and would render printing and applying ArUco markers at the correct positions unnecessary. This would greatly improve the effectiveness of this application.

Fig. 26: Examples of possible visualizations. A) several types including the notification pin. B) the current version, which consists of a ring around the electrode that changes color depending on the impedance value. C) a bar that fills up and changes color depending on the impedance value.
Source: A) Image used with permissions from Benedikt Ehinger

Currently, the application only contains a single type of visualization, a colored ring around each electrode. There have been some ideas on different visualizations, which would need to be evaluated to determine an appropriate visualization that is informative and not distracting or obstructive. One idea was to use a bar instead of a ring, which fills up depending on the value. It could also change color, like the ring, to show the value in the form of its color and also its fill level. Additionally, it would be possible to show an indicator for electrodes that are not visible from the current point of view to make it easier to find electrodes with high impedance. This indicator could be a warning icon and a line that indicates which electrode is meant. This could work for electrodes on the back side of the cap, which are currently not visible. Figure 26 shows examples of some possible visualizations. It could also help

more skilled personnel to show the absolute impedance values next to the electrodes. This would allow the operator to evaluate the impedance values without using predefined colors of the application.

However, it would also be possible to display other values, like the actual voltage measurements, after the cap has been prepared. This would show which electrodes have high activity on the actual head of the patient, instead of a 2D graph without any relation to the position of the electrodes on the head. Brain activity could be visualized by overlaying a heatmap over the head, representing the activity measured from the electrodes. A heatmap wouldn't necessarily need a perfect alignment of the virtual electrodes to be useful and could visualize spatial associations between the EEG data and the actual head.

This application could also be used to help align the cap on the head in the first place. Before even inserting the gel into the electrodes, it is important to align the cap and, thus, the electrodes at the correct positions on the head. Conventionally, this is achieved by using anatomical landmarks as a reference. The application could, for example, use face detection algorithms to determine the position of facial landmarks that can be used to identify the correct cap position on the head. It could then guide the operator in aligning it correctly on the head. This could also include external electrodes that are not integrated into the cap itself, but rather are attached to it with wires. This process could also include guidance on which electrodes, like the ground and reference electrodes, have to be prepared first and how.

7 Conclusion

The goal of this thesis was to develop a method of detecting and identifying electrodes from camera images. The proposed application of this thesis uses positions and labels of detected electrodes to help a person in the preparation of EEG caps. It uses Augmented Reality (AR) to visualize impedance values over the electrodes themselves, which lets the operator see important values directly without needing to look at a separate device.

Different methods for detecting electrodes from a camera image and identifying those detected electrodes have been tested and evaluated for performance and reliability. There were several methods for detecting electrodes tested, which are designed to find circles in images. Those methods don't work for detecting electrodes, since electrodes may appear as ellipses in the camera image due to the perspective warp. The method of using detected contours in the image and fitting ellipse shapes that match these contours as close as possible has the most promising results. It uses parameters of the fitted ellipses to filter out false detections and can detect nearly all of the electrodes that are reasonably visible in the camera image reliably enough for the application.

The determination of the color of the labels of electrodes by using the mean hue value of the label in the image works in most cases. It is possible due to the combination of the camera of the HoloLens and the lighting conditions in our environment. The hue values of the colors green and yellow are similar, which means that these colors get detected incorrectly in some cases. Although, in most cases, the color determination works reliably enough as well.

The methods for identifying labels of detected electrodes do not work reliably enough. Methods like the feature-matching algorithms and the ICP algorithm are not designed for this application. Those methods can't identify any electrode and don't perform reliably enough to be used in the application. The method using pure color sequences of the neighbors of detected electrodes can't detect many electrodes over multiple consecutive frames and also doesn't work reliably enough. Even small errors in the determination of neighbors of electrodes and, therefore, the color sequence of electrodes lead to this method not identifying electrodes.

The recursive and iterative algorithm work better in that regard. However, both need a starting point of already pre-assigned electrodes from which they can go on to identify others. The iterative approach can detect more electrodes than the recursive approach. It could identify approximately twice as many electrodes as the recursive approach while having less pre-assigned electrodes as the starting point. Despite those results, the iterative approach still depends on a mostly correct determination of neighbors and color sequences for each detected electrode. Due to this, the results of the iterative approach are not reliable enough to be used in the application.

The alignment of the virtual representation of the EEG cap to the real one by using ArUco markers does work for a rough alignment. It uses five ArUco markers and their predefined positions to calculate position, rotation, and scale in each of the three axes. The application can use those to align virtual electrodes with the real electrodes. This approach is not perfect and the alignment does not work for all electrodes. However, this alignment can be improved by adjusting the predefined positions for the virtual electrodes.

It might be possible to combine markers with one of the identification methods to determine a starting point, which helps in identifying all other electrodes. If a method can be found that can identify electrodes without the need for markers, it could remove the need for markers that have to be prepared before the application can be used. This would help in removing preparation steps and decrease the time required for the preparation. However, the current version adds additional preparation steps, like the markers, the LSL-HoloBridge, and the HoloLens itself.

No study was performed to test the effectiveness of the application. It is unknown whether the application helps in improving the preparation time of EEG caps, or what the difference is between using the application and using conventional methods for preparation. The application adds additional preparation time required for the setup of the HoloLens, the ArUco markers, the alignment procedure, and the setup of the LSL stream.

Depending on the number of electrodes and the skill level of the operator, this additional preparation time can increase the overall time required to prepare an EEG cap.

Acknowledgments

I would like to express my gratitude to Jun. Prof. Benedikt Ehinger and M.Sc. Vladimir Mikheev for proposing the topic of this thesis and for supporting me and giving me new ideas during the thesis. I would also like to thank Prof. Dieter Schmalstieg, who proposed using colors and the color sequences of electrodes to identify them.

References

[1] ANT neuro waveguard™ original Product Page. https://www.ant-neuro.com/products/waveguard_original. Accessed: 2024-07-26.

[2] ANT neuro's Webpage. https://www.ant-neuro.com. Accessed: 2024-07-26.

[3] Brain Products actiCAP active Electrodes Walkthrough. https://pressrelease.brainproducts.com/active-electrodes-walkthrough/. Accessed: 2024-07-25.

[4] Brain Products actiCAP Product Page. https://www.brainproducts.com/solutions/acticap/. Accessed: 2024-07-25.

[5] Brain Products Webpage. https://www.brainproducts.com. Accessed: 2024-07-25.

[6] Electroencephalography Wikipedia Article. https://en.wikipedia.org/wiki/Electroencephalography. Accessed: 2024-07-25.

[7] Godot Webpage. https://godotengine.org. Accessed: 2024-07-26.

[8] Lab Streaming Layer GitHub Repository. https://github.com/sccn/labstreaminglayer. Accessed: 2024-07-26.

[9] LSLHoloBridge GitLab Repository. https://gitlab.csl.uni-bremen.de/fkroll/LSLHoloBridge. Accessed: 2024-07-27, Used Commit (Hash): a1efe0d1cee984c8bddc253385aa087db1a22074.

[10] Microsoft Hololens 2 Product Page. https://www.microsoft.com/de-de/hololens. Accessed: 2024-07-26.

[11] Mixed Reality Toolkit 2 Developer Documentation. https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05. Accessed: 2024-07-26.

[12] Mixed Reality Toolkit 3 Developer Documentation. https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk3-overview/. Accessed: 2024-07-26.

[13] OpenCV GitHub Repository. https://github.com/opencv/opencv. Accessed: 2024-07-26, Used Version: Release 4.10.0.

[14] OpenCV Webpage. https://opencv.org. Accessed: 2024-07-26.

[15] Unity Webpage. https://unity.com/de. Accessed: 2024-07-26.

[16] Unreal Engine Webpage. https://www.unrealengine.com/de. Accessed: 2024-07-26.

[17] Youtube Video of the S-CCS explaining the setup, experiment and cleaning procedures for gel-based EEG caps. https://www.youtube.com/watch?v=CbKPxwYPV9g. Accessed: 2024-07-27.

[18] Patrique Fiedler, Carlos Fonseca, Eko Supriyanto, Frank Zanow, and Jens Haueisen. A high-density 256-channel cap for dry electroencephalography. *Human Brain Mapping*, 43(4):1295–1308, 2022. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/hbm.25721, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/hbm.25721, doi:10.1002/hbm.25721.

[19] Seungchan Lee, Younghak Shin, Soogil Woo, Kiseon Kim, and Heung-No Lee. Dry electrode design and performance evaluation for eeg based bci systems. In *2013 International Winter Workshop on Brain-Computer Interface (BCI)*, pages 52–53, 2013. doi:10.1109/IWW-BCI.2013.6506627.

[20] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady*, 10:707–710, 1965.

[21] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999. doi:10.1109/ICCV.1999.790410.

[22] Jaakko Malmivuo, Sari Ahokas, and Toni Välkky. High-resolution eeg recording system using smart electrodes. In *2014 14th Biennial Baltic Electronic Conference (BEC)*, pages 21–24, 2014. doi:10.1109/BEC.2014.7320546.

[23] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011. doi:10.1109/ICCV.2011.6126544.

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

Date and Signature: