Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
Informatik und
Wirtschaftsinformatik

HKA

Bachelor Thesis

Name:	Fabian Rapp
Topic:	Development of an App to record RGBD Data for AI-based Diagnosis and Monitoring of Infant Head Deformation.
Place of work:	inovex GmbH, Karlsruhe
Supervisor:	Prof. Dr.-Ing. Laubenheimer
Co-examiner:	Prof. Dr.-Ing. Vogelsang
Deadline:	30/06/2022

Karlsruhe, 29/03/2022

The Chairman of the examination committee

H. Kö

Prof. Dr. Heiko Körner

Declaration of Authorship

Hereby, I declare that I have composed the presented thesis independently on my own and without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such.

Karlsruhe, July 1, 2022 . . . .

Abstract

Up to 20 percent of all infants under the age of one suffer from an asymmetric head deformation that can be diagnosed by a physical therapist using a calliper to measure the infant's head. However, this technique is prone to human error in terms of accuracy. In this thesis, a technical prototype that explores a smartphone-based diagnostic technique by creating a 3D model of the infants' head, is further developed. The RGBD data for this 3D model is captured with the TrueDepth camera system on the front of iPhones. Due to the camera system's location, the iPhone screen is turned away from the user, which poses a challenge to the user to correctly position the device when taking a scan.

In this thesis, the technical prototype is developed into an iOS app that provides positioning and progress feedback so that the user can take a scan despite the described challenge. On top of that, the app is designed to be easy-to-use and intuitive.

The effectiveness of the feedback concept and the general usability of the app is evaluated by a team of physical therapists that test the app in a real-life setting. These tests showed that the developed feedback concept indeed helped the user to take a good scan without properly seeing the screen. Giving feedback about the distance to the head and the progress of the scan was perceived as very effective. The feedback about how to position the iPhone in a two-dimensional way was more difficult to follow and might need further development.

Contents

List of Figures

Acronyms

CVAI	Cranial Vault Asymmetry Index.
GPU	Graphics Processing Unit.
IDE	Integrated Development Environment.
ISRG	Intelligent Systems Research Group.
RGBD	Red Green Blue Depth.

1 Introduction

1.1 Motivation

Up to 20 per cent of all infants are affected by a head deformation, medically referred to as plagiocephaly [41]. Plagiocephaly is an asymmetric deformation of the skull of an infant which mostly develops due to a preferred sleeping position of the infant. As the skull is still flexible at this stage of their development deformations can develop and need to be treated before their skull starts growing together and hardens. Correct treatment within the first eight months of their life is effective [41]. Depending on the severity of the plagiocephaly there are different treatment possibilities ranging from a simple change of sleeping positions to specifically formed pillows to helmet therapy [36]. If this condition remains untreated it leads to impairments in the sense of balance because of the shift of the ears and might also affect parts of the brain [41]. The most common technique to diagnose plagiocephaly is to use a calliper to measure the diagonal lengths of each side of the head and use these measurements to calculate the Cranial Vault Asymmetry Index (CVAI) which defines the severity of the plagiocephaly. However, due to human error, these measurements may not be 100% correct [36]. Based on an existing technical prototype it is the goal of this thesis to further develop an iOS application that enables a user to diagnose plagiocephaly and monitor the effectiveness of the treatment. The app performs a 3D reconstruction of the infant's head by utilizing depth information which is captured with the TrueDepth camera system of the iPhone.

1.2 Context

This bachelor thesis is conducted as part of the research project InferMod3D which is led by the Intelligent Systems Research Group (ISRG) of the University of Applied Sciences Karlsruhe in cooperation of the companies inovex and Varilag. This project started in 2020 and is estimated to terminate in 2022. It is funded by the Ministry of Science, Research and Arts of Baden-Württemberg. The partner company Varilag is a physical therapy company that specializes in treating plagiocephaly with a specially designed pillow to control the sleeping position of an infant and therefore reverse the deformation of the head. Varilag is also the company that came up with the idea for a smartphone-based diagnosis and monitoring solution. The ISRG is leading this research project and is also responsible for the algorithms to compute a 3D reconstruction of the head with point clouds and for an initial prototype of the app. Finally, inovex is an IT service provider that offers its services in the areas of applications, analytics and infrastructure, which include mobile apps, web frontend, backend, data science etc. It has seven locations across Germany with 450 employees. In this research project, inovex will support the development of the mobile application.

1.3 Relevance and Importance

The goal of the entire research project is to develop an app that allows both physical therapists and parents to diagnose and monitor plagiocephaly with just their iPhones equipped with FaceID. Eventually, the aim is to start a clinical trial to medically certify the app which means the app can be prescribed by doctors to their patients. This would be the first medically certified app for diagnosing and monitoring plagiocephaly. The version of the app that is developed within the scope of this thesis is a major step toward that goal as it serves the purpose of collecting large amounts of data of infant head deformations to further train a model. This model enables a precise calculation of the CVAI. On top of that, a smartphone-based solution is completely non-invasive. Creating a 3D model of the infant's head with other medical tools most times requires Computed Tomography (CT) and Nuclear Magnetic Resonance Imaging (MRI)[4]. While being extremely accurate they bear the disadvantage that the patient has to remain completely still. As the patient in the case of plagiocephaly is an infant that can become a difficult undertaking and therefore

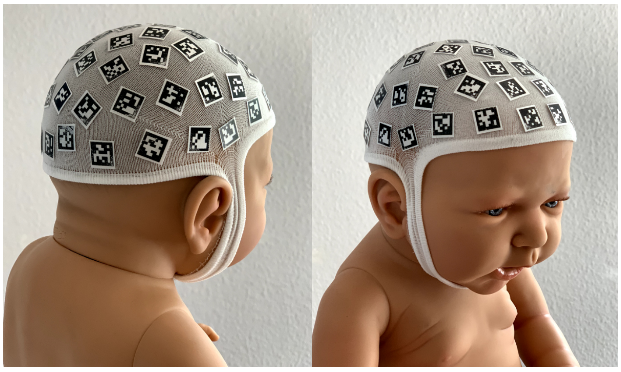


Figure 1.1: A doll wearing a cap with markers that was used for testing.

sedation of the infant is usually needed.

Regardless of the version of the app, the great advantage of it is its accessibility to a lot of people and its ease of use.

First, iPhones with FaceID sensors have been available since late 2017, so at the time of writing this thesis, for more than four years, which improves the likeliness that a physical therapist or a parent owns such a device. This means that a lot of people already have all the equipment in their pockets. And for the first version of the app, the physical therapist already possesses the most expensive part of the necessary equipment. On top of that, the infant needs to wear a special cap during the scanning process (see 1.1).

Second, a user-friendly application also provides the benefit that no special training is needed for its use, which means that both physical therapists and parents can use it. For a future version of the app, this also reduces the number of necessary appointments that have to be made, as the parents can now monitor the effectiveness of the treatment from the comfort of their own home and don't need to visit a physical therapist each time.

1.4 Objective and Research Question

The objective of this thesis is to significantly enhance an iOS application that allows physical therapists to diagnose and monitor plagiocephaly. This includes the development and implementation of a user experience concept to provide feedback to the user while performing a scan. The app is intended to be ready-to-use and easy-to-operate by a physical therapist to collect data from infant heads, which will then be further used for training 3D models in the research project. This app allows a user to record a scan and get real-time feedback during the scan regarding its quality. Furthermore, a user can select a previously recorded scan for replay. If needed there is also the possibility to rename the filenames of the recorded scans.

This thesis aims to answer the following research questions:

How can an iPhone provide feedback to a user, so that a user makes a better scan of the head of a potentially not cooperating infant?

What is a highly usable user experience design implemented by an easy-to-use user interface of an iOS app that is used while working with infants and operated by parents and physical therapists?

1.5 Overview of the Structure

The first chapter of this thesis serves as a motivation for the entire project, while also giving an overview of the context in which this thesis is conducted in and what it aims to achieve.
The basics chapter will provide an overview and basic understanding of the framework SwiftUI and the Metal-API from Apple. This knowledge is necessary to understand how the views and the 3D-point clouds are implemented. On top of that, it will outline further basic information that is needed for the understanding and setup of this project.
The following chapter describes the preconditions that this thesis builds upon. This ensures a clear picture of what was already present at the start of this thesis and what has been added later. To understand what functionality the app offers multiple usage scenarios are presented. As the purpose of this app is to make scans of the heads of infants with the front-facing camera of the iPhone, one major challenge is that the phone screen might not always be visible. Therefore it is necessary to develop a concept for interacting with the user during a scan, which forms the content of the second part of this chapter. It will examine different approaches for achieving the best possible interaction.
Having in mind what preconditions had to be dealt with and why a certain concept for giving feedback to the user was chosen, the implementation chapter outlines the overall structure of the app. This chapter provides a detailed insight into how the different views are built and how the logic works, as well as how they interact with each other. To evaluate the effectiveness of the implemented features, the app will be tested and these tests will also be presented and evaluated.
The final chapter of this thesis includes a conclusion and summary of the initially set objective and how it could be implemented. Furthermore, it will give an outlook of how this app can be extended further in future work.

2 Basics

2.1 SwiftUI

Apple provides two frameworks for building iOS apps. UIKit [29] and SwiftUI [28]. UIKit is the most popular framework and has been present for many years. However in 2019 Apple introduced a new framework, called SwiftUI. Ever since more and more features have been implemented and iOS features like Widgets are even SwiftUI exclusive.
The technical prototype of the scanning app is implemented using UIKit, but the version that is developed in this thesis uses SwiftUI. This chapter will provide a basic understanding of the SwiftUI framework to ease the understanding of the final implementation.

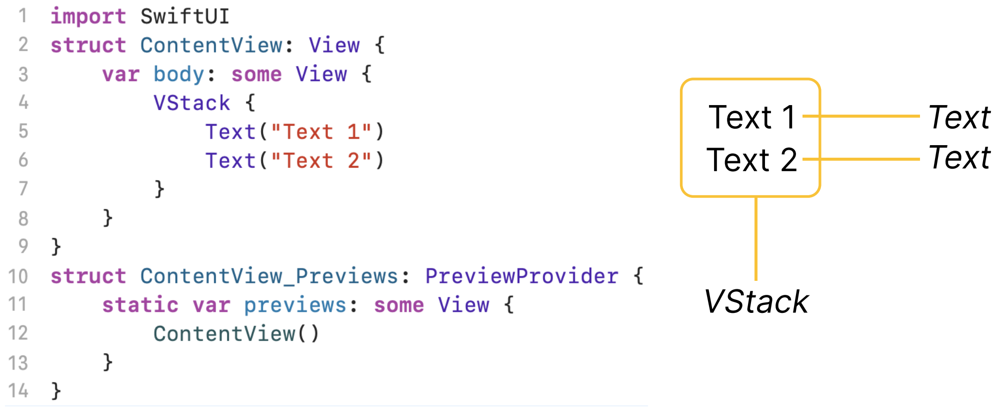
Entry Point

To develop a project with SwiftUI or UIKit the Integrated Development Environment (IDE) Xcode [35] is used. When a new SwiftUI project is created two files will be created.

- One file is named like the project name and is the entry point to the app (like the `main` method in other programming languages).
- The other file is named `ContentView` and is the first view the user is presented with.

Views

The user interface of an app is made up of one or more views. A view can either fill the entire screen or just be a dot on the screen. One view can be formed by one or more views. Figure 2.1 shows an example view with a preview of what it would look like on the right. In SwiftUI every element that is visible on screen is a `struct` that conforms to the `View` protocol [34] (line 2 in 2.1). This protocol forces the developer to implement a property called `body`[13] (line 3 in 2.1). This property is a so-called view builder and returns the opaque type `some view`. This means that the code inside the curly brackets of this property needs to return some kind of view. This can be a text view, an image view, a button view or multiple views combined as long as they together form a single



```
 1  import SwiftUI
 2  struct ContentView: View {
 3      var body: some View {
 4          VStack {
 5              Text("Text 1")
 6              Text("Text 2")
 7          }
 8      }
 9  }
10  struct ContentView_Previews: PreviewProvider {
11      static var previews: some View {
12          ContentView()
13      }
14  }
```

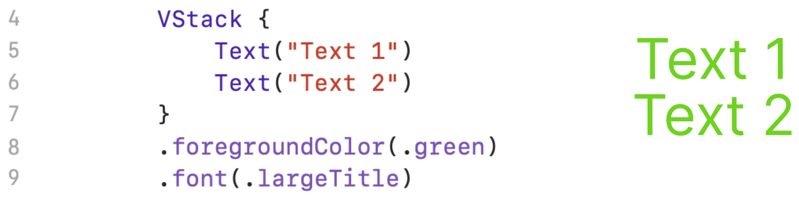
Figure 2.1: A view called `ContentView` that is made up of two vertically aligned `Text` elements. The yellow lines show the SwiftUI elements.

view. SwiftUI offers prebuilt components like text (line 5, 6 in 2.1), images, buttons, etc. To combine multiple view components to form a single view a `VStack` (line 4 in 2.1), `HStack` or `ZStack`, which correspond to vertical stack, horizontal stack and z-stack, are used. The children elements of these stacks are aligned just like their names suggest either in a vertical line, horizontal line or placed on top of each other. These stacks can also be placed inside each other.

Lines 10-14 in 2.1 serve as a preview of the view created by the code on the left side, which is displayed in the IDE Xcode during development. This is a very convenient feature of SwiftUI.

Styling

View modifiers (lines 8, 9 in 2.2) are used to style view components. They can be used to set the background color, the foreground color (lines 8 in 2.2), padding, corner radius, etc. View modifiers set the corresponding property for the view they are attached to and its children. In the example 2.2, both texts would be colored green and have the text size



```
4          VStack {
5              Text("Text 1")
6              Text("Text 2")
7          }
8          .foregroundColor(.green)
9          .font(.largeTitle)
```

Figure 2.2: A vertical stack containing two colored text elements with a preview on the right.

of `largeTitle` as the modifiers are placed on the `VStack` that encapsulates the two text elements.

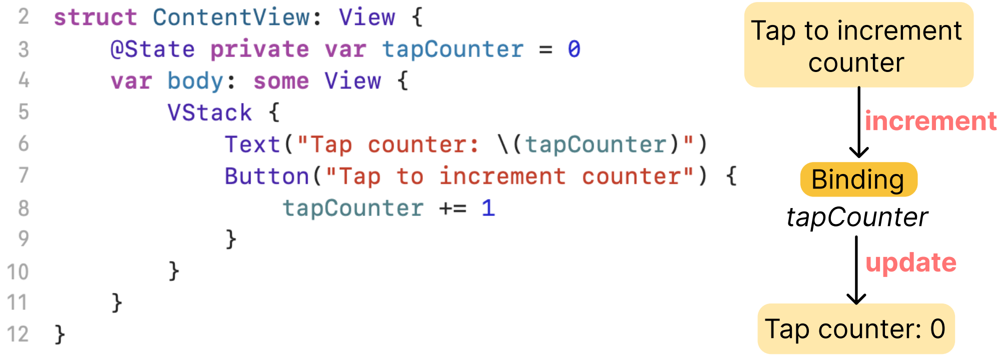
Listening to Events

To listen to events like a tap-gesture, a swipe action, the event that a view appears or disappears, etc., input and event modifiers are used. Just like view modifiers they are attached to a certain view component. For a comprehensive list, refer to the Apple documentation [18].

State Management with Property Wrappers

When building user interfaces it is often necessary to save a state, for example, the number of times a button was tapped. Furthermore, a view component that displays such a value or depends on it, should be updated once the value is updated. In SwiftUI property wrappers offer exactly that functionality. Property Wrappers are written before the variable with a `@` sign.
For a complete list of all property wrappers, refer to this page [39].

@State



```
 2  struct ContentView: View {
 3      @State private var tapCounter = 0
 4      var body: some View {
 5          VStack {
 6              Text("Tap counter: \(tapCounter)")
 7              Button("Tap to increment counter") {
 8                  tapCounter += 1
 9              }
10          }
11      }
12  }
```

Figure 2.3: An example for using the property wrapper @State, including a diagram.

To read and write a value inside a `struct` and to update the view components that depend on this value, the property wrapper `@State` [26] (line 3 in 2.3) is used. This also applies to the scenario where a `@State` value is passed to a child view. The child view will also be updated accordingly. Modifying a value inside a `struct` would normally not be possible as structs are value types, but with the @State annotation this value is no longer stored in the struct but in the managed storage of SwiftUI. In the example 2.3 the text is updated every time the button is tapped.

@Binding

However, if the goal is to move the Text element that displays the tapCounter to a custom view that is called from ContentView, the property wrapper @Binding is necessary. ChildView only contains the Text element from ContentView and the code from listing 2.1.

```swift
@Binding var value: Int
```

Listing 2.1: An example for declaring a binding

Passing a state value via parameter to a binding requires a dollar sign 2.2.

```swift
ChildView(value: $tapCounter)
```

Listing 2.2: An example for passing a binding as a parameter

Like before, the text is updated once the button is tapped.

@EnvironmentObject

To share an object throughout the app, it can be passed into the app environment. If an object is accessible in the environment it does not have to be passed through any layers of hierarchy [19], so it does not have to be passed as a parameter from one child view to the next. With the modifier environmentObject, an object is placed in the environment of all descendant views of the view to which the modifier is applied to [19]. In the following example 2.3, the object book is available to every view inside ExampleView.

```swift
@StateObject var book = Book()
var body: some View {
    ExampleView()
        .environmentObject(book)
}
```

Listing 2.3: An example for using @StateObject and environmentObject

To access an environment object inside a child view, use the property wrapper `@EnvironmentObject` (listing 2.4).

```swift
struct ChildView: View {
    @EnvironmentObject var book: Book
}
```

Listing 2.4: An example for using `@EnvironmentObject`

Navigation between Views

An app usually contains multiple views that fill the entire screen. To access the different views there has to be some kind of navigation between them. In SwiftUI there are two types of navigation techniques. Either stack navigation or modal navigation.
SwiftUI offers the `struct NavigationView` [22] for a stack-based navigation where the `struct NavigationLink` is used to trigger a new view to be slid on top of the original view with a slide animation from right to left. In the upper left corner, a button to navigate back is placed by default.
Example 2.5: Navigate to the view `NoteEditor` when tapping on a list row.

```swift
NavigationView {
    List(model.notes) { note in
        NavigationLink(note.title, destination: NoteEditor(id: note.id))
    }
    Text("Select a Note")
}
```

Listing 2.5: An example for using a `NavigationView` [22]

Modal navigation is achieved by using either the method `sheet` [25] or `fullScreenCover` [17]. The view to navigate to is presented by a slide animation from bottom to top. Both methods require a boolean state value which indicates whether to show the modal or not.
Example 2.6: Show `TargetView` when the button is tapped.

```swift
@State private var showSheet = false
var body: some View {
    Button("Show modal view") {
        showSheet.toggle()
    }
    .sheet(isPresented: $showSheet) {
        TargetView()
    }
}
```

Listing 2.6: An example for using modal navigation

Differences between UIKit and SwiftUI

Category	UIKit	SwiftUI
programming paradigm	imperative	declarative
API coverage	complete	limited
iOS support	all versions	iOS 13 and later
live-preview	no	yes
architecture	MVC	MVVM

Table 2.1: Differences between UIKit and SwiftUI

The main difference between the two frameworks is that SwiftUI uses the declarative programming paradigm while UIKit uses imperative programming. Imperative programming means coding step by step what and how it should be done, whereas in declarative programming the code describes the desired result, but not necessarily the exact steps that lead to that result. This makes building user interfaces oftentimes a lot faster and requires less code.

As SwiftUI is a rather new framework it only supports the iOS version 13 and later versions. So if the goal of an app is to support very old iOS versions, UIKit might be the better choice.

During development Xcode offers a live preview of the SwiftUI view that is currently created, without the need to start a simulator (a simulator is however still available). The most significant drawback of SwiftUI is its limited API coverage. This is an issue that will get smaller and smaller over time as more APIs will be supported with SwiftUI. This doesn't mean that they can't be used. It is just a little more complicated to do so.

Interacting with UIKit

As SwiftUI is not yet as established as UIKit, there are certain features that are not yet available in SwiftUI. However, it is still possible to use them with the help of the protocols

UIViewRepresentable [31] and UIViewControllerRepresentable [30].
Instead of building a struct that conforms to the View-protocol like with a normal SwiftUI view, a struct that conforms to one of the protocols from above, is built.
These protocols require the developer to implement the methods makeView and updateView. Inside the method makeView all code is written to create the specific UIKit view. The method updateView can be left empty or used if there is a value from another SwiftUI view, that is needed in the UIKit view. The method updateView is then called every time the binding value inside the UIKit-struct is updated.
If the goal is to communicate changes from the UIKit-view to other SwiftUI-views, a class called Coordinator needs to be implemented.

2.2 Metal

Metal [21] is an API from Apple for GPU (Graphics Processing Unit)-accelerated 3D graphics. It is designed just for Apple products, but therefore highly optimized with low overhead. In this project, Metal is used to render the 3D point cloud. Figure 2.4 shows the most important classes, protocols and methods, including their dependencies and relationships, that are necessary to render something on the screen with Metal. They are sorted from top to bottom in the order they would be called in.
The diagram is divided into two parts. What the developer needs to define once and what is newly defined for every screen refresh.
The following description is based on the sources [33], [6] and [1].
A MTLDevice is a reference to the GPU, so the part that executes the rendering with Metal. A CAMetalLayer is used to create a texture on the screen where later a triangle or dot will be displayed on. In Metal every custom view that is built, is made up of Metal primitives like for example a triangle or dot. The vertices of those primitives, so their location on screen and their color, are stored in an MTLBuffer. Next, the function VertexShader is called for every vertex in the MTLBuffer and is reponsible for further processing. Afterwards, the function FragmentShader is called for every pixel and determines its final color. These two functions are called as part of a pipeline that is made up of a MTLRenderPipelineDescriptor which defines which VertexShader, which FragmentShader and what pixel format should be used for this specific pipeline. Finally, this is bundled in a MTLRenderPipelineState which is used to reference and use this pipeline. Next, a CommandQueue holds all the commands that the GPU should execute next which are made up of MTLCommandBuffers each. An MTLRenderPassDescriptor defines the clear color, so the color shown on screen before anything else is rendered on

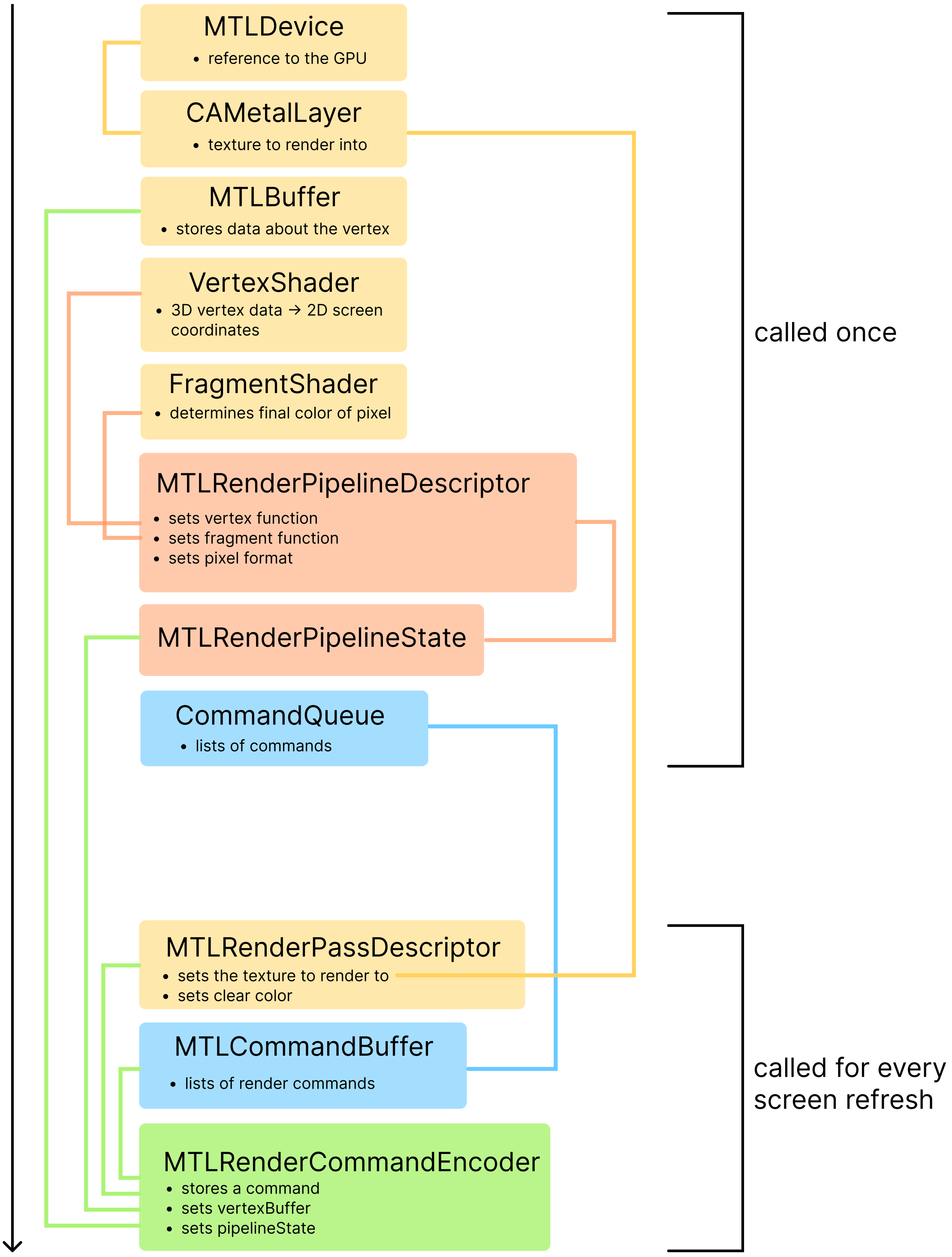


Figure 2.4: Metal rendering process based on [33], [6], [1]

screen, and where for example the triangle should be drawn onto (`CAMetalLayer`). Finally, a `MTLRenderCommandEncoder` is used to encode the earlier defined configurations into the `MTLCommandBuffer`, which is lastly executed by the GPU and the result is presented on the screen.

2.3 Functionality of the TrueDepth camera system

The TrueDepth camera system was introduced with the iPhone X and is located at the front of the iPhone inside the black area called "notch". For daily use, it is used for a feature called FaceID, which is a biometric authentication that allows users to unlock their iPhone or to use Apple Pay. In this project, the TrueDepth camera system serves as a means to record RGBD (Red Green Blue Depth) data. The depth data allows us to detect depth differences on the head of an infant. The TrueDepth camera system is made up of different components (see 2.5).

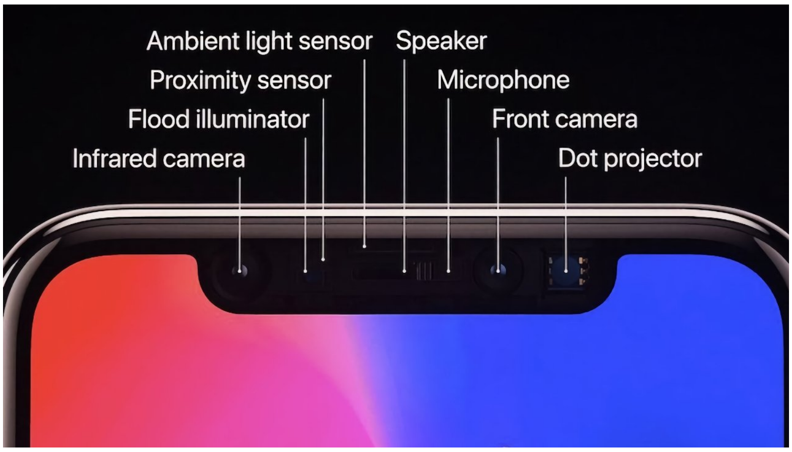


Figure 2.5: The TrueDepth camera system of the iPhone X [2]

How it works

The components of the TrueDepth camera system to measure depth data use infrared light, which works well with both low-lit and bright-lit conditions. Another advantage is that a human cannot see infrared light so the light does not blind the user during use. First, the flood illuminator creates infrared light to illuminate the entire face and the dot projector creates about 30000 infrared dots which are projected onto the surface of the face of the user [37].

Next, the infrared camera can detect infrared light and can capture both a heat image of the face as well as create a depth map from the infrared dots [37]. The information that the TrueDepth camera system provides, allows us to determine the distance of a pixel from the front-facing camera. The minimum distance from the object should be more than 15cm for the sensor to retrieve valid data [5].

Why LiDAR is not used

The TrueDepth sensor's minimal working distance and its accuracy are in the millimetre range which makes it ideal for this project's use case as the person performing the scan is very close to the infant. Although LiDAR also works with infrared light, it works best for longer distances and it places infrared dots further apart than the TrueDepth system does, which makes it less precise [42]. The TrueDepth camera system is better suited for this particular use case and is available on every iPhone, starting with the iPhone X, whereas LiDAR is only available on the Pro-series of the iPhones, starting with the iPhone 12 Pro.

2.4 Inclusion of OpenCV in an iOS Xcode Project

OpenCV (Open Source Computer Vision Library) [38] is an open source computer vision and machine learning software library with more than 18 million downloads. It provides all kinds of algorithms for computer vision use cases and can be used in C++, Python, Java and MATLAB, however, it is written natively in C++ [38]. OpenCV is used in this application to detect so-called ArUco markers 2.6 on the cap that the infant is wearing during the scan (see 1.1).

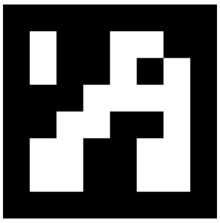


Figure 2.6: An example of an ArUco marker [1]

These instructions are expanded from the official OpenCV installation manual [3] and explain how to install OpenCV and use it in an Xcode project.

[1] https://docs.opencv.org/4.x/marker23.png

1. If the Xcode command line tools are not already installed on your mac, install them by executing the following command in an open terminal:

```
xcode-select --install
```

2. Create a folder on your mac with a name and location of your choice.

3. Inside the created folder, open a terminal and run the following two commands:

```
git clone https://github.com/opencv/opencv.git
git clone https://github.com/opencv/opencv_contrib.git
```

4. Inside the same folder created in step 2, run the following command. This can take a long time. `Python` as well as `cmake` have to be installed for this step.

```
python opencv/platforms/ios/build_framework.py ios --contrib
    opencv_contrib
```

5. Confirm the success of the previous step by checking if there is a folder called "opencv2.framework" inside the "ios" folder inside the folder created in step 2.

6. Clone or download the InferMod3D project from Gitlab [40] and open it in Xcode.

7. Drag the folder "opencv2.framework" into the left column of Xcode, where all the source files are listed.

8. In the pop-up that is shown, select "Copy items if needed", "Add to target" and click on "copy".

9. In Xcode navigate to app settings and select the tab "Build Phases".

10. Expand the section "Link Binaries With Libraries" and make sure that "opencv2-.framework" is listed in this section.

3 Setup and Concept

3.1 Prerequisites

The version of the app that is developed in this thesis, has the following prerequisites:

- An iPhone X or newer with a working TrueDepth camera system.
- At least three gigabytes of free storage, as one scan can be as large as three gigabytes.
- A cap with ArUco markers on it (see figure 1.1).
- An infant or doll that wears the described cap.

3.2 Preconditions

At the start of this bachelor thesis, the ISRG already provided a technical prototype of the app. This section describes the functionality of this prototype. For a visual impression of how the prototype looks, see figure 3.1. The prototype is based on a tutorial app from Apple on how to display RGBD data as a point cloud [27].

Main screen

The screen shown in this screenshot is the only screen of the prototype. This screen displays the current recording as a point cloud or the point cloud from a previous recording.

Why the infant needs to wear a cap with markers

For the scanning procedure the infant has to wear a special cap (figure 3.1) with so-called ArUco markers (figure 2.6) on it. Each ArUco marker serves as a reference point to be able to construct a 3D model from all the recorded frames. For this construction it is necessary to know where each frame is positioned in 3D space in relation to another frame. The markers, i.e. reference points, are the means to achieve this. If the user only

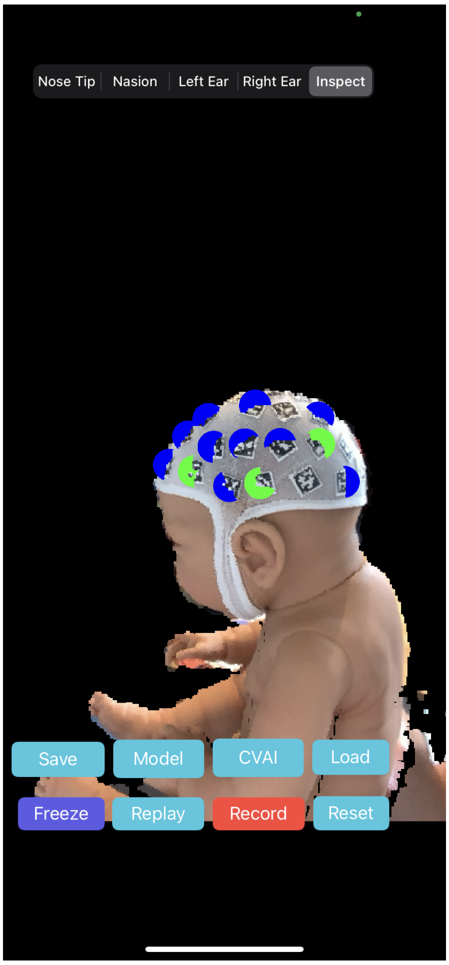


Figure 3.1: Screenshot of the initial technical prototype

records video footage from each side of the head, with no overlapping footage, there are almost no reference points where to connect the frames.
The ArUco markers that are detected on the cap, are colored in either red, blue or green (see 3.1). At least every frame, when there is new data from the camera and sensor, these markers are extracted if possible. Each color symbolizes a different overlap state:

- red: marker is in the current frame, but there is no connection to an older frame
- blue: marker is not in the current frame, but it was in an older frame
- green: marker seen in current and older frame

Why the user needs to set annotations

The objective of recording RGBD data of the head is to create a 3D model of the head, which provides data to calculate the CVAI. To combine all the recorded frames, reference points in the frames are necessary. The ArUco markers on the cap are such reference points (see previous section). However, there are no such markers on the face or on the ears, even though these body parts are necessary to calculate the CVAI. With adequate computer vision algorithms, it would be possible to detect these body parts from the head. The problem is that there is not enough image data of the heads of infants. This is why the user has to mark the nose tip, the nasion, the left ear and the right ear in the scan replay by placing a dot (also called an annotation) to this body part. The goal is to collect enough head data to train an AI than can detect these body parts in the future.

Overview of the framework and most important classes and structs

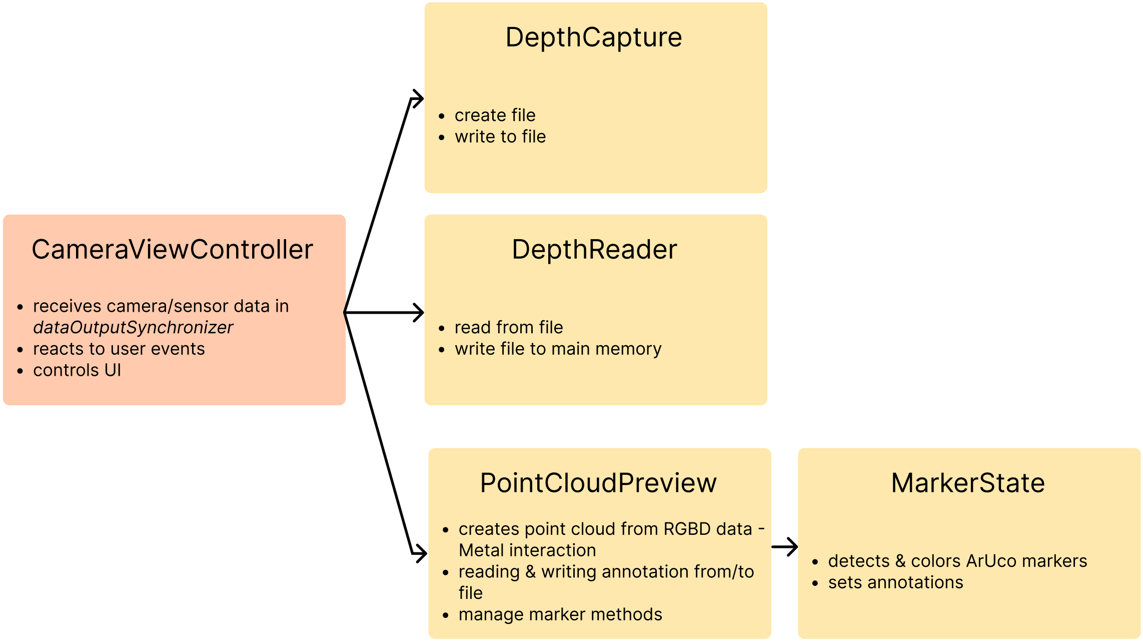

DepthCapture
• create file
• write to file
CameraViewController
• receives camera/sensor data in *dataOutputSynchronizer*
• reacts to user events
• controls UI
DepthReader
• read from file
• write file to main memory
PointCloudPreview
• creates point cloud from RGBD data - Metal interaction
• reading & writing annotation from/to file
• manage marker methods
MarkerState
• detects & colors ArUco markers
• sets annotations

Figure 3.2: Most important files in the technical prototype

The prototype is built with the framework UIKit [29] from Apple. UIKit projects can either be written in the programming language Swift or Objective-C. This prototype uses both languages although Swift is used for interacting with the user interface of the app.
In a UIKit project, every screen of the app has a class that manages this screen. This class

inherits from the UIKit class `UIViewController`, which provides all the functionality that is necessary for dealing with the user interface. Figure 3.2 shows an overview of the most important files in the prototype. As the prototype only has a single screen the class `CameraViewController` manages the entire logic and user interface. When, for example, a button is clicked, the click event is received by a function within `CameraViewController` and then the necessary calculations or function calls are made.
Apart from being responsible for the interaction with the user interface, the class `CameraViewController` is also responsible for handling the camera and sensor data. The framework AVFoundation [9] from Apple is used for that purpose. It allows a developer to interact with audiovisual media on Apple products. To use this functionality the class `CameraViewController` inherits from the protocol `AVCaptureDataOutputSynchronizerDelegate`, which forces the developer to implement the method `dataOutputSynchronizer`. When the app uses the TrueDepth camera system this method is constantly called and provides both the video and the depth data.

Functionality of the buttons

To understand what functionality the technical prototype offers and how it works, the following sections will go through each button present on the screen (3.1). The buttons are mentioned in the order they would be used when creating a scan.
When the app is launched and no button is pressed the live camera feed is displayed on the screen. However, it is displayed in the form of a point cloud.

Record-button

The "Record"-button starts and stops a recording. The class `DepthCapture` is responsible for saving the currently recorded depth and video data to a file (see 3.2). Starting a recording creates a new file in the document directory of the iPhone. The filename is formed by the current date. During the recording, a method from `DepthCapture` is called, every time the `dataOutputSynchronizer` method is called. The RGBD Data that is received through the `dataOutputSynchronizer` method is directly passed to `DepthCapture` where it is written into the previously created file. The passed data is in the form of a pixel buffer.

Replay-button

The "Replay"-button replays a recorded scan. Reading the scan data from a file is handled by the class `DepthReader`. The scan file needs to have the same name as written in the code, otherwise, the app crashes, i.e. the filename is not dynamic. Provided that this step succeeds, the screen shows the point cloud exactly as during recording. Just like during the recording part, every time the `dataOutputSynchronizer` method is called, a method from `DepthReader` is called that is responsible for reading from the file. Again the RGBD data is passed to `DepthReader`. First, a pointer to the base address of the passed pixel buffer is determined, then a certain number of bytes are read from the file and lastly this data is copied to the determined address. The number of bytes that are read from the file correspond to the number of bytes of the passed pixel buffer which is the reason why the camera is still needed for replaying a recorded scan.

Model-button

The "Model"-button allows a user to switch between the normal point cloud view with markers and a view where only the detected ArUco markers and annotation dots are shown.

Freeze-button

The "Freeze"-button pauses the replay and allows the user to set annotations to a specific part of the head.

Annotation-selector

To create a complete 3D model of the scanned head and to calculate the CVAI-index, the user has to identify the nose tip, nasion, left ear and right ear. To set an annotation to a part of the head, this part first has to be selected from the selector on the top of the screen (see 3.1). When a head-part is selected, the user can drag over the screen and a colored dot (the annotation) will appear that can be placed at a certain position. If the current location of the head is not ideal a user can also rotate and zoom the displayed point cloud by selecting "Inspect" at the top of the screen and dragging over the screen. A double-tap resets the view's position and orientation. The logic for this functionality is contained in `PointCloudMetalView` and called from `CameraViewController`.

Save-button

After the user sets the annotations to the right ear, left ear, nasion and nose tip they can be saved into a separate text file, named "AnnotationMarkerCoordinates.txt" by pressing the "Save"-button. Unlike the scan saving, this saving part is handled by a method in `PointCloudMetalView` and the filename is not dynamic. Apart from the annotations also the markers are saved in this file.

Load-button

If annotations were already set and saved in a previous replay they can be loaded and shown in the model with the "Load"-button. If these annotations don't exist, the app crashes. This part is also handled by a method in `PointCloudMetalView`.

CVAI-button

Once a replay is fully played, all annotations are set and the device is newer or equal than the iPhone 11 the "CVAI"-button calculates the Cranial Vault Asymmetry Index. The result is shown in the console output in Xcode. For a reliable result, the replay has to be replayed at least one time so that all markers can be detected and all the annotations set. The reason why this only works on an iPhone 11 or newer is that a technique called vertex amplification is used for calculation. This is a feature from the framework Metal. This technique however requires at least the A13 Bionic processor [20] which was introduced with the iPhone 11. On iPhone X, XR and XS which already have FaceID, but an older processor, the app will crash on launch when failing the check if vertex amplification is supported. For the devices where this calculation is supported, 36 images are saved into separate files that show the outline of the head at a certain layer. During calculation the head is divided into layers from top to bottom and the goal is to find the layer with the longest contour.
The ISRG also published a separate paper [43] on how the 3D head reconstruction and CVAI calculation work in detail.

Reset-button

Finally the "Reset"-button resets the point cloud model.

Further functionality

When a user installs the app the first time, it will ask for permission to use the camera. This is required by iOS. If a user declines an alert is shown where the user can navigate to the app settings to enable it. Furthermore, there is an alert when the TrueDepth sensor gets too warm.

3.3 User Experience Concept

The purpose of this app is to create scans of the heads of infants with the help of the TrueDepth camera system of the iPhone.
The camera system is located at the front of the iPhone at the top of the screen (as explained in section 2.3). This creates a challenge for the person creating the scan because the camera has to point to the infant which means it points away from the operator, which finally means that the person that is recording cannot see the screen accurately. However, the user still has to make sure that the head of the infant is not too close, not too far away and not just partially in the camera's view.
The next challenge for the user is to somehow know when the entire head has been recorded so that the recording can be safely stopped. After all, the goal of this version of the app is to collect high quality data to train a model. When it is time to export the data from the phone to the server for training, it would be very unfortunate if some of the data can't be used because it is incomplete. That is why there has to be feedback to the user regarding the completeness of the scan.
This chapter will present the steps that were taken to accomplish the final feedback concept that is implemented in the app.

What feedback output mechanisms does the iPhone offer?

The first step is to find out all the features that an iPhone offers that could be used to provide feedback to the user.

Screen

The most obvious solution for giving feedback on a phone is the screen, as it offers a wide variety of visual feedback possibilities. But as described in the previous section, the screen is not always visible to the person recording as the phone faces in the other direction. Therefore the screen can very well be used for any feedback before starting a recording and after stopping a recording, but it is not ideal for feedback during a recording.

Audio

Another option is to use audio feedback. Audio output has the advantage that it doesn't matter which way the phone is turned, the audio just has to be loud enough for the user to hear it. Audio allows us to use simple sounds like in a parking sensor or use actual recordings of words or even sentences. Words or sentences have the advantage of being very descriptive. One thing to keep in mind is that the user of this app is dealing with infants who might react very differently to audio than the user would. A loud beeping sound like in a parking sensor could frighten the infant which can't be the goal of this app. One solution could be to wear headphones while recording so that the infant does not hear the sounds.

Haptic

A further non-visual feedback mechanism is haptic feedback, i.e. vibrations. Vibrations are normally used to, for example, indicate that a new message arrived, that a timer has finished counting or that somebody is calling. Vibrations can be customized regarding their intensity, their sharpness and their duration. Just like with audio feedback there is also the advantage that the phone has certain default vibrations for success and error that the user is already familiar with. This could make the usage of the app easier when a user can already associate certain vibrations with certain meanings.

Light

A final option is the flashlight of the phone. It is conveniently placed on the back of the iPhone which is visible to the user at all times during the recording. It does not offer as much variety as audio or haptic feedback, but it can still be controlled regarding its brightness and how long it is turned on. However, like audio feedback, it might scare the infant and it has to be examined whether the brightness is bearable for the user or if it is distracting.

Which events require feedback?

Before choosing a feedback output mechanism, the events that require them, have to be determined first.

Positioning

The first event is the positioning of the iPhone during a scan. As the iPhone is faced away from the user during the recording, it is hard to see the rendered point cloud on screen and therefore the user can often only guess if the iPhone is held in the correct position, meaning that the head of the infant is fully within the camera's view.
The position can be off in a few different ways:
The iPhone is held too far away from the head, too close, too far to the right, too far to the left, too far to the top or too far to the bottom.
The positioning also affects how much the video frames are overlapping. As only overlapping frames can be used for the construction of the 3D model, it is important that every part of the head is recorded with enough overlap (as explained in section 3.2).

Progress

The second event that requires feedback is the progress of the scan, as already insinuated in the introduction of this chapter. The user has to be informed when a scan is complete and when the recording can be stopped. But what about the time before that? It might also be a good idea to give some feedback when the user has reached the middle of the scan, to give him/her an idea of the progress so far. However, giving progress feedback for every ten percent progress is probably too much and there is a high chance that the progress feedback might interfere with the positioning feedback and just confuse the user.

What data is available to create feedback from?

Now the question arises what data does the app process that can be used as a criteria to trigger feedback. So what data is processed in the app that can be used to make a statement about the positioning of the iPhone and the progress of the scan?

Markers on the cap

The first possible data source is the markers on the cap. During the scan, the infant has to wear a cap with ArUco markers (2.6) on it.
Each marker has a unique ID that is always the same, meaning it is not an ID that the software assigns the marker dynamically, but an ID that can be read from the look of the marker itself. The app already has the functionality to detect these markers and extract their ID.

On top of that, the app already determines the amount of overlap between the video frames using the ArUco markers. The technical prototype already colors the markers in the live point cloud video feed according to their overlap status (as shown in figure 3.1).

Depth and video data

Another data source is depth and video data. Detecting the position of a head of an infant from the video is easy for a human being, but for an app to do it, it needs computer vision algorithms for head detection. As such algorithms are not implemented in the app, this option is discarded for now, because it should be examined first if there already is functionality in the app that can be used as a data source for either the positioning or progress feedback.
The depth data could potentially serve as a means to detect the distance of the iPhone from the head.

How ArUco markers are used as a feedback data source

The option that seems most suitable is to use the ArUco markers as a data source, because feedback can be provided both about the positioning and the progress.
First, the total number of ArUco markers on the cap is determined. During the recording, the number of detected marker IDs is counted and in combination with the total marker count, the progress of the scan can be calculated. When counting the markers there can't be any duplicates, but with the unique ID of each marker, this isn't a problem.
At least once per frame, the overlap of the markers is calculated. A 3D model can only be successfully created if the frames that are used to build it, have enough overlap. When the iPhone is positioned poorly, the overlap is insufficient, because fewer or no markers are detected. When the overlap is sufficient, the iPhone is also positioned well. So the overlap status is used as a indicator for how good or bad the iPhone is positioned.

Combining data with a feedback mechanism

We now established what data serves as a source for the positioning and progress feedback. In the following, different output mechanisms to give feedback about the position are evaluated.

Vibrations to indicate bad positioning

The first attempt is to use vibrations to indicate that there is not enough overlap between the frames, i.e. the positioning is bad. For this kind of feedback, the system error vibration pattern (3.3) is used. If the user already has an iPhone he/she is probably familiar with this vibration pattern and associates some kind of error with it. For each frame the overlap status is determined and when there is too little overlap a vibration is initiated.

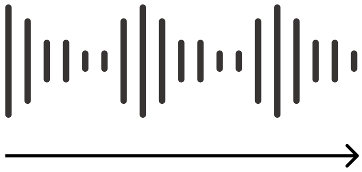


Figure 3.3: iOS system error vibration pattern

Setting a threshold for a feedback event

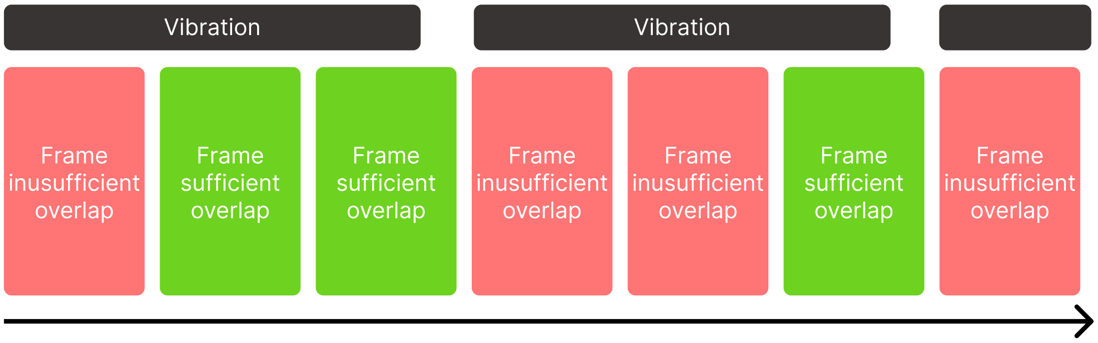

Vibration
Vibration
Frame inusufficient overlap
Frame sufficient overlap
Frame sufficient overlap
Frame inusufficient overlap
Frame inusufficient overlap
Frame sufficient overlap
Frame inusufficient overlap

Figure 3.4: First feedback mechanism attempt

However, while testing, this approach didn't seem very effective anymore. As the overlap is determined per frame and there are dozens of frames per second, the overlap value is updated a lot. Oftentimes there was too little overlap every other frame and as programmed for each insufficient overlap, a vibration is played (see 3.4). So a vibration would start, but then in the next frame the overlap was sufficient again, however, the vibration would still be in progress and then a few frames later the overlap was insufficient again, which again triggered a vibration. In total it sometimes seemed like the phone is vibrating all the time. Therefore we can't use the value of the overlap variable in every frame. There has to be some kind of threshold and only if this threshold is reached a vibration is triggered. So the idea is to only update the overlap status if the overlap value has been the same for a certain number of frames in a row. The question is where should the threshold be set at? Heuristically this value is set at ten, which in practice proved to be acceptable.

Creating a custom vibration

Although the update problem is solved, the vibration pattern also didn't seem ideal. It was a very light vibration and not that noticeable (see 3.3), especially when taking into consideration that the infant might make noises during the scan and therefore distract the physical therapist. The next step is to create a custom vibration that has the right intensity and sharpness. To determine an ideal combination of intensity and sharpness a tutorial haptics app from Apple was used [32].

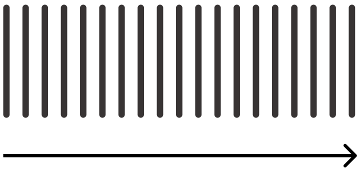


Figure 3.5: custom error vibration pattern

The new custom vibration 3.5 is continuous and doesn't change in intensity while being played. So now a continuous vibration is played when there is insufficient overlap for ten frames in a row. However, it is still not ideal.
Consider the following scenario: The insufficient overlap threshold is reached, a vibration is triggered and then the vibration has finished playing. How does the user now know when there is enough overlap again, so he/she can stop adjusting the iPhone's position? When there is no new vibration, correct? True, but what if there is just no new vibration because the overlap status stays false because for a certain amount of time, there are not enough frames in a row to flip it to true. This means that the user thinks the phone is positioned correctly, because there is no new vibration since the last one, but in reality, the overlap status just stayed insufficient and that is why there is no new vibration. To conclude, we need some kind of confirmation when the iPhone is back in an ideal position.

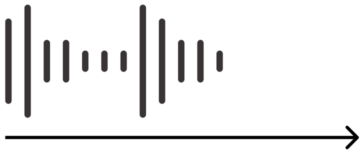


Figure 3.6: iOS system success vibration pattern

Adding a second vibration

For the confirmation, the system's success vibration 3.6 is used. This vibration is less intense than the custom vibration that was created for indicating a not ideal position. This makes it easy for the user to distinguish the two vibrations. In the scenario from above, a user would know that if there is no success vibration after the error vibration, he/she is still not in an ideal position and can keep adjusting it, until the success vibration occurs.

Feedback with the flashlight

Next, the possibility to use the flashlight on the iPhone is examined. To replicate the method we used before with vibrations, we need two different patterns with one symbolizing an error and the other success. As we don't want to hurt the eyes of the user the brightness needs to be at the lowest setting. However, this only leaves us with the possibility to turn it on or off, so replicating the vibration pattern with the flashlight might be possible, but hard to remember for the user. A more effective way seems to be to turn the light on, when the overlap is insufficient and turn it off when the overlap is sufficient again. Having the flashlight turned on when there is too little overlap turned out to be very effective and easier for the user. The reason is due to a specific scenario that occurred while using vibrations. A vibration would be triggered, so the threshold counter would be reset. Then the threshold is met again and there is another error vibration without having a success vibration after the first error vibration. This was somewhat confusing and the option to have the flashlight turned on the entire time seemed more effective. Yet the flashlight is very bright even at its lowest setting and therefore not a viable feedback option.

Feedback with audio

One feedback mechanism that has not been tested yet is audio. As audio can be very descriptive and easy to use, the feedback logic that was used in the first attempt with the vibrations is used again. Meaning that there is some audio output when the overlap is insufficient and another audio output when the overlap is sufficient again. A recording of the words „off“ and „in“ is made, symbolizing the user is out of position or back in position. Unlike with the vibrations, it is necessary that the iPhone is not in „do not disturb“-mode as the audio would usually not be played in this case. Apart from this minor inconvenience, audio proves to be a viable feedback mechanism.
To conclude, the option to use vibration is almost ideal, using the flashlight is not an option and audio is ideal.

Feedback about the scan progress

Now that all possible feedback mechanisms have been evaluated, they need to be applied to providing feedback about the scan progress as well. As already mentioned, progress feedback does not require that much feedback. Letting the user know when he/she has reached the middle of the scan and the end of the scan appears to be sufficient until properly tested. Using audio output for this kind of feedback seems to be the most effective way as it is the most descriptive option. It allows us to play a recording of the word "50" to indicate that the user has reached the 50 percent mark of the scan and the word "Done" to indicate the end of the scan. If we were to use vibrations for that kind of information the user would need to remember some kind of vibration pattern that represents "50" and another pattern that represents "Done", which is too much work for the user.

Choosing a feedback mechanism for the scan progress and the marker overlap

When using audio output for the progress feedback the question arises what feedback mechanism to use for giving feedback about the overlap? The result of the previous evaluation of the different feedback mechanisms concluded that audio is ideal and vibration almost ideal. Even though audio is ideal it has to be taken into account that the progress feedback is using audio as well, which could potentially mix with the overlap feedback. Maybe there is a way to make vibrations ideal as well. And there is.
In the previous evaluation, part of the disadvantage of the vibration feedback mechanism was that there were two kinds of vibration patterns, one for error and one for success.

When evaluating the flashlight option it was not suitable because of the brightness, but the pure logic for giving feedback via light concluded to be very effective and intuitive for the user. To recap, the light is turned on when the position is not ideal and turned off again when it is ideal. So why not use that logic for vibrations? In practice this means that the continuous vibration pattern, shown in figure 3.5 is played the entire time the iPhone is not in an ideal position and once the position is ideal, the vibration is stopped. As already mentioned this has the great advantage that the user does not have to focus as much on what kind of vibration pattern was emitted and if there is an error or success. Now there is just one kind of vibration pattern, which indicates an error and if the iPhone stops vibrating the user can be certain that the position is ideal.

Adding visual feedback for the scan progress

Regarding the feedback for the overall scan progress, there is one more aspect worth implementing. Imagine the following scenario: The user is performing a scan and hears the audio output "50", keeps on scanning and then the audio output "Done" is played. However at that moment, the infant is crying very loud and the user is not sure if this audio output was already played or not, so he/she doesn't know if the recording can be stopped. Wouldn't it be helpful in this case to show the progress on the screen? The user can't really see the screen most of the time, but in this case, the iPhone can be turned a little bit to allow the user to check the progress and to give him/her certainty about whether the scan is finished. To show the progress on the screen, a progress bar is implemented.

The positioning feedback is not enough

The current state of the feedback is:
Position feedback

- Start vibration, if the marker overlap is insufficient ten times in a row.
- Stop vibration, if the marker overlap is sufficient ten times in a row.

Progress feedback

- Play audio "50", when 50 percent of all marker IDs have been detected.
- Play audio "Done", when 100 percent of all marker IDs have been detected.
- Show the progress visually as a progress bar.

Although it is very helpful to know when the position is not ideal and needs to be adjusted it does not make it easier to find the correct positioning of the iPhone so that the position is ideal again. There are six potential directions the iPhone could be moved towards. Towards the head, away from the head, to the left, to the right, up and down. If the user only knows that the position has to be corrected, he/she can either guess which direction is the correct one by examining how the iPhone positioning looks in relation to the head, or he/she can try every possibility until the vibration stops.
In section 3.3 we stated that we only have the marker information, the depth-and video data.

Delivering more expressive positioning feedback - first idea

The first idea is to use the depth information of the TrueDepth camera system to detect the distance from the head and in return, give the user feedback to either move further away or close in. However, the following questions need to be answered for that:

- Which points of all recorded points belong to the head of the infant and which ones to the surrounding?
- Which point is the closest one to the camera?

So the great challenge is to figure out a point to retrieve the depth information from. Looping through all points and selecting the one with the smallest distance for every single frame, is performance-wise not possible as it would take way too long.

Delivering more expressive positioning feedback - using the ArUco markers

Even if looping through all points to find the closest one would work, it still would only be a guess, that the closest point is a point of the infant's head. However, using the markers on the cap erases this guess.
For each corner of an ArUco marker its x and y coordinates in the image, as well as its depth data are determined. Testing showed that when the sensor gets too close to an ArUco marker, the depth information of a corner returns `nan`, which corresponds to "not-a-number". This is the only scenario where this value is returned, which makes it ideal as a criteria to determine whether the user is too close to the head. This leaves five more directions to figure out.
Increasing the distance between the markers and the sensor only results in detecting fewer markers which is not a good indicator of being too far away, because this can also occur due to bad lighting or because only parts of the head are in the frame.

However, the x and y coordinates of the corners of the markers can be used to determine the other four directions (left, right, top, bottom). The goal is to find the center of the head or at least the center of the part that is visible in the recorded image (green cross in 3.7). By comparing this center position to the center of the image, its position in the image can be determined and appropriate feedback delivered to the user. Depending on the position of the head center the feedback can be "up", "down", "left", "right", "left-up", "left-down", "right-up", "right-down". If the head center lines up with the center of the image, the head is positioned ideally. To deliver this kind of feedback, vibration is not a viable option as it would require too many different feedback patterns to symbolize each scenario. This is why audio is used for this purpose. To indicate that the position is not ideal in general, including being too close and otherwise off position, vibration is still used. Audio is used as an addition to give positioning instructions when possible.

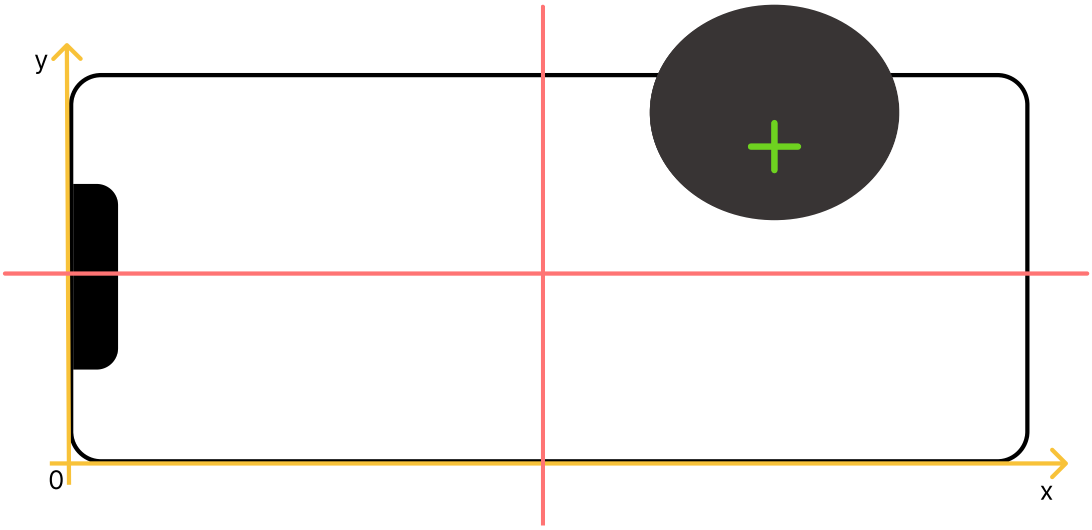


Figure 3.7: Determine 2D position of the head within the recorded image

When to deliver which positioning feedback - criteria

After developing a technique to determine instructions on how to position the iPhone, we again have to combine all the available data sources and define how to set the criteria for the positioning feedback.
Available information to potentially trigger positioning feedback:

- marker overlap (with or without threshold)

- number of detected marker IDs
- number of markers with invalid depth information
- 2D position of the head, if markers are detected

The marker overlap still is a very strong criteria as it is crucial for constructing a complete 3D model of the head and therefore remains a primary criteria.
There are three scenarios to distinguish between. The position is ideal, the position is too close to the head or the position is otherwise off.
In the ideal position, the marker overlap is sufficient and all markers have valid depth information. To distinguish between being too close to the head and being otherwise off position there has to be some threshold for the ratio of markers with invalid depth information and all detected markers. This ratio is set to about 30 percent. So the position is too close when more than 30 percent of all markers have invalid depth information. On the other hand, if the number of markers with invalid depth information is less than 30 percent of all markers and the overlap is insufficient, the position has to be otherwise off. This could be too far away or too far to a specific side. And in this scenario, the new positioning feedback with the directions the iPhone has to be moved to, comes into play. Like previously for the new criteria a threshold, as described in section 3.3, is used again.

Making the positioning feedback criteria less strict

Testing showed, that the criteria, described in the previous section, is too strict and trigger feedback even though the position seems ideal when looking at the display. Now the primary criteria that decides whether the position is off or ideal is the marker overlap. Only if the marker overlap is not enough, it is distinguished between being too close and otherwise off by looking at the ratio of invalid markers to all markers. On top of that, the criteria of whether the head is in the center of the image was loosened. First, a range of five percent next to the section of the x-axis and y-axis middle was introduced (red area in 3.8). Second, we no longer use the center of the head and compare it to the middle of the image, but the vertical and horizontal maximum values of the head (green lines in 3.8). In the scenario, shown in figure 3.8 the audio feedback would be "left-down". The figure portrays the device in landscape mode because this is the way the TrueDepth sensor delivers its data. However, the user will hold the device in portrait mode. The feedback is "left-down" and not "left-up", because the camera of the iPhone is attached at the top of the device. So moving the device down, moves the head in the picture up.

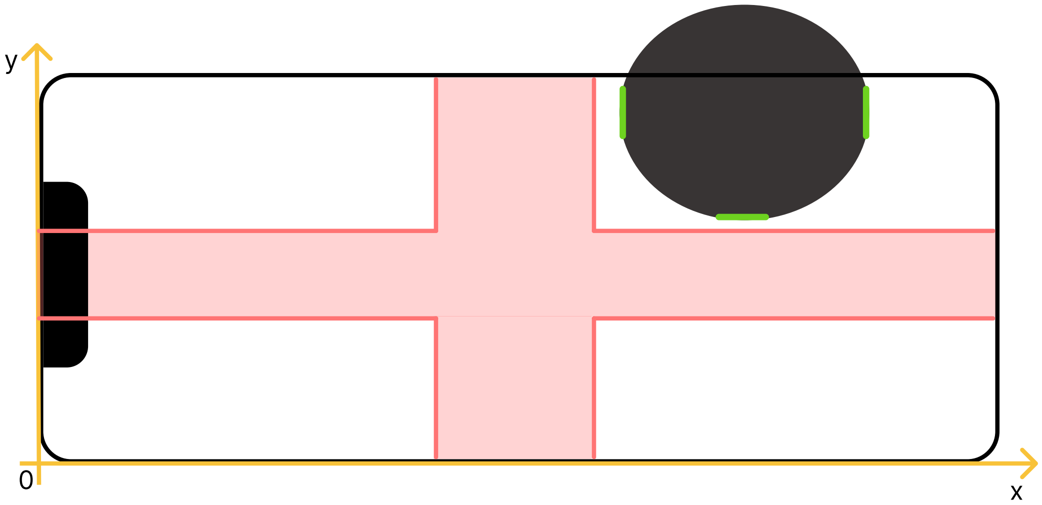


Figure 3.8: Determine 2D position of the head within the recorded image

Summary of the final feedback concept

Both the positioning of the iPhone and the progress of the scan are determined with information from the ArUco markers.

Position feedback

- Primary feedback: continuous vibration (see 3.5) while the marker overlap is insufficient for four frames in a row.
- Secondary feedback:
  - Audio output "too close" if more than 30 percent of all markers have invalid depth information.
  - Audible positioning instructions ("left-up", "right", ...) if less than 30 percent of all markers have invalid depth information.

Progress feedback

- Determined by the number of detected marker IDs. The user can select between:
  - Audio output every 25 percent.
  - Audio output every 50 percent.
  - Audio output at 100 percent.
- The progress is also shown visually as a progress bar, which uses the colors from a traffic light.

3.4 Usage Scenario

This section describes potential usage scenarios of the app and provides a walkthrough of the features in each situation.

Create a scan

A new infant with possible head deformation is brought in for a diagnosis. The physical therapist opens up the InferMod3D Recorder app, makes sure that there are more than three gigabytes of free storage, checks that the sensor's thermal state is normal and adjusts the audio volume according to the surroundings. Afterwards he/she opens the scanning screen. The infant is meanwhile held by a parent. The record button is tapped and the front facing camera of the iPhone is turned towards the infant. The physical therapist walks around the infant to record the entire head. During the recording the haptic and audible feedback helps him/her to position the iPhone correctly. The progress feedback via audio informs about the progress of the scan. Either by listening to the audio output or by looking at the progress bar on the screen, the therapist knows when to stop the recording.

Collect manual measurement data and rate the experience

After a recording is stopped the patient data input screen automatically appears. The physical therapist enters data like the diameter, diagonal length, etc. from a manual measurement of the head. This measurement was conducted either before the scan or can be conducted right now. Once complete, the data is saved and the user is automatically navigated to the screen for rating the scan. In this step the user rates the previous scan experience by using sliders and toggle switches. After saving this data, the user is automatically navigated back to the home screen of the app.

Replay and annotate the scan

Afterwards a list of all the recorded scans is opened. The last recorded scan is opened and starts being played. Now the job is to set an annotation to the right ear, left ear, nose tip and the nasion. Refer to section 3.2 for why this is necessary. When the replay is at a position where one of the named body parts is clearly visible, the physical therapist pauses the replay. From the buttons that appear in the blue tile at the bottom of the screen the corresponding body part is selected and with a drag gesture, an annotation dot is placed at the specific body part. This is repeated for all body parts. When paused, the 3D model can be turned and enlarged by either a pan-or zoom gesture to improve the view on a specific body part even more. That way setting an annotation becomes easier. To reset the view double tap on the screen. Previously annotated body parts can, of course, be edited again, if a more suitable position in the replay is presented. If the replay is over and a suitable position for annotating a body part was missed, the replay can be restarted.

Rename a file

If a scan file should be renamed after the name of the patient for example, the screen to manage all the files can be opened. From the list of all the files that the app saved, the desired one is selected by tapping the edit files button at the top and then selecting the row. The user clicks on rename and a new view appears where the new name can be entered and the rename executed.

Export and delete scans

Every once in a while, all the annotated recordings are exported to the server for training a model. The iPhone is plugged into a Mac and in the Finder the iPhone is selected and the tab "Files" is opened. Below the name of the InferMod3D Recorder app, all saved files appear and can be selected and dragged out of the finder into a new storage location. To delete the no longer needed files, they can either be deleted from within the Finder or within the file screen of the app.

Check how the feedback works

If the user is not sure anymore what kind of feedback the app offers and what it means, he/she can navigate to the instructions view, via the home view. Each feedback mechanism is described. By tapping on the feedback mechanism a demo vibration or demo audio output is also provided.

4 Implementation - Views and Logic

4.1 What all views have in common

Before each screen/view of the app is presented separately, there are some aspects that all views have in common to achieve a coherent design and user experience throughout the app.

The core principle of the entire app is to only present something to the user that he/she can and should use at a specific moment. For example, a button to edit an item from a list is only shown, once a list item is selected. This prevents the scenario where a user taps on a button only to find out that nothing happens or that an error occurred because he/she should have only used that button later in time.

All screenshots in this chapter are captured when the iPhone's appearance is set to light mode, but every view of the app also supports dark mode. That way, the user can keep using his/her preferred appearance mode even when using the app.

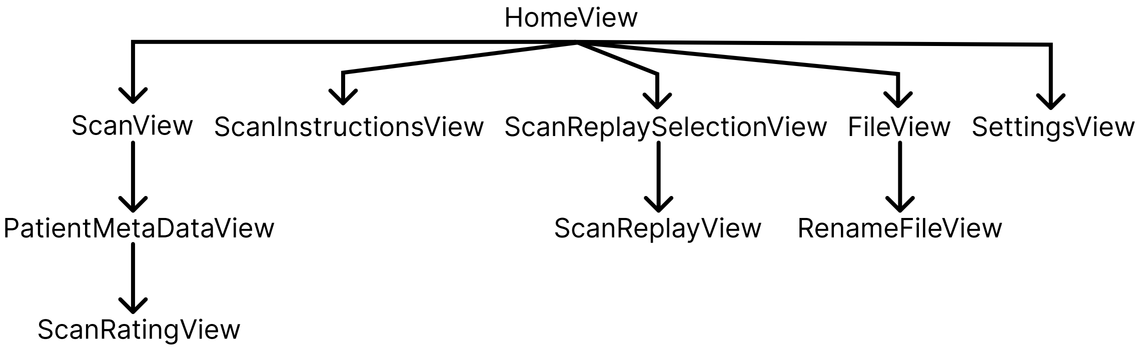


Figure 4.1: All main views of the app. The arrows show the navigation paths.

NavigationView

For navigation purposes two methods were introduced in section 2.1: Stack based navigation and modal navigation. This app always uses modal navigation for every view that is newly presented, so in figure 4.1 the ScanView is presented as a modal from the

HomeView.
However, every main view (see figure 4.1) is nested inside a `NavigationView` [22]. Although navigation is done via modal navigation, a `NavigationView` is used for design reasons. The SwiftUI component `NavigationView` does not only offer the navigation itself, but also certain design features.
For example, the developer can set a navigation title which is the main title at the top of the view. This title can either be set as a large title (as in the ScanInstructionView 4.3b) or as an inline title (as in the ScanView 4.5a). If there are so many elements inside a `NavigationView` that the user needs to scroll the page to see them all, the title changes its design to an inline title. It also allows placing buttons in the area next to or above the navigation title (as in the FileView 4.12a).

Modal navigation via fullScreenCover

All views, except the RenameFileView, are presented via a `fullScreenCover` [17]. With this function, a view can be placed on top of another view while covering the entire screen. To dismiss or hide a modal view the following approach is taken. The so-called `presentation- Mode` [23] is an environment variable that SwiftUI provides, which states if a modal is being presented. It also provides a function, called `dismiss` [16], which closes a specific modally presented view.

All Strings managed from localizable file

Normally when creating a `Text`-element, the string that this element should display is placed inside the parenthesis (as seen here: 2.1). This means that when certain texts should be changed, the developer has to navigate to each `Text`-element in the code and change it there.
To make this process more convenient and as a side effect even offer the possibility to support multiple languages in the app, a process called localization is used.
With localization, the content of a `Text`-element is not placed directly inside it, but a key is placed inside it (see listing 4.1). This key refers to the text content.

```swift
Text(LocalizedStringKey("exampleIdentifier"))
```

Listing 4.1: Using a localizable key in a `Text` element

The text identifier and the text content are saved in a file called `Localizable.strings` (see listing 4.2).

```
"exampleIdentifier" = "Example text content";
```

Listing 4.2: Defining a localizable key-value pair

To change text anywhere in the app, the developer only has to find the corresponding text-key in the localizable-file and change the text value/content there. It will be automatically adapted in the `Text` element.

Button design

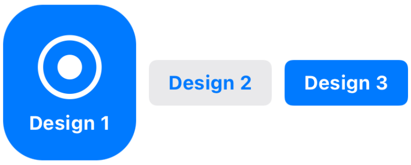
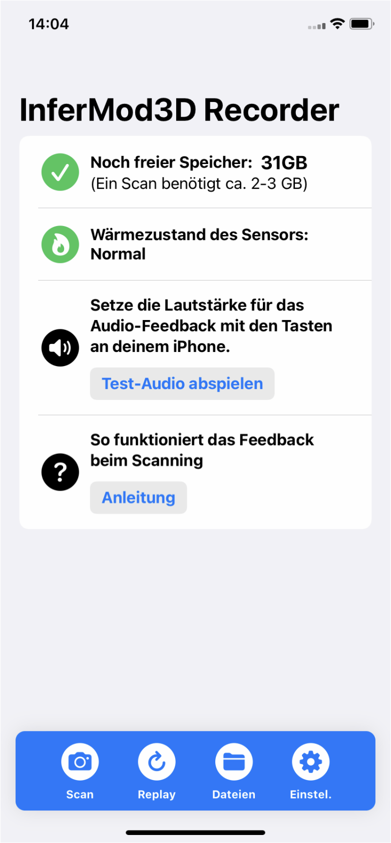


Figure 4.2: The three different button designs in the app

Three different button designs are used in the app (see figure 4.2).
Design 1 is a round button with an icon and text. This design is used, for example, to access a screen like the ScanView or for starting a recording. The text of this button changes according to the current state of the app. The dynamic text is meant to make the app more intuitive and easy to use by providing instructions and tips when to pause a scan replay, for example.
Design 2 is a rather plain design that is used for buttons that the user should be aware of but not for buttons that represent a primary action button. For example, a button to close a view. A close button is not the primary goal of the screen but it is still necessary.
Design 3 is a more prominent design and is meant for a more important button, like renaming a file.

4.2 HomeView

The HomeView (figure 4.3a) is the first view that the user is presented with. It is the entry point to the app. This means that this is the place to show information that is relevant to the user before doing anything in the app, like creating a scan.



14:04

InferMod3D Recorder

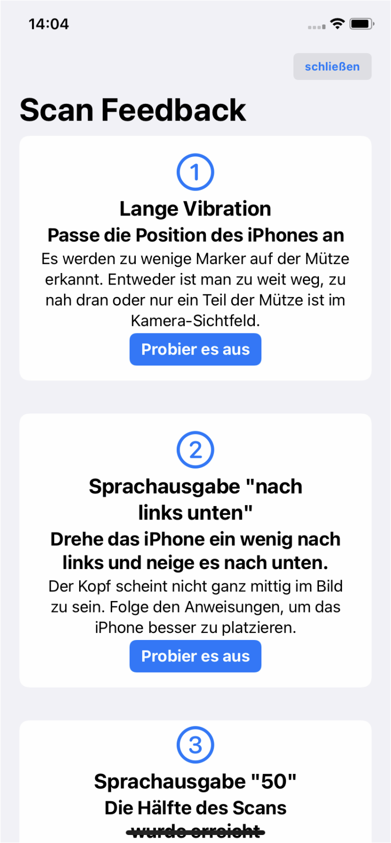
✓ Noch freier Speicher: 31GB
(Ein Scan benötigt ca. 2-3 GB)

Wärmezustand des Sensors:
Normal

Setze die Lautstärke für das Audio-Feedback mit den Tasten an deinem iPhone.

? So funktioniert das Feedback beim Scanning

(a) HomeView



14:04

Scan Feedback

1

Lange Vibration

Passe die Position des iPhones an

Es werden zu wenige Marker auf der Mütze erkannt. Entweder ist man zu weit weg, zu nah dran oder nur ein Teil der Mütze ist im Kamera-Sichtfeld.

2

Sprachausgabe "nach links unten"

Drehe das iPhone ein wenig nach links und neige es nach unten.

Der Kopf scheint nicht ganz mittig im Bild zu sein. Folge den Anweisungen, um das iPhone besser zu platzieren.

3

Sprachausgabe "50"

Die Hälfte des Scans

~~wurde erreicht~~

(b) ScanInstructionsView

Memory check

First, the user needs to know how much free storage is left on the iPhone, as a single scan is about two to three gigabytes large. This prevents the scenario where a scan is created and during the scan, the memory runs out. Once the memory is below three gigabytes the icon will change to red and show a cross. The free memory is calculated when the app launches and every time the ScanView or the FileView are closed because those are the views where either a new scan is recorded or an old scan is deleted, in other words where the amount of free storage will probably change.

Thermal check

Second, for recording a scan the TrueDepth camera system is used, which can easily get hot resulting in the user receiving a warning message to pause its use. To check if the sensor is cold enough again before starting a new scan, he/she can simply check the

HomeView. The icon changes colors from green to orange to red. There are four different thermal levels: `nominal`, `fair`, `serious` and `critical` [24]. When the thermal state is either `serious` or `critical` the sensor should not be used. Color-wise these levels are orange and red. On top of that, a message next to the icon is shown to make it clear to the user, not to start another scan right now. Unlike the free storage space, the thermal state is monitored all the time by listening to the `thermalStateDidChangeNotification` notification from iOS.

Audio check

Third, the app uses audio as a feedback mechanism. In order to understand the audio, the volume must be set sufficiently. This is especially true, if taken into account, that the infant might not be quiet during the scan recording. The audio volume on an iPhone can be set via two buttons on the side of the iPhone. On the HomeView the user is reminded to check the audio volume before starting a scan by playing a test sound to avoid missing out on any information.

Logic check

Fourth, the user might not be aware anymore how the feedback works and what kind of feedback is available. This is why a link to the ScanInstructionsView is provided (4.1).

How audio is played - SoundManager

The class `SoundManager` holds all the functionality to play audio. Text-to-speech is used to play audio to provide feedback to the user. This technique is convenient, especially in the long run, as the text that is said, can be easily modified, instead of needing to create a new recording of a person saying a certain word or phrase. Furthermore, with text-to-speech the volume level is consistent, regardless of the word or phrase, whereas when manually recording a word, it is very hard to always speak at the same volume level.
The class `AVSpeechUtterance` [12] is used to configure the text that should be spoken and in combination with the class `AVSpeechSynthesisVoice` [10] the language of the voice, in our case german, can be set. The class `AVSpeechSynthesizer` synthesizes speech from the provided text and provides the method `speak` to output the audio [11].
Another vital functionality to consider when using audio feedback is to ensure that the audio is always audible. Every iPhone has a silent mode which can be turned on by a button on the side of the iPhone. In this mode, the audio would not be played.

To solve this issue two options were considered:
First, detect when the iPhone is in silent mode and notify the user about it when the ScanView is opened. That way he/she won't forget to turn off the silent mode.
Second, explore an option where audio is still played even in silent mode. The thought that triggered this idea was that music apps like `Apple Music` or `Spotify` would still play audio even if the iPhone is in silent mode, which means there has to be an option for that functionality.
The second approach meant less work for the user, so this was implemented. The class `AVAudioSession` is used to communicate to the system how the audio in the app is used [8]. To achieve the desired functionality, the category `playback` needs to be set 4.3.

```
try AVAudioSession.sharedInstance().setCategory(.playback)
```

Listing 4.3: Configure the audio session to continue playing in silent mode

4.3 ScanInstructionsView

This view (figure 4.3b) helps the user remember the different feedback mechanisms. Each list item represents one feedback type. The main heading of a list item is a description of the specific feedback. The secondary heading tells the user what he/she should do when that feedback occurs. The description text offers further details about the feedback type. Lastly, there is a button in each list item, that lets the user try out the corresponding feedback. This can either be audio or vibration. That way he/she knows exactly what kind of feedback the text is referring to.
Refer to section 4.2 to see how audio is played.

How vibrations are generated - VibrationManager

The class `VibrationManager` holds all the functionality to generate vibrations. First, the framework `CoreHaptics` [15] is imported. Refer to figure 4.4 to see what components are necessary to generate a custom vibration. An instance of the class `CHHapticEngine` creates a connection to the hardware of the iPhone that is responsible for playing haptic feedback [14].
The next step is to create the actual haptic pattern. A haptic pattern is represented by the class `CHHapticPattern`. A haptic pattern is made up of haptic events. A haptic event consists of an `eventType`, an intensity, a sharpness and a duration.

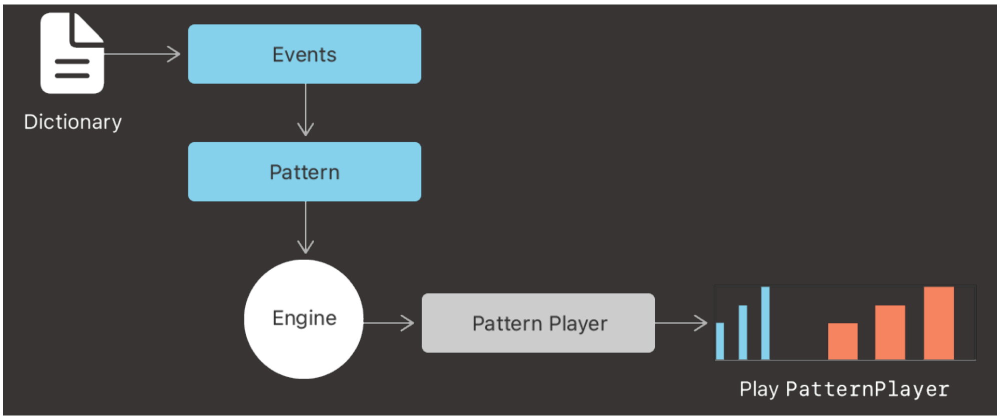


Figure 4.4: The necessary components to generate a custom haptic pattern [14]

The last step is to create a haptic player of the type `CHHapticAdvancedPatternPlayer` which provides the methods `start` and `stop`.

4.4 ScanView

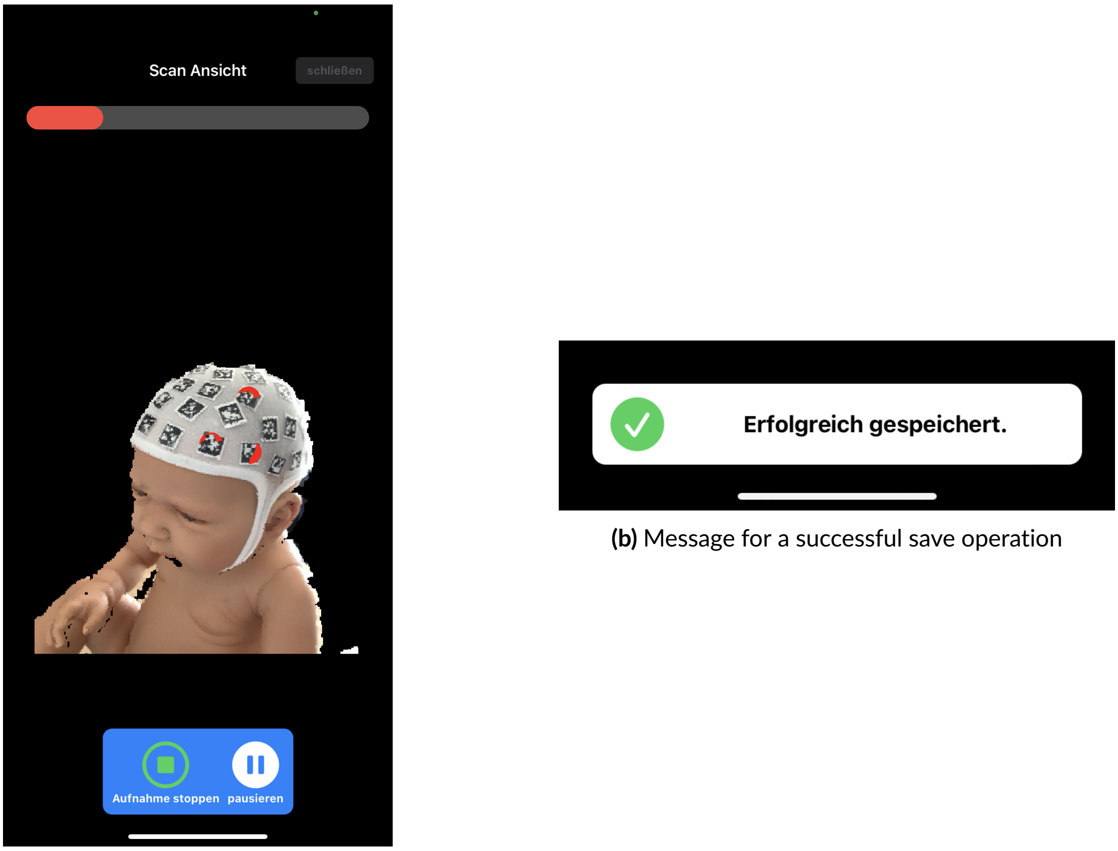
The ScanView (figure 4.5a) is used for creating a scan of the head of the infant. This is the view where feedback regarding the scan is provided to the user. The main part of this view displays a real-time video feed in form of a point cloud.

Start a recording

At the bottom of the screen in the blue bar, the user can start the recording by pressing the corresponding button. This action triggers a progress bar to appear at the top of the screen (top of 4.5a), which displays the overall progress of the scan. The progress bar uses the same colors as a traffic light to be as intuitive as possible for the user. The color red is used for a progress of zero percent to 49 percent, orange for 50 percent to 99 percent and green for 100 percent. Furthermore, the haptic and audio feedback are now activated as well. The button at the top-right corner to close the view is now deactivated, so that the view is not accidentally closed while recording.

Pausing a recording

When the user pauses the recording, the feedback is also paused and the camera is turned off as it is not needed and this also allows the TrueDepth sensor to cool down.



Erfolgreich gespeichert.

(b) Message for a successful save operation

(a) ScanView

Stopping a recording

Once the recording is stopped, a confirmation message (4.5b) appears at the bottom of the screen. After one second the PatientMetaDataView is opened automatically.

Memory alert

If upon opening the ScanView there are less than three gigabytes of free storage left, an alert appears to inform the user about this. On the HomeView the memory row also displays this fact, but in case the user missed it, he/she is reminded of it again.
If the TrueDepth sensor gets too hot while the ScanView is open, an alert is shown that informs the user about the thermal state of the TrueDepth sensor and also tells the user to pause the recording.

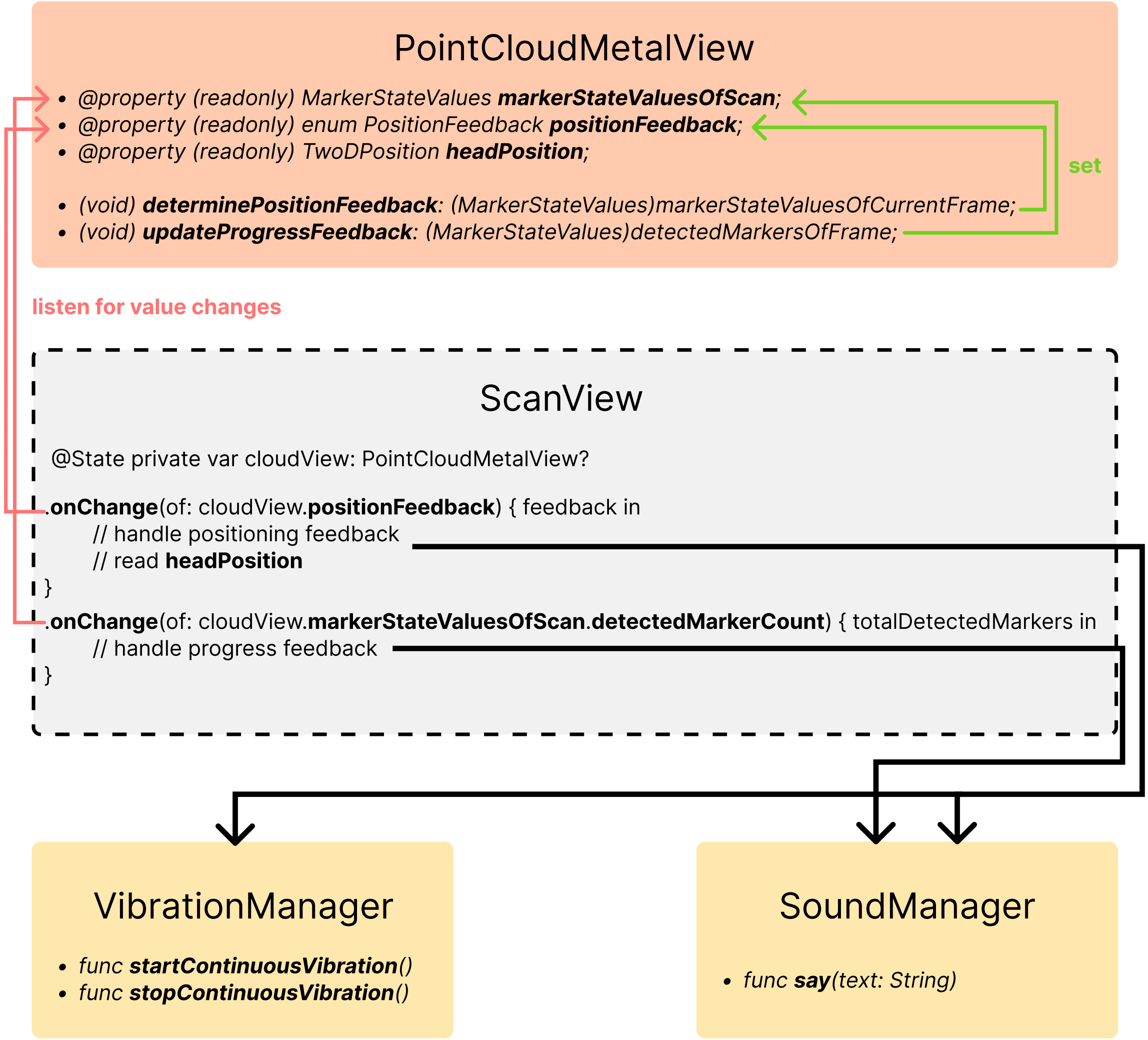

PointCloudMetalView
• @property (readonly) MarkerStateValues markerStateValuesOfScan;
• @property (readonly) enum PositionFeedback positionFeedback;
• @property (readonly) TwoDPosition headPosition;
• (void) determinePositionFeedback: (MarkerStateValues)markerStateValuesOfCurrentFrame;
• (void) updateProgressFeedback: (MarkerStateValues)detectedMarkersOfFrame;
set
listen for value changes
ScanView
@State private var cloudView: PointCloudMetalView?
.onChange(of: cloudView.positionFeedback) { feedback in
// handle positioning feedback
// read headPosition
}
.onChange(of: cloudView.markerStateValuesOfScan.detectedMarkerCount) { totalDetectedMarkers in
// handle progress feedback
}
VibrationManager
• func startContinuousVibration()
• func stopContinuousVibration()
SoundManager
• func say(text: String)

Figure 4.6: The communication from backend to fronted for providing feedback

Dataflow from the marker detection to the feedback execution

Sections 3.3 and 3.3 already described the idea behind the implemented feedback logic. This section explains the dataflow from detecting the ArUco markers in the video to triggering the appropriate feedback. Refer to figure 4.6 for a visual representation of the data flow.

The class `PointCloudMetalView` receives the camera and sensor data 30 times per second and holds all the functionality for creating a point cloud from that data with the framework `Metal`. The classes `VibrationManager` and `SoundManager` were already introduced in sections 4.3 and 4.2.

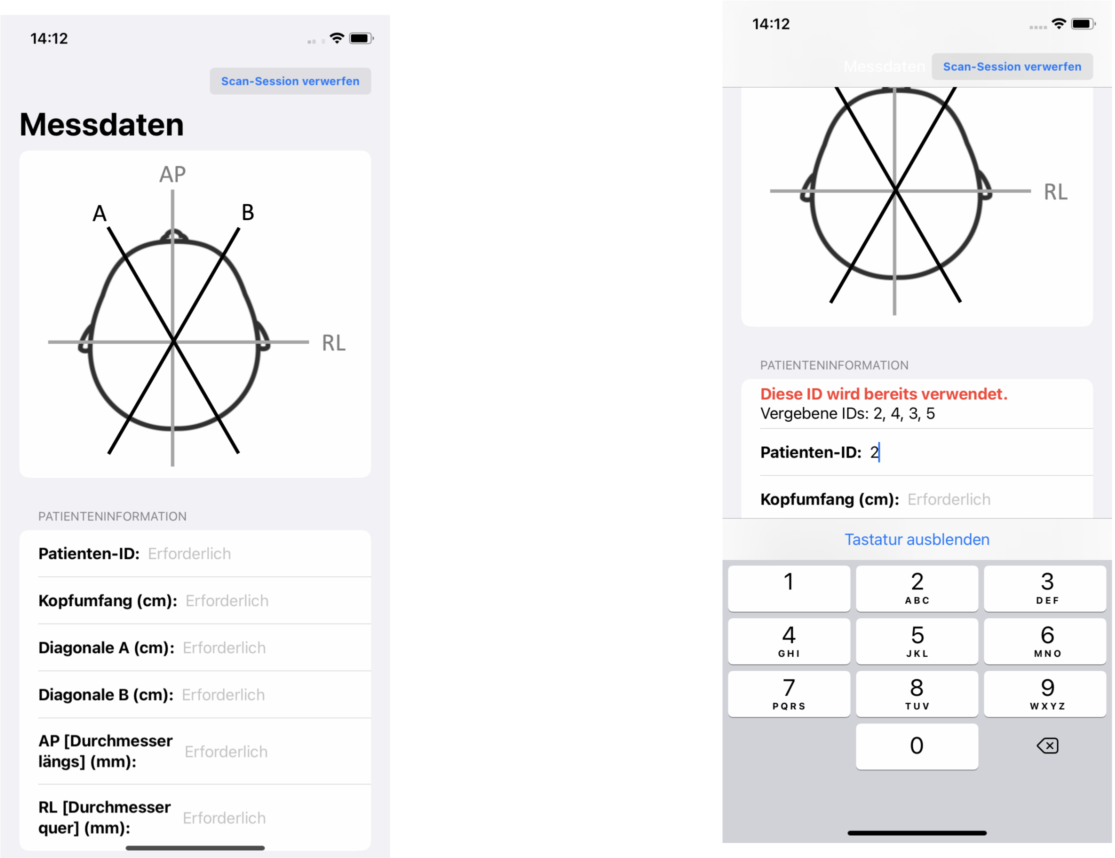
In the class `PointCloudMetalView` the data that is extracted from the ArUco markers is directly processed in the methods `determinePositionFeedback` and `updateProgressFeedback`. Only if the position feedback (ideal position, too close, otherwise off) is the same four times in a row, the property `positionFeedback` is updated. In ScanView this property is observed and the `onChange` function will be executed every time the value changes. Depending on which value the property `positionFeedback` and the property `headPosition` holds, inside `onChange` either a vibration and/or an audio output is triggered. The scan progress is handled the same, except that there is no threshold that has to be surpassed.

4.5 PatientMetaDataView

The PatientMetaDataView (figure 4.7a) is used for collecting data of the infant's head that was measured using the "old" measurement method by using a calliper to manually measure the head of the child. The result from this method can then be compared with the measurement result of the new 3D method.

Aborting a scan session

If the user just recorded a scan to try it out and doesn't want to save it and therefore doesn't want to collect any measurement data, he/she can abort the session by tapping the button at the top-right corner of the screen that triggers an alert to confirm the user's choice. By confirming, the previously recorded scan is deleted and the user is navigated back to the HomeView.



14:12

Scan-Session verwerfen

Messdaten

PATIENTENINFORMATION

Patienten-ID: Erforderlich

Kopfumfang (cm): Erforderlich

Diagonale A (cm): Erforderlich

Diagonale B (cm): Erforderlich

AP [Durchmesser längs] (mm): Erforderlich

RL [Durchmesser quer] (mm): Erforderlich

14:12

Messdaten Scan-Session verwerfen

PATIENTENINFORMATION

Diese ID wird bereits verwendet.
Vergebene IDs: 2, 4, 3, 5

Patienten-ID: 2

Kopfumfang (cm): Erforderlich

Tastatur ausblenden

(a) PatientMetaDataView

(b) PatientMetaDataView: Entering an already used patient ID

Text field validation

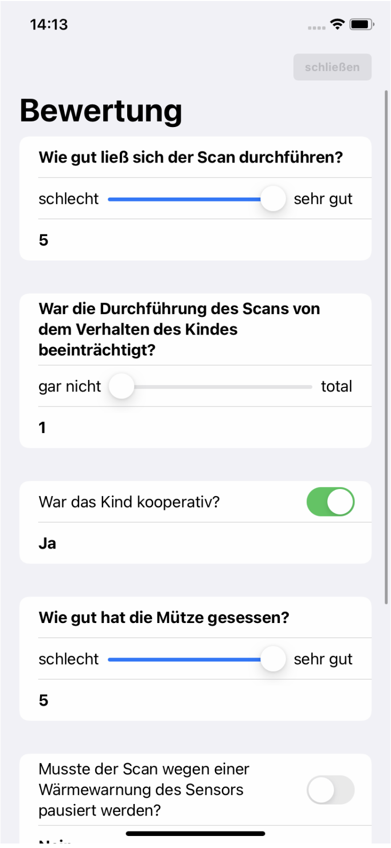
The image at the top of the screen (figure 4.7a) illustrates which text field refers to which part of the head. Each text field validates its inputs. Only if all text fields contain a valid input, does the save button at the bottom of the screen become active.
The patient-ID field can only be an integer number. Because of that, the user receives a keyboard that only contains numbers, no commas and no letters or other symbols. This again follows the guiding principle, to only show relevant information to the user, that he/she should use. On top of that, the entered ID is compared to all IDs that are currently saved on the device, to avoid using an ID multiple times. If an already used ID is entered, a warning appears above the text field that shows the user the IDs that are already in use (see 4.7b).
The rest of the text fields represent data that can be a floating-point number. This is why the number pads also contain a comma in these cases. Each input is validated to be

a valid decimal number. By mistake, the user could type two commas, which would be invalid. If a number is invalid the text field label turns red.

Saving and continuing

Once all text fields contain valid inputs, the save button becomes active and saves the data in a JSON-formatted file. The filename contains the ID that the user assigned to the patient. The scan-filename consisted of a temporary name until now as no ID has been assigned yet. But now the scan-filename can be changed to also contain the ID. After that the ScanRatingView is automatically opened.

4.6 ScanRatingView



Bewertung

Wie gut ließ sich der Scan durchführen?

schlecht sehr gut

5

War die Durchführung des Scans von dem Verhalten des Kindes beeinträchtigt?

gar nicht total

1

War das Kind kooperativ?

Ja

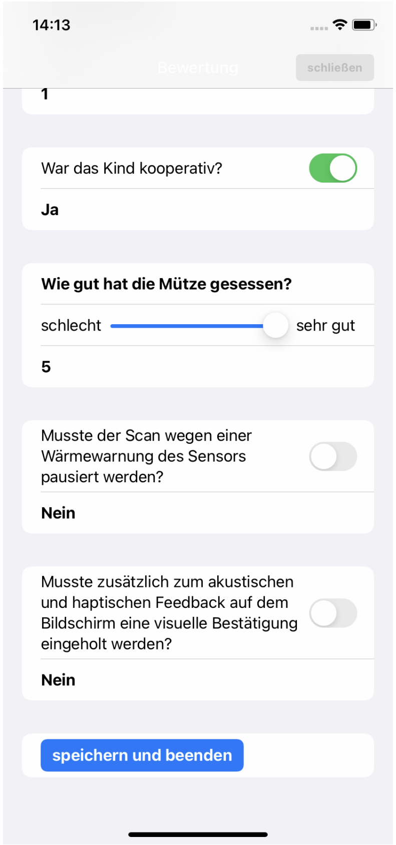
Wie gut hat die Mütze gesessen?

schlecht sehr gut

5

Musste der Scan wegen einer Wärmewarnung des Sensors pausiert werden?

(a) ScanRatingView



1

War das Kind kooperativ?

Ja

Wie gut hat die Mütze gesessen?

schlecht sehr gut

5

Musste der Scan wegen einer Wärmewarnung des Sensors pausiert werden?

Nein

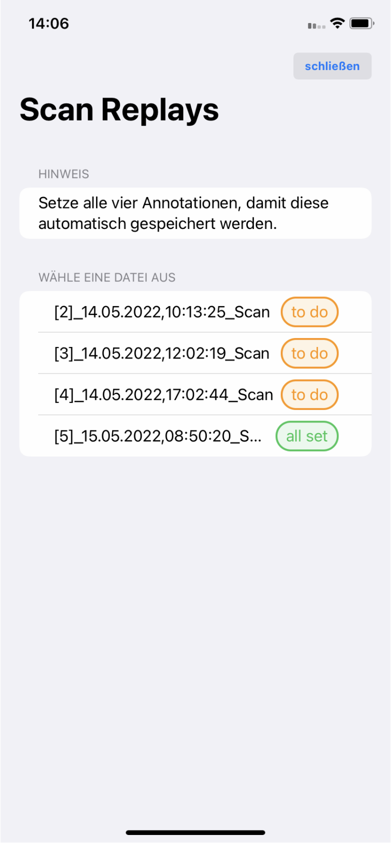
Musste zusätzlich zum akustischen und haptischen Feedback auf dem Bildschirm eine visuelle Bestätigung eingeholt werden?

Nein

(b) Second part of the ScanRatingView

The ScanRatingView (figure 4.8a) is used to evaluate the effectiveness of the feedback concept that is used during the scan. To make this a quick process and not use too much of the physical therapist's time, sliders and toggle switches are used, except for one text field to give further information about the infant's behaviour if the infant was not co-operative. The sliders have a value range of one to five. The questions are also about the behaviour of the child. As a very calm behaviour might lead to a better scan than a restless behaviour.
Just like for the collected measurement data of the PatientMetaDataView, the rating data is also saved in a JSON-formatted file. On top of the values that are shown on screen, the choices about the feedback, that can be set in the SettingsView, are automatically stored in the same file.

4.7 ScanReplaySelectionView


14:06

Scan Replays

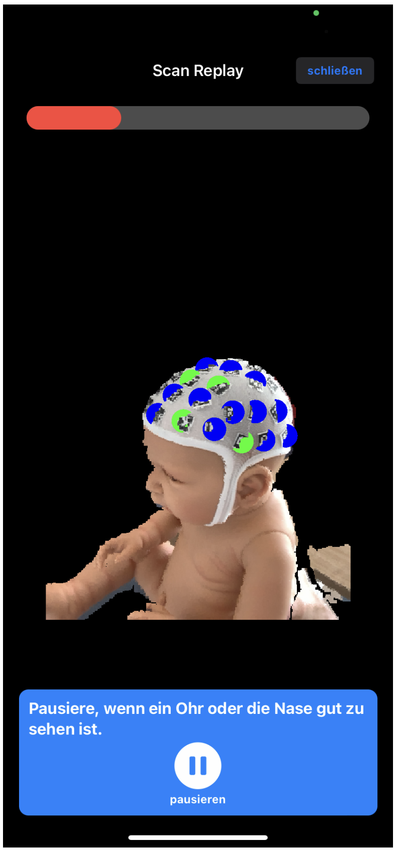
HINWEIS

Setze alle vier Annotationen, damit diese automatisch gespeichert werden.

WÄHLE EINE DATEI AUS

[2]_14.05.2022,10:13:25_Scan to do
[3]_14.05.2022,12:02:19_Scan to do
[4]_14.05.2022,17:02:44_Scan to do
[5]_15.05.2022,08:50:20_S... all set

(a) ScanReplaySelectionView



Pausiere, wenn ein Ohr oder die Nase gut zu sehen ist.

(b) ScanReplayView

To select a created scan for replay, the user navigates to the ScanReplaySelectionView (figure 4.9a) where he/she is presented with a list of all the scans that the app saved.
A list row is made up of the filename and a tag. This tag either displays "to do" or "all set". This phrase is referring to the status of the annotations. If annotations to the nose tip, nasion, left ear and right ear are set, the status switches from "to do" to "all set". Refer to section 3.2 for an explanation of why annotations are needed. Tapping on a list row opens the ScanReplayView.
When a scan replay is closed and the scan list is shown again, a message at the bottom of the screen appears, that informs the user whether the annotations were successfully saved in an annotation file. A hint above the list, informs the user about this saving behaviour.

How to identify a scan file

As the app does not only save scan files but also files for the annotations and markers, the manual CVAI measurement and the rating of a scan, there has to be a way to distinguish them.
The name of a scan file is formed in the following format:
[ID]_date_Scan
A scan file always has an ID, a date that forms the first part of the filename and the word `Scan`. Only files that have an ID and contain the word `Scan` are shown in the list.

Solving a Memory Leak during development

During the development the following scenario would occur:
After opening and closing about three to five scan replays, the app would crash with a few different error messages:
`Cannot allocate memory, Insufficient memory, Cannot create buffer of zero length, Failed to allocate IOSurface`
This behaviour of course should not occur as the user should be able to open as many replays after one another as he/she wishes to.
As mentioned before, a scan itself can be about two to three gigabytes large and the error occurred on an iPhone XR, which has three gigabytes of main memory. So if the scan would be loaded into main memory for replay all at once, the scenario where the app crashes due to insufficient memory would be valid. However, this is not the case as the scan is read into memory frame by frame.

Automatic Reference Counting and Retain cycles

The next suspicion is a memory leak caused by a so-called retain cycle. To understand what a retain cycle is, the concept of how Swift manages memory must be understood first.
Swift uses Automatic Reference Counting (ARC). This means that Swift counts the number of references to a class instance. A reference is called a strong reference when a class instance is assigned to a variable or property [7]. By counting the number of references, Swift knows when a class is used or not. Once the reference count reaches zero, Swift can safely deallocate this class instance as it is no longer used. The deallocation frees up the memory that was used by the class instance. To know when a class is initialized and deinitialized (deallocated) the methods `init` and `deinit` can be implemented in the class.
A retain cycle is caused when a class does not get deinitialized as there is a strong reference to that class instance, so the reference count is greater than zero. This can, for example, happen if two classes refer to each other and keep each other alive [7]. If this happens, often the memory grows and grows which causes a memory leak.

How to detect a retain cycle

The next step is to detect the retain cycle. First, the `deinit`-method and the `init`-method of all classes that are used are implemented with a `print` statement to log to the console when and if they are initialized and deinitialized. The expected behaviour, in this case, would be that all the classes that are used inside the ScanReplayView are deinitialized when the ScanReplayView is closed. It turns out that all the classes except the class `CameraManager`, `PointCloudMetalView`, `PointCloud` are deinitialized. This behaviour can also be detected by letting Xcode build a memory graph of a specific situation. When this memory graph is built after the ScanReplayView was closed, it showed that the mentioned classes are still held in memory. Figure 4.10 shows the constellation of classes and structs that were used to display a point cloud in the ScanReplayView.

- The class `PointCloudMetalView` is responsible for processing the camera and sensor data and creating a point cloud from this data with the framework Metal.

- The class `PointCloud` is responsible for instantiating `PointCloudMetalView` and sharing that instance with the ScanReplayView. This allows the ScanReplayView to access variables and methods from `PointCloudMetalView`.

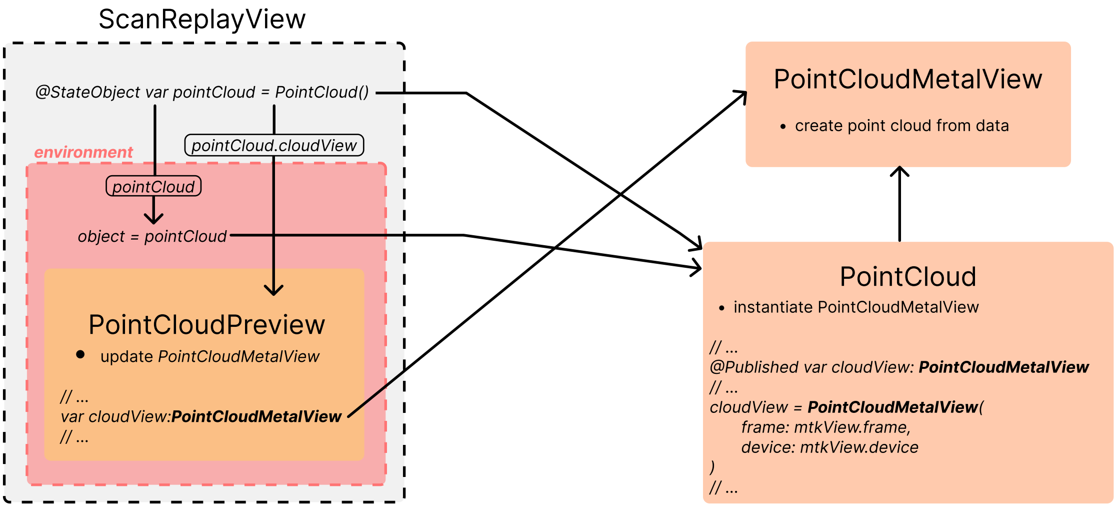

ScanReplayView
@StateObject var pointCloud = PointCloud()
environment
pointCloud.cloudView
pointCloud
object = pointCloud
PointCloudPreview
• update PointCloudMetalView
// ...
var cloudView:PointCloudMetalView
// ...
PointCloudMetalView
• create point cloud from data
PointCloud
• instantiate PointCloudMetalView
// ...
@Published var cloudView: PointCloudMetalView
// ...
cloudView = PointCloudMetalView(
frame: mtkView.frame,
device: mtkView.device
)
// ...

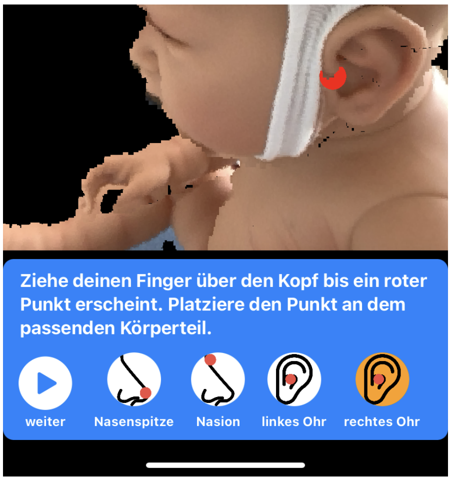
Figure 4.10: Constellation for displaying a point cloud that caused a retain cycle

- To display a Metal view in SwiftUI the struct `PointCloudPreview` is necessary. It updates the `PointCloudMetalView` with new camera and sensor data.
- Inside ScanReplayView there are other views that need access to the `pointCloud`-instance. To avoid having to pass that instance as a parameter every time, it is placed inside the environment, through which the subviews can access this instance.

The combination of passing the `PointCloud` instance to the environment and the `PointCloudMetalView` instance, inside the `PointCloud` instance, to `PointCloudPreview` causes the problem that the classes `PointCloudMetalView` and `PointCloud` are not deinitialized after the ScanReplayView is closed.

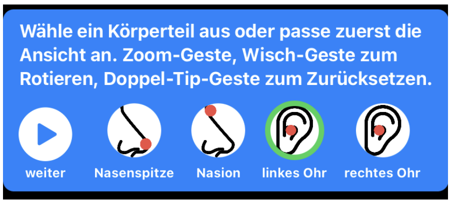
4.8 ScanReplayView

To be able to create a correct and complete 3D model of the scanned head, the user has to locate certain parts of the head in the scan (as explained in section 3.2). These parts are the nose tip, nasion, left ear and right ear. The ScanReplayView (figure 4.9b) is the place where the user locates these body parts by placing a dot, an annotation, on this body part.
Like the ScanView, it is made up of a progress bar at the top of the screen, a blue tile with a textual instruction and button at the bottom. The center of the screen is made up of



Ziehe deinen Finger über den Kopf bis ein roter Punkt erscheint. Platziere den Punkt an dem passenden Körperteil.

(a) ScanReplayView: Setting an annotation to the left ear



Wähle ein Körperteil aus oder passe zuerst die Ansicht an. Zoom-Geste, Wisch-Geste zum Rotieren, Doppel-Tip-Geste zum Zurücksetzen.

(b) ScanReplayView: After setting an annotation

the point cloud that is replayed. The text in the blue tile changes depending on the state of the replay to give the user helpful instructions.

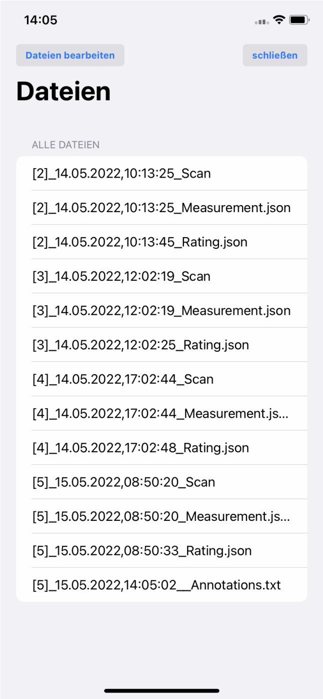
Setting an annotation

To set an annotation, the user has to pause the replay. To simplify the process of setting an annotation to a specific body part, the point cloud can be enlarged with a zoom gesture, it can be rotated with a drag gesture and can finally be reset to the default perspective by double-tapping on the screen.
To set an annotation, the user selects the body part (4.11a) and can then drag over the screen, as described in the text above the buttons. After moving the annotation and deselecting the button, he/she sees a visual confirmation that this annotation was set, by the green stroke around the button (4.11b).
An annotation can only be set when the frame it was paused in, has enough marker overlap. To avoid the scenario where the user pauses the replay and ends up in a state with insufficient marker overlap, the pause feature waits for that moment and pauses only then. So it is more like a "pause once the moment is ideal" feature.

4.9 FileView

The FileView offers functionality for the following scenarios:

- Delete files from the iPhone after exporting them.



14:05

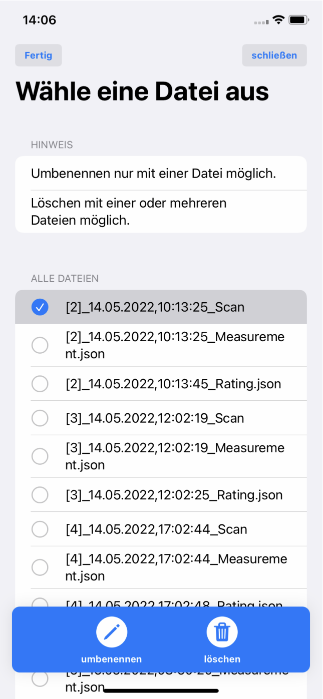
Dateien bearbeiten | schließen

Dateien

ALLE DATEIEN

[2]_14.05.2022,10:13:25_Scan
[2]_14.05.2022,10:13:25_Measurement.json
[2]_14.05.2022,10:13:45_Rating.json
[3]_14.05.2022,12:02:19_Scan
[3]_14.05.2022,12:02:19_Measurement.json
[3]_14.05.2022,12:02:25_Rating.json
[4]_14.05.2022,17:02:44_Scan
[4]_14.05.2022,17:02:44_Measurement.js...
[4]_14.05.2022,17:02:48_Rating.json
[5]_15.05.2022,08:50:20_Scan
[5]_15.05.2022,08:50:20_Measurement.js...
[5]_15.05.2022,08:50:33_Rating.json
[5]_15.05.2022,14:05:02__Annotations.txt

(a) FileView



14:06

Fertig | schließen

Wähle eine Datei aus

HINWEIS

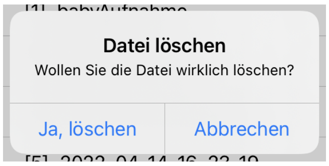
Umbenennen nur mit einer Datei möglich.

Löschen mit einer oder mehreren Dateien möglich.

ALLE DATEIEN

☑ [2]_14.05.2022,10:13:25_Scan
○ [2]_14.05.2022,10:13:25_Measurement.json
○ [2]_14.05.2022,10:13:45_Rating.json
○ [3]_14.05.2022,12:02:19_Scan
○ [3]_14.05.2022,12:02:19_Measurement.json
○ [3]_14.05.2022,12:02:25_Rating.json
○ [4]_14.05.2022,17:02:44_Scan
○ [4]_14.05.2022,17:02:44_Measurement.json

(b) FileView in edit mode



Datei löschen

Wollen Sie die Datei wirklich löschen?

(c) FileView confirm to delete alert

- One or more files should be deleted if they were created by accident or just to try out the scan functionality.
- A file should be renamed to maybe include the name of the patient.

The FileView (figure 4.12a) allows the user to rename and delete files. The file list displays all of the files that the app saved. Tapping the button "Dateien bearbeiten" at the top left of the screen activates the edit mode (figure 4.12b). Above the list, two hints are displayed, which inform the user about the possibilities. Only one file can be renamed at a time, whereas one or multiple files can be deleted at a time. In edit mode the list turns into a selectable list, meaning that the user can select one or multiple rows from the list. In the blue tile at the bottom of the screen, there is a rename button and a delete button. If one file from the list is selected, both buttons can be tapped. If more than one file is selected, the rename button is deactivated.
After tapping the delete button an alert appears (figure 4.12c) to confirm the user's choice to permanently delete the selected files.
After tapping the rename button the RenameFileView appears. Once the RenameFileView is closed, after a successful rename, the file list is reloaded.

How to make a list selectable

Before deleting or renaming a file, it has to be selected from the list. The FileView uses a `List`-element to display all the files. A SwiftUI list can be divided into sections. In figure 4.12b there are two sections, one for the hints and one for the files. Inside the files section a `ForEach`-loop goes through the array of filenames and displays them.
The first step for achieving the functionality and design to select one or more rows of a list (figure 4.12b), is to pass a set of strings to the `List`-element. This set represents the current selection of list rows.
Step two is to access an environment value of SwiftUI which is called `editMode`. The `editMode` is either `active` or `inactive`. The `List`-element automatically changes its appearance once the `editMode` changes. This mode is toggled by the button at the top-left of the screen.

4.10 RenameFileView

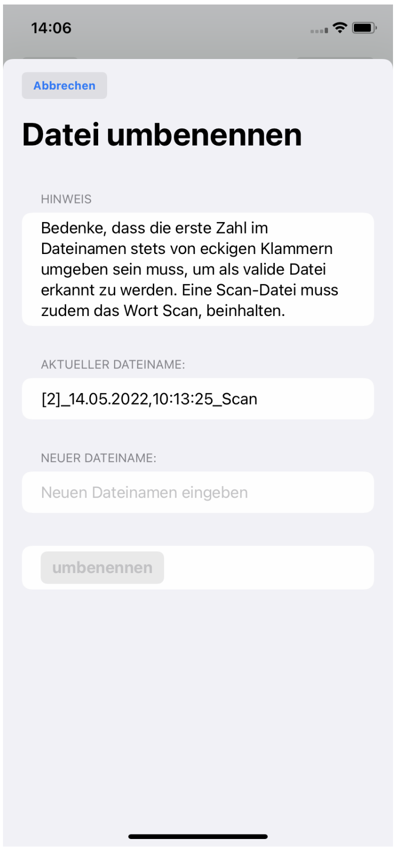
The RenameFileView (figure 4.13a) displays the current name of the file and a text field to enter a new name. Once a new filename is entered the rename button is activated. A hint at the top of the screen informs the user about the format of the filenames so that he/she does not accidentally destroy the format. After tapping the rename button a message appears whether the renaming was successful (figure 4.13b). The RenameFileView automatically dismisses itself after three seconds. Listing 4.10 shows how to execute code after a certain number of seconds.

```
DispatchQueue.main.asyncAfter(deadline: .now() + 3.0) {
    // code to execute
}
```

How to rename a file

At first a method called `rename` makes sure that the current filename exists and that the new filename does not exist. This is done by the method `fileExists` of the class `FileManager`. Renaming a file is achieved by moving the file to a new location 4.10.

```
try FileManager.default.moveItem(at: originalPath, to: targetPath)
```



14:06

Abbrechen

Datei umbenennen

HINWEIS

Bedenke, dass die erste Zahl im Dateinamen stets von eckigen Klammern umgeben sein muss, um als valide Datei erkannt zu werden. Eine Scan-Datei muss zudem das Wort Scan, beinhalten.

AKTUELLER DATEINAME:

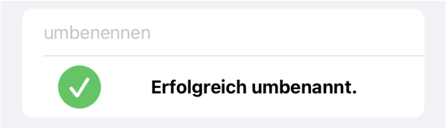
[2]_14.05.2022,10:13:25_Scan

NEUER DATEINAME:

Neuen Dateinamen eingeben

umbenennen

(a) RenameFileView



umbenennen

✓ Erfolgreich umbenannt.

(b) RenameFileView - rename success message

4.11 SettingsView

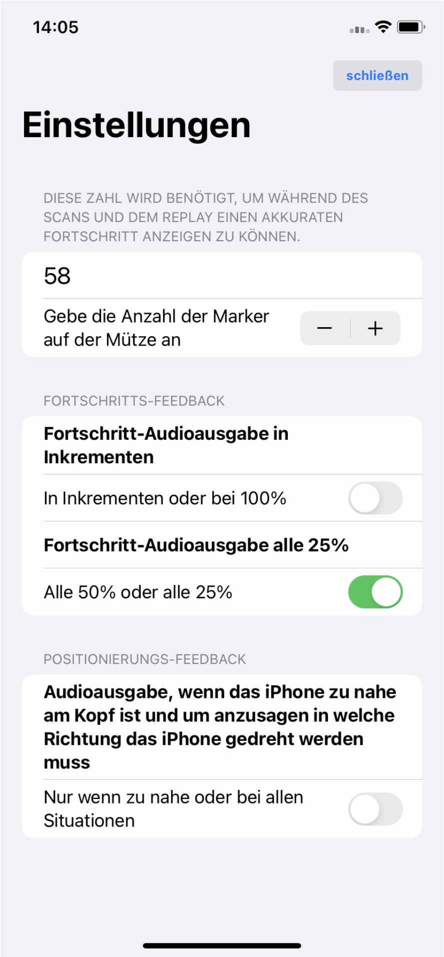
The SettingsView allows the user to configure a few settings that affect his/her experience while scanning.

Setting the number of markers

During scanning or during the replay of a scan, the progress of the scan is shown via a progress bar and via audio. To correctly calculate the progress, the total number of markers on the cap must be set.

Setting the increments for the progress feedback

The user can also configure the number of increments in which he/she wants to receive audio-feedback about the scan progress. The options are every 25 percent, every 50 percent or just at 100 percent.



14:05

schließen

Einstellungen

DIESE ZAHL WIRD BENÖTIGT, UM WÄHREND DES SCANS UND DEM REPLAY EINEN AKKURATEN FORTSCHRITT ANZEIGEN ZU KÖNNEN.

58

Gebe die Anzahl der Marker auf der Mütze an

FORTSCHRITTS-FEEDBACK

Fortschritt-Audioausgabe in Inkrementen

In Inkrementen oder bei 100% ○

Fortschritt-Audioausgabe alle 25%

Alle 50% oder alle 25% ●

POSITIONIERUNGS-FEEDBACK

Audioausgabe, wenn das iPhone zu nahe am Kopf ist und um anzusagen in welche Richtung das iPhone gedreht werden muss

Nur wenn zu nahe oder bei allen Situationen ○

Figure 4.14: SettingsView

Setting the audio feedback for the positioning

When the position of the iPhone is not ideal during the recording of a scan, the device will vibrate and this can't be changed. However, the user can select, if he/she just wants to get audio feedback when the device is too close to the head or if he/she also wants to receive instructions in which direction to move the device.

Persisting the user configuration

When the user modifies a configuration, it should be permanently saved and not lost after the app is closed. All the choices that can be configured in the SettingsView are persisted in the `UserDefaults` storage of the iPhone. `UserDefaults` is a database where key-value pairs can be persisted. It is only meant for storing very small amounts of data, like the number of markers on a hat. Such values could also be stored in the document directory. However, it is much more convenient to use `UserDefaults` for that, especially in SwiftUI. Only one line of code is necessary to access a value from `UserDefaults` (4.11).

```
@AppStorage("markerCountOnHat") var markerCount: Int = 57
```

By assigning a new value to the variable, it is automatically saved.

5 Testing and Evaluation

To test and evaluate the features of the app and even more importantly the user experience of the feedback concept, it is used by a team of physical therapists at Varilag for four weeks. During that time, an initial feedback about the scan process is collected via the ScanRatingView, as described in section 4.6. This feedback is collected after every scan. To get an even better understanding about the pros and cons of the app, an in-person interview will be conducted with the physical therapists that used the app in the four weeks. The results of both evaluations will be presented in this section.

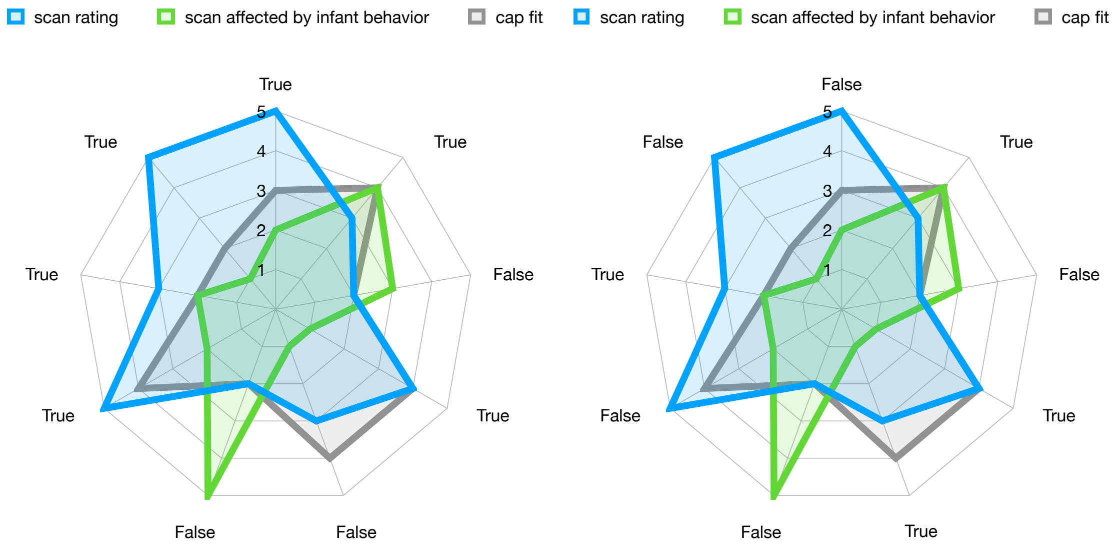
5.1 Collected data of the scan ratings

The following questions were asked after every scan:

- How well did the scan go?
- Was the performance of the scan affected by the child's behavior?
- Was the child cooperative?
- How well did the cap fit?
- Did the scan have to be paused because of a sensor heat warning?
- Did visual confirmation need to be obtained in addition to audible and haptic feedback on the screen?

The following feedback configurations were also saved in the rating file:

- Play progress audio only at 100 percent.
- Play progress audio every 50 percent or every 25 percent.
- Play positioning audio only if the device is too close to the infant's head.



True = infant was cooperative

True = visual feedback was also needed

Figure 5.1: Evaluation of the data collected through the ScanRatingView

Results and evaluation of the scan ratings

Refer to attachment 1 for all the collected data in the ScanRatingView. Nine scan ratings were collected in total.

Figure 5.1 shows two net diagrams, each displaying the overall scan rating if the scan was affected by the behaviour of the infant and how well the cap with the markers fit on the head. The diagram on the left puts these metrics in relation to the cooperation of the infant and on the right in relation to the metric if visual feedback was also needed, i.e. if the user had to look at the screen.

The diagrams show that when the scan rating was bad (2) the fit of the cap was bad (2) and the infant was not cooperative. Furthermore, the scan was affected by the infant's behavior (3 and 5) and visual feedback was not needed (see ratings 2 and 5 in attachment 1). This suggests that the scan was not good because of the infant's behavior and the fit of the cap.

When the scan rating was good (4 and 5), the cap fit was in the range of 2 to 4, the scan was not affected by the infant's behavior (1 and 2), the infant was cooperative and visual feedback was only needed once (see ratings 0, 3, 6, 8 in attachment 1). Although the infant was always cooperative, the scan was not affected by the infant's behavior which suggests that the scan was good because of the app itself and not because of the infant's behavior. Unlike bad scans, a good scan does not seem to correlate with how the cap

fits.
However rating 1 (in attachment 1) shows an overall rating of 3 with a fitting cap (4). The infant was not cooperative, but the scan was not affected by the infant's behavior and visual feedback was needed which suggests that the scan was mediocre because of the app itself and not because of the infant or the cap.
Furthermore, in rating 5 (in attachment 1), the scan was rated with a 2 and the infant was not cooperative as it turned away its head, the user needed to pause the scan due to a heat warning. As this is the only scan where a heat warning happened a plausible explanation is that the scan process took longer because the infant turned away its head all the time.
Lastly, the average scan rating is 3.56. In total there were four very good (ratings of 4 or 5) scans and five bad (ratings of 1, 2 or 3) scans.

5.2 Results of the in-person interview

Refer to the appendix 2, 3, 4 and 5 for the notes taken in each interview.
The times the app was used by each physical therapist in the four weeks ranges from one time by interviewee 4 to about five times by interviewee 5. Due to a very tight schedule, they were only able to use the ScanView, PatientMetaDataView and the ScanRatingView. Therefore they were also unable to explore all features of the app. The content of the interviews is about the scan feedback, how the infant reacted and what the parents thought.

Positioning feedback

All interviewees agreed that the part of the positioning feedback that provides information about the distance to the head via audio, was really helpful and always allowed to find the correct distance.
The opinion about the positioning instructions on how to move the iPhone differed. Interviewee 2, 3 and 5 found them a little unintuitive and hard to interpret whereas interviewee 4 had no problem at all with them.
For interviewee 3 it sometimes felt like the iPhone was constantly vibrating, but when looking at the screen the positioning appeared to be fine. On the other hand, interviewee 5 really liked them.

Progress feedback

For interviewee 2 the progress feedback seemed confusing sometimes as it said "done" although, from the therapist's personal feeling, the scan could not be done yet. Apart from that, the feedback about the progress was very positive and interviewee 3 even said that it was motivational and interesting for the parents.
Interviewees 2 and 4 would have liked it even better if there was feedback at 50% of the progress, which there is, however, due to the time constraint they were not able to explore the app and therefore didn't find out about that feature. The same happened to interviewee 5 who had progress feedback at 50%, but wished to have had it at 25% increments.

Parents

The parents were very open-minded, motivated and interested. Except for one case with interviewee 5 they never asked questions about the app or any results. In this one case, they were wondering where the images, taken of the head of their child, are used. To make sure that they were not published anywhere. When the cap didn't quite fit the head of their child or their child complained, the parents sometimes became a little nervous.

Infants

Interviewees 4 and 5 mentioned that the infants were sometimes distracted by the display of the iPhone and turned their heads while the iPhone was moved around them. Interviewee 4 solved this by making the parents distract their child. Interviewee 5 added that the vibrations or the audio did not influence the child at all.

Cap

All interviewees agreed that the cap was the biggest problem and the cause of a bad scan. First, all the infants did not like when the cap was put on, which didn't make the parents feel any better. Second, interviewees 4 and 5 complained that the cap was too small for some children and barely closed at the bottom. Third, interviewees 3 and 5 mentioned that the stickers did not hold well and would often fall off which caused interviewee 5 not to reach the scan completion in some cases because the number of stickers left on the cap and the configured sticker amount in the SettingsView differed. Interviewee 3 also mentioned that the color on the stickers smears.

Improvements

Both interviewee 2 and 3 agreed that an improvement would be to include a feature that reminds the app operator to specifically scan the nose and ears at the end of the scan. Even better would be a feature that detects if the ears and nose are properly captured in the scan.

Interviewee 2 would find it more intuitive if the positioning instructions (for example "left-up") were replaced by instructions about which part of the head is not properly scanned yet. Interviewee 3 would find it more intuitive if a positioning instruction like "left-up" means that the head of the infant should move to the top left in the camera instead of the iPhone itself.

Lastly, interviewee 2 would prefer a more descriptive picture in the PatientMetaDataView instead of just for example "Diagonal A".

5.3 Combining the ScanRatingView data and the interviews

The evaluation of the rating data captured in the ScanRatingView still allowed for multiple different interpretations, for example, about the causes of a bad scan rating. These interpretations can now be augmented with the content of the interviews.

The interviews confirmed the assumption that a bad scan was mostly due to the fit of the cap or the stickers not holding properly.

They also confirmed that a scan could still be good even though the cap fit might not be ideal.

Additionally, a scan with hard-to-follow positioning instructions, because at the time they seemed non-intuitive, can also worsen the overall scan rating, even though the cap fit was okay and the infant behaved well.

6 Conclusion and Outlook

6.1 Conclusion

In this thesis, a technical prototype was developed into a ready-to-use app that allows physical therapists to create scans of the heads of infants and it proved its capabilities in a real-life setting.
By using information extracted from markers on the cap that an infant is wearing during the scan in combination with depth data captured by the TrueDepth camera system, the app is able to create feedback both about the progress of a scan and the positioning of the iPhone with respect to the infant's head. This feedback is provided to the user by using haptic and audible feedback mechanisms, which allow the user to capture a high quality scan despite the inability to properly see the point cloud on the screen.
In a real-life therapy setting, physical therapists then tested this concept and its implementation. After every scan, they were questioned about the scan experience and its circumstances inside the app and after four weeks of testing, interviews were conducted to further explore the highlights and improvement factors. This showed that the positioning feedback about the distance from the head was very effective at helping all therapists to correct their position. Meanwhile, the positioning instructions on where to move the device in a two-dimensional way were only partially effective, as they were sometimes conceived as difficult to realize and took more practice.
Overall an easy-to-use interface was achieved by focusing on the principle of only showing elements on the screen which are relevant to the user at this particular moment and following a coherent design throughout the app. This, however, could only be tested by the physical therapists for the parts of the app that are necessary to create a scan due to their tight schedule. The scan process and user experience were perceived as intuitive and therefore easy to use which was a key objective of this work.

6.2 Outlook

For a future version, the next step is to improve the cap and extend the app with the functionality to calculate the CVAI on-device. With an improved cap, the data collection becomes easier and the infants might be more comfortable wearing it. The collected data is used to further train an AI.

This data could first be used to automatically detect if the ears and the nose are scanned which will improve the feedback. The app, with improved feedback, the ability to reliably calculate the CVAI combined with an improved cap would allow parents to use it.

Afterwards, once enough data is collected, the trained AI can be used to allow a future version of the app to take scans without needing the child to wear a cap with markers on it. This version would mark the goal of the research project "InferMod3D", which is to develop an app that can be used by both parents and physical therapists to scan an infant's head and calculate the CVAI with just an iPhone.

Bibliography

[1] Kharchyshyn Andrew et al. *Metal Tutorial: Getting Started | raywenderlich.com*. Oct. 2018. URL: `https://www.raywenderlich.com/7475-metal-tutorial-getting-started` (visited on 05/31/2022).

[2] AppleInsider. *Face ID | iPhone, iPad, Masks*. URL: `https://appleinsider.com/inside/face-id` (visited on 04/01/2022).

[3] Myagkov Artem, Feicho Eduard, and Nicholson Steve. *OpenCV: Installation in iOS*. URL: `https://docs.opencv.org/4.x/d5/da3/tutorial_ios_install.html` (visited on 04/01/2022).

[4] Inés Barbero-García, José Luis Lerma, and Gaspar Mora-Navarro. "Fully automatic smartphone-based photogrammetric 3D modelling of infant's heads for cranial deformation analysis". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 166 (Aug. 2020), pp. 268–277. ISSN: 0924-2716. DOI: `10.1016/J.ISPRSJPRS.2020.06.013`. (Visited on 03/28/2022).

[5] Andreas Breitbarth et al. "Measurement accuracy and dependence on external influences of the iPhone X TrueDepth sensor". In: (Sept. 2019), p. 7. ISSN: 1996756X. DOI: `10.1117/12.2530544`. (Visited on 04/01/2022).

[6] Pinckney Donald. *Metal 3D Graphics Part 1: Basic Rendering | Donald Pinckney*. July 2018. URL: `https://donaldpinckney.com/metal/2018/07/05/metal-intro-1.html` (visited on 05/31/2022).

[7] Apple Inc. *Automatic Reference Counting — The Swift Programming Language (Swift 5.6)*. URL: `https://docs.swift.org/swift-book/LanguageGuide/AutomaticReferenceCounting.html` (visited on 04/27/2022).

[8] Apple Inc. *AVAudioSession | Apple Developer Documentation*. URL: `https://developer.apple.com/documentation/avfaudio/avaudiosession` (visited on 04/25/2022).

[9] Apple Inc. *AVFoundation Overview - Apple Developer*. URL: `https://developer.apple.com/av-foundation/` (visited on 06/20/2022).

[10] Apple Inc. *AVSpeechSynthesisVoice | Apple Developer Documentation*. URL: `https://developer.apple.com/documentation/avfaudio/avspeechsynthesisvoice` (visited on 06/27/2022).

[11] Apple Inc. *AVSpeechSynthesizer | Apple Developer Documentation*. URL: `https://developer.apple.com/documentation/avfaudio/avspeechsynthesizer` (visited on 06/14/2022).

[12] Apple Inc. *AVSpeechUtterance | Apple Developer Documentation*. URL: `https://developer.apple.com/documentation/avfaudio/avspeechutterance` (visited on 06/27/2022).

[13] Apple Inc. *body | Apple Developer Documentation*. URL: `https://developer.apple.com/documentation/SwiftUI/View/body-swift.property` (visited on 04/11/2022).

[14] Apple Inc. *CHHapticEngine | Apple Developer Documentation*. URL: `https://developer.apple.com/documentation/corehaptics/chhapticengine` (visited on 04/25/2022).

[15] Apple Inc. *Core Haptics | Apple Developer Documentation*. URL: `https://developer.apple.com/documentation/corehaptics` (visited on 06/27/2022).

[16] Apple Inc. *dismiss()*. URL: `https://developer.apple.com/documentation/swiftui/presentationmode/dismiss()` (visited on 06/23/2022).

[17] Apple Inc. *fullScreenCover(isPresented:onDismiss:content:) | Apple Developer Documentation*. URL: `https://developer.apple.com/documentation/swiftui/form/fullscreencover(ispresented:ondismiss:content:)` (visited on 06/23/2022).

[18] Apple Inc. *Input and Event Modifiers | Apple Developer Documentation*. URL: `https://developer.apple.com/documentation/swiftui/view-input-and-events` (visited on 04/11/2022).

[19] Apple Inc. *Managing Model Data in Your App*. URL: `https://developer.apple.com/documentation/swiftui/managing-model-data-in-your-app` (visited on 04/11/2022).

[20] Apple Inc. *Metal Enhancements for A13 Bionic - Tech Talks - Videos - Apple Developer*. URL: `https://developer.apple.com/videos/play/tech-talks/608/` (visited on 03/14/2022).

[21] Apple Inc. *Metal Overview - Apple Developer*. URL: `https://developer.apple.com/metal/` (visited on 06/15/2022).

[22] Apple Inc. *NavigationView | Apple Developer Documentation*. URL: https://developer.apple.com/documentation/swiftui/navigationview (visited on 04/11/2022).

[23] Apple Inc. *PresentationMode*. URL: https://developer.apple.com/documentation/swiftui/presentationmode (visited on 06/23/2022).

[24] Apple Inc. *ProcessInfo.ThermalState | Apple Developer Documentation*. URL: https://developer.apple.com/documentation/foundation/processinfo/thermalstate (visited on 06/06/2022).

[25] Apple Inc. *sheet(isPresented:onDismiss:content:) | Apple Developer Documentation*. URL: https://developer.apple.com/documentation/SwiftUI/View/sheet(isPresented:onDismiss:content:) (visited on 06/27/2022).

[26] Apple Inc. *State | Apple Developer Documentation*. URL: https://developer.apple.com/documentation/swiftui/state (visited on 04/11/2022).

[27] Apple Inc. *Streaming Depth Data from the TrueDepth Camera | Apple Developer Documentation*. URL: https://developer.apple.com/documentation/avfoundation/cameras_and_media_capture/streaming_depth_data_from_the_truedepth_camera (visited on 03/11/2022).

[28] Apple Inc. *SwiftUI Overview - Xcode - Apple Developer*. URL: https://developer.apple.com/xcode/swiftui/ (visited on 06/13/2022).

[29] Apple Inc. *UIKit | Apple Developer Documentation*. URL: https://developer.apple.com/documentation/uikit (visited on 06/27/2022).

[30] Apple Inc. *UIViewControllerRepresentable | Apple Developer Documentation*. URL: https://developer.apple.com/documentation/swiftui/uiviewcontrollerrepresentable (visited on 06/27/2022).

[31] Apple Inc. *UIViewRepresentable | Apple Developer Documentation*. URL: https://developer.apple.com/documentation/swiftui/uiviewrepresentable (visited on 06/27/2022).

[32] Apple Inc. *Updating Continuous and Transient Haptic Parameters in Real Time | Apple Developer Documentation*. URL: https://developer.apple.com/documentation/corehaptics/updating_continuous_and_transient_haptic_parameters_in_real_time (visited on 03/29/2022).

[33] Apple Inc. *Using a Render Pipeline to Render Primitives | Apple Developer Documentation*. URL: https://developer.apple.com/documentation/metal/using_a_render_pipeline_to_render_primitives (visited on 05/31/2022).

[34] Apple Inc. *View | Apple Developer Documentation*. URL: https://developer.apple.com/documentation/swiftui/view (visited on 06/27/2022).

[35] Apple Inc. *Xcode 14 Overview - Apple Developer*. URL: https://developer.apple.com/xcode/ (visited on 06/27/2022).

[36] Bok Ki Jung and In Sik Yun. "Diagnosis and treatment of positional plagiocephaly". In: *Archives of Craniofacial Surgery* 21 (2 2020), p. 80. ISSN: 22875603. DOI: 10.7181/ACFS.2020.00059. URL: /labs/pmc/articles/PMC7206465/%20/labs/pmc/articles/PMC7206465/?report=abstract%20https://www.ncbi.nlm.nih.gov/labs/pmc/articles/PMC7206465/ (visited on 03/11/2022).

[37] Chamary JV. *How Face ID Works On iPhone X*. Sept. 2017. URL: https://www.forbes.com/sites/jvchamary/2017/09/16/how-face-id-works-apple-iphone-x/?sh=474d66cc624d (visited on 04/01/2022).

[38] OpenCV. *About - OpenCV*. URL: https://opencv.org/about/ (visited on 04/13/2022).

[39] Hudson Paul. *All SwiftUI property wrappers explained and compared - a free SwiftUI by Example tutorial*. 2021. URL: https://www.hackingwithswift.com/quick-start/swiftui/all-swiftui-property-wrappers-explained-and-compared (visited on 04/11/2022).

[40] Fabian Rapp, Samuel Zeitvogel, and Christian Wernet. *ISRG / InferMod3D / InferMod3D-Recorder · GitLab*. May 2022. URL: https://iz-gitlab-01.hs-karlsruhe.de/isrg/infermod3d/infermod3d-recorder/-/tree/master (visited on 06/20/2022).

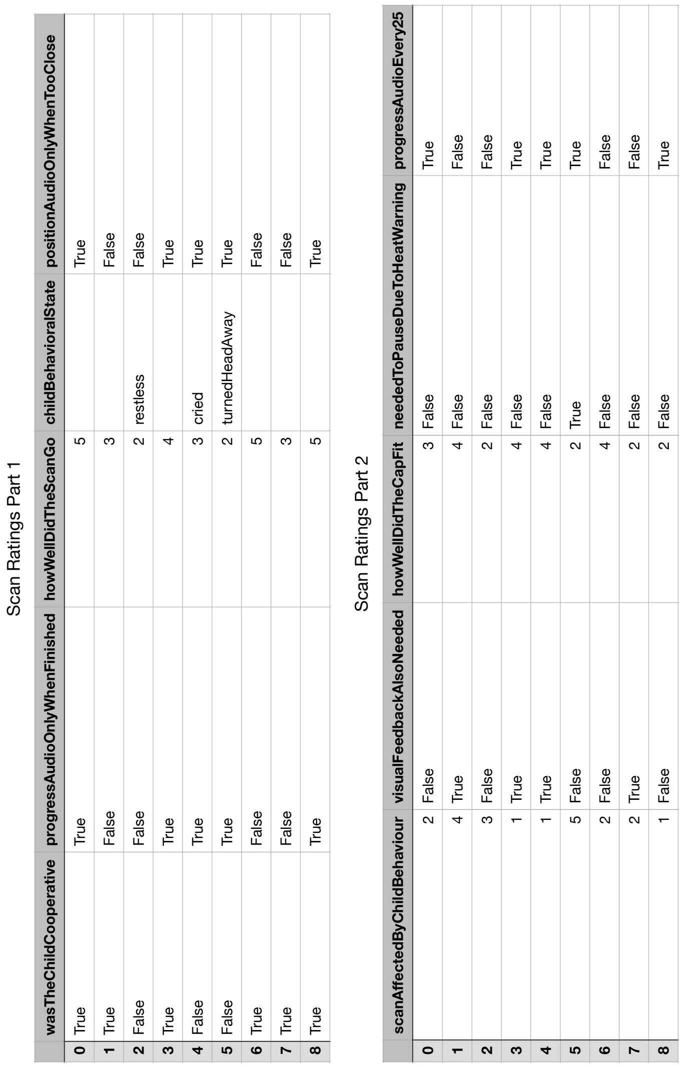
[41] Varilag. *Plagiocephalus: Ursachen, Spätfolgen& Behandlung von Plagiozephalie*. 2022. URL: https://www.varilag.de/ratgeber/plagiocephalus/ (visited on 03/15/2022).

[42] Maximilian Vogt, Adrian Rips, and Claus Emmelmann. "Comparison of iPad Pro®'s LiDAR and TrueDepth Capabilities with an Industrial 3D Scanning Solution". In: *Technologies* 9 (2 Apr. 2021), p. 25. DOI: 10.3390/TECHNOLOGIES9020025. (Visited on 04/01/2022).

[43] Samuel Zeitvogel et al. "RGBD Infant Head Reconstruction for Cranial Vault Asymmetry Estimation". In: *IEEE Access* 10 (2022), pp. 36208–36219. ISSN: 21693536. DOI: 10.1109/ACCESS.2022.3160749. (Visited on 05/30/2022).

Attachments

1 Rating data collected through the ScanRatingView



Scan Ratings Part 1

	wasTheChildCooperative	progressAudioOnlyWhenFinished	howWellDidTheScanGo	childBehavioralState	positionAudioOnlyWhenTooClose
0	True	True	5		True
1	True	False	3		False
2	False	False	2	restless	False
3	True	True	4		True
4	False	True	3	cried	True
5	False	True	2	turnedHeadAway	True
6	True	False	5		False
7	True	False	3		False
8	True	True	5		True

Scan Ratings Part 2

	scanAffectedByChildBehaviour	visualFeedbackAlsoNeeded	howWellDidTheCapFit	neededToPauseDueToHeatWarning	progressAudioEvery25
0	2	False	3	False	True
1	4	True	4	False	False
2	3	False	2	False	False
3	1	True	4	False	True
4	1	True	4	False	True
5	5	False	2	True	True
6	2	False	4	False	False
7	2	True	2	False	False
8	1	False	2	False	True

Figure 1: All the scan ratings from the quantitative testing in the ScanRatingView.

2 Notes from the physical therapist interview 1

Anzahl erstellter Scans: 2
Positionierungsfeedback:

- Audio “zu nah” war sehr hilfreich und nützlich
- Positionierungsinstruktionen waren schwierig zu befolgen

Fortschrittsfeedback:

- Audio “fertig” war manchmal verwirrend, da der Scan gefühlsmäßig noch nicht fertig war
- Nach dem Scan Rückmeldung wie gut der Scan war (sind alle Körperteile aufgenommen worden?) wäre hilfreich, um gegenenfalls aus Fehlern zu lernen
- Audio “50” wäre hilfreich gewesen

Eltern: Waren sehr interessiert
Mütze: Das Aufziehen der Mütze hat Kindern nicht gefallen
Sonstiges: Keine Zeit gehabt, um die App zu erforschen
Verbesserungsfeedback:

- Anstatt Postionierungsinstruktionen “links-oben”, “rechts”, … ansagen welcher Teil des Kopfes noch nicht passt
- PatientMetaDataView: Diagonale A und B des Kopfes verwechselt, trotz Bild mit Beschriftung. Diagonale A: links vorne nach rechts hinten, wäre besser
- Erinnerung am Ende des Scans die Ohren abzufilmen

3 Notes from the physical therapist interview 2

Positionierungsfeedback:

- Audio “zu nah” war sehr gut
- Manchmal gab es das Gefühl als würde das Gerät ständig vibrieren
- Positionierungsangaben waren unintuitiv

Fortschrittsfeedback:

- Fortschrittsangabe war auch für die Eltern sehr hilfreich und motivierend
- Bevorzugte Inkremente: Audioausgabe bei 50% und bei 100%

Eltern:

- Geduld der Eltern ca. 1:30 - 2:00 Minuten. Wurde weniger, wenn Kind unruhig wurde
- Waren sehr motiviert

Mütze: Die Mütze war das Problem, die Sticker verschmieren und lösen sich
Sonstiges:

- Kam ohne Bildschirm besser zurecht als auf den Bildschirm zu schauen
- Viel besser als erwartet
- App war allgemein intuitiv
- Zeit ist das Problem, vor allem, wenn bei einem Scan noch Annotationen gesetzt werden müssen

Verbesserungsfeedback:

- Positionierungsinstruktionen nicht wohin das iPhone bewegt werden soll, sondern wohin der Kopf bewegt werden muss
- Hinweis am Ende des Scans, ob die Ohren und die Nase richtig im Scan sind

4 Notes from the physical therapist interview 2

Anzahl erstellter Scans: 1
Positionierungsfeedback:

- Positionierungsangaben waren intuitiv

Fortschrittsfeedback:

- Zwischendruch wurde auf den Fortschrittsbalken geschaut
- Audio “50” wäre hilfreich gewesen

Kind: Das Kind hat dem iPhone hinterhergeschaut. Wenn Eltern das Kind abgelenkt haben, war das kein Problem mehr
Eltern:

- Waren aufgeschlossen
- Von keinem kam die Frage was jetzt das Resultat des Scans ist

Mütze: Die Mütze war zu klein, ging kaum zu
Sonstiges:

- Der Scan ging sehr schnell, wenig Probleme
- App ist gut erklärt, gut verständlich

Verbesserungsfeedback: -

5 Notes from the physical therapist interview 2

Anzahl erstellter Scans: ca. 5
Positionierungsfeedback:

- Ganz wenige Probleme
- Musste wenig bis gar nicht auf den Bildschirm schauen
- Audio "zu nah" war sehr gut. Hat bei den ersten Scans sehr geholfen den richtigen Abstand zu finden
- Vibrationen waren sehr gut
- Positionierungsangaben waren schwierig, ein wenig unintuitiv
- Einfärben der Marker im Scan war klasse

Fortschrittsfeedback:

- 100
- 50
- Kleinere Inkremente wären besser gewesen (alle 25

Kind: Vibrationen oder Audioausgaben haben kein Kind beeinflusst. Nur das leuchtende Display hat Kinder abgelenkt
Eltern:

- Waren durchweg positiv gestimmt
- Sehr aufgeschlossen, begeistert
- Haben die Nutzung von den Bildern erfragt. Datenschutz wichtig
- Mütze war für Eltern eine Hürde. Die App alleine war kein Problem
- Messdaten wie CVAI wären für die Eltern interessant
- Eltern wollten die Messung nicht sehen

Mütze:

- Sehr schlecht
- Mütze aufziehen war für Kinder schlimm
- Aufkleber haben nicht gut gehalten
- Mütze war zu klein
- Schlechter Scan lag an der Mütze

Sonstiges:

- Scan war in früherer Version sehr lange. In dieser Version ging der Scan sehr schnell
- Zusätzlich Bewertungen nach dem Scan zu geben, war zeitlich herausfordernd
- Scan in der Behandlung unterzubringen, wäre gar kein Problem

Verbesserungsfeedback: -