HTL - Perg
Höhere Abteilung für Informatik

Diplomarbeit

Altem Recognition

Projektteam: Hofer Benedikt
Schmid Jakob
Haider Florian
Projektbetreuer: Ing. Dominik Raffetseder, MSc

In Zusammenarbeit mit Fa. ITPRO - Consulting & Software GmbH
Betreuer Herr Duschlbauer Klemens, MSc

Bearbeitungszeitraum: 01.11.2023 – 04.04.2024

1. Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Perg, 2. April 2024 Unterschrift _____

Hofer Benedikt

Perg, 2. April 2024 Unterschrift _____

Schmid Jakob

Perg, 2. April 2024 Unterschrift _____

Haider Florian

Benedikt Hofer

Digital unterschrieben von Benedikt Hofer
Datum: 2024.04.02 22:01:30 +02'00'

Signiert von:	Jakob Simon Schmid
Datum:	02.04.2024 21:48:10

Dieses Dokument ist digital signiert!

Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Prüfinformation: Informationen zur Prüfung der elektronischen Signatur finden Sie unter: www.a-trust.at/pdf

Florian Haider

Digital unterschrieben von Florian Haider
Datum: 2024.04.03 08:26:49 +02'00'

2. Gendererklärung

In der vorliegenden Arbeit wird darauf verzichtet, bei Personenbezeichnungen sowohl die männliche als auch die weibliche Form zu nennen. Es wird auf das generische Maskulinum zurückgegriffen, um alle Geschlechter gleichermaßen anzusprechen.

Dies soll jedoch keinesfalls eine Geschlechterdiskriminierung oder eine Verletzung des Gleichheitsgrundsatzes zum Ausdruck bringen.

Perg, 2. April 2024 Unterschrift Benedikt Hofer

Hofer Benedikt

Perg, 2. April 2024 Unterschrift

Schmid Jakob

Perg, 2. April 2024 Unterschrift Florian Haider

Haider Florian

3. Danksagung

An dieser Stelle möchten wir uns bei jenen Personen bedanken, die uns im Verlauf der Erstellung der Diplomarbeit in jeglicher Hinsicht unterstützt haben.

Ein besonderer Dank gilt unserer Betreuungslehrkraft, Herrn Professor Ing. Dominik Raffetseder, der uns bei der Entwicklung mit hilfreichen Tipps stets zur Seite gestanden ist.

Ein weiteres Danke geht an unsere Ansprechpartner Klemens Duschlbauer MSc. Im Laufe der Erstellung unseres Produktes, unterstützte er uns bei Problemen und trug ebenfalls maßgeblich zur erfolgreichen Erstellung unserer Diplomarbeit bei. Besonders möchten wir uns ebenfalls bei Delia Dorn bedanken, welche die Organisation und Leitung der regelmäßigen Meetings übernommen hat.

Vielen Dank auch an Familie und Freunde, die während dieser stressigen Zeit Rücksicht auf uns genommen und uns von der einen oder anderen Arbeit befreit haben. Danke sagen möchten wir auch Jonas Schmid, welcher durch die Bereitstellung seines Android Handys, die Entwicklung vereinfachte.

4. Impressum

Schule
HTBLA Perg für Informatik
Machlandstraße 48
4320 Perg

Schuljahr
2023/24

Klasse
5BHIF

Projekttitel
AItem Recognition

Projektteam

Benedikt Hofer

Jakob Schmid

Florian Haider

Betreuungslehrer
Prof. Ing. Dominik Raffetseder, MSc

Auftraggeber
ITPRO - Consulting & Software GmbH

5. Kurzfassung

Aufgabenstellung

Die Aufgabe besteht darin, in Zusammenarbeit mit dem Unternehmen ITPRO eine Erweiterung für das Schulprojekt "Project AR" zu entwickeln. Das Ziel von „Project AR“ ist es, die Kommunikation zwischen ITPRO-Monteuren und Technikern zu verbessern, indem visuelle Hilfestellungen während Montagearbeiten bereitgestellt werden. Das Projekt „AItem Recognition“ soll es den Monteuren ermöglichen, Maschinenteile automatisch in Echtzeit zu erkennen und relevante Informationen einzublenden, um ihre Tätigkeiten eigenständig ausführen zu können. Darüber hinaus soll die Lösung auch bei fehlender Verfügbarkeit eines Technikers Unterstützung bieten.

Realisierung

Die Realisierung erfolgt durch die Entwicklung einer Android Augmented-Reality-App mithilfe von Unity und der „Model Targets“ Technologie von Vuforia. Unsere Applikation erkennt Maschinenteile automatisch in der Kameraaufnahme des Monteurs und blendet relevante Textinformationen ein. Parallel dazu wird eine .NET CORE REST-API entwickelt und als App-Service auf der Plattform Azure[1] erreichbar gemacht, um die Kommunikation zwischen der Android-App und dem Backend-Services zu ermöglichen. Zusätzlich wird eine weitere .NET CORE REST-API in einem Docker-Container auf Azure bereitgestellt, die die Funktionalität des Generierens eines 3D-Modells aus Fotos zur Verfügung stellt. Dies bietet in weiterer Folge innerhalb der App die Möglichkeit, Objekte zu scannen, mit Titeln und Beschreibungen zu versehen und hochzuladen.

Ergebnis

Das Ergebnis ist eine voll funktionsfähige Android AR-App, die Montagearbeiten durch automatische Objekterkennung und Einblendung von hilfreichen Informationen unterstützt. Die App läuft reibungslos auf Zebra-Geräten und wurde von uns als APK-Datei bereitgestellt. Die .NET CORE REST-APIs ermöglichen eine nahtlose Kommunikation zwischen der App und den Backend-Services auf Azure, was die Erweiterung der Funktionalität sowie das Hochladen und Verwalten von Objekten ermöglicht. Das Projekt bietet eine effektive Lösung, um die Effizienz der Montagearbeiten zu verbessern und Techniker zu entlasten.

Nach einer kurzen Einschulung und finalen Erklärung überging das System nach Abschluss des Projekts in den Produktivbetrieb durch ITPRO.

[1] [MICROSOFT]

6. Abstract

In collaboration with the company ITPRO, this project aimed to extend the existing “Project AR”, which was developed by us in a school project, by developing an extension called “AItem Recognition”. The primary objective of the school project is to support communication between ITPRO technicians and installers. This is done by placing visual indicators inside of an Augmented Reality environment. The extension “AItem Recognition” empowers workers with the ability to automatically identify machine parts in real-time and access relevant information to speed up the time it takes to complete tasks. The main reason this system was developed, is so help can be offered even in absence of an ITPRO technician.

The implementation phase involved the enlargement of the already existing mobile application. A backend system was developed and deployed on a cloud platform to facilitate seamless communication between the mobile application and backend services. Additionally, another component was deployed to enable the generation of 3D models from photos, allowing users to scan an object, annotate it with a title and a description and finally upload them within the application. Afterwards those objects must be trained by the “Vuforia Target Model API” to be able to recognize them afterwards.

The outcome of the project is a fully functional mobile application that supports assembly tasks through automatic object recognition and information overlay. The backend system ensures seamless communication between the application and backend services, facilitating functionality extension and object management. From an implementation perspective, the application comprises a mobile app for field workers and a backend consisting of multiple components handling general business logic, data storage, and additional functionalities. Additionally, a web application simplifies management tasks.

7. Inhaltsverzeichnis

8. Einleitung

8.1. Motivation

8.1.1. Geschäftlicher Hintergrund

Der Hintergrund unserer Diplomarbeit ist geprägt von dem wachsenden Bedarf an effizienten Lösungen. Besonders im Bereich von Montagearbeiten und technischen Unterstützungen. Unser Kooperationspartner ITPRO[2], hat sich auf Software- und Techniklösungen spezialisiert und bietet ebenfalls Außendienst Lösungen an. Um die Kommunikation zwischen einem Monteur und einem Techniker im Büro, welcher bei Komplikationen den Mitarbeiter im Außendienst unterstützen kann, zu verbessern, haben wir im Rahmen des Schulunterrichts ein Projekt mit dem Titel „Project-AR“ entwickelt. Dieses wird durch unsere Diplomarbeit dahingehen erweitert, dass falls bei Unklarheiten kein Techniker im Büro verfügbar ist, anderweitig Installationsschritte oder ähnliche Informationen zum richtigen Teil erhalten werden können. Durch die Nutzung von Cloud-Technologien ist es uns möglich eine flexible Lösung für unser Vorhaben entwickeln zu können.

8.1.2. Projektauslöser

Wie bereits im obigen Kapitel Geschäftlicher Hintergrund erwähnt, treten bei der Installation von Hardwarekomponenten durch unseren Kooperationspartner oft Komplikationen auf. Hierbei ist als Beispiel ein GPS-System für LKWs zu nennen. Um das System zu installieren, müssen Monteure vor Ort die nötigen Komponenten einbauen. Falls Probleme auftreten, wird ein Techniker im Büro von ITPRO kontaktiert, welcher versucht den Mitarbeiter zu helfen. Unser Schulprojekt „Project-AR“ können dadurch visuelle Schritte innerhalb einer Augmented Reality Umgebung platziert, um somit Komplikationen zu beseitigen.

Da sich nicht zu jeder Zeit jemand, der auch die nötigen Schritte zur Installation der Komponente weiß, erweitert unsere Diplomarbeit „AItem Recognition“ diese Monteurunterstützungs-App dahingehend, dass ein weiterer Modus hinzugefügt wird, welcher unsere Diplomarbeit umfasst.

8.2. Zielsetzung

8.2.1. Geschäftsziel

Das Geschäftsziel unserer Diplomarbeit besteht darin, die Effizienz und Produktivität der Montagearbeiten von ITPRO-Monteuren zu verbessern und gleichzeitig die Qualität der technischen Unterstützung zu erhöhen. Dadurch soll die Kommunikation zwischen den Mitarbeitern vor Ort und den technischen Experten optimiert werden. Außerdem soll eine umfassende Unterstützung,

[2] siehe Kapitel 8.4.2

während der Montageprozesse geboten werden in Form von automatischer Objekterkennung und die Bereitstellung relevanter Informationen in Echtzeit. Durch die Realisierung dieses Geschäftsziels ist es möglich eine signifikante Zeiteinsparung und somit geringere Montagekosten zu erreichen.

8.2.2. Projektziel

Das Ziel des Projekts ist es, den Monteuren auch Hilfe zu bieten, wenn kein Techniker zur Verfügung steht indem bereits gespeicherte Objekte automatisch erkannt und die dazugehörigen Beschreibungen eingeblendet werden. Somit sollen zum Beispiel Maschinenteile automatisch erkannt und relevante Informationen darüber angezeigt werden. Die App soll eine nahtlose Integration von visuellen Hilfestellungen in den Arbeitsprozess der Monteure bieten und gleichzeitig die Notwendigkeit einer kontinuierlichen, direkten technischen Unterstützung reduzieren. Zusätzlich soll dieses Projekt die Techniker entlasten, da die Monteure sich im ersten Schritt selbst helfen können. Außerdem können Monteure auch selbst Objekte einscannen, mit einem Titel und einer Beschreibung versehen und dann hochladen.

8.2.3. Projektvorgehensziele

Die Projektvorgehensziele unserer von „AItem Recognition“ umfassen mehrere Aspekte, um eine erfolgreiche Umsetzung der Anforderungen sicherzustellen. Zu den wichtigsten Zielen gehören:

(1) Definition der Anforderungen
Klare Definition der Anforderungen an die Applikation und der Backend-Services, basierend auf den Zielen von ITPRO

(2) Technologie Auswahl
Auswahl der geeigneten Technologien für die Generierung von 3D Modellen und der Backend-Services. Die Technologie für die Applikation, ist durch den Fakt, dass unser bestehendes „Project-AR“-Projekt erweitert wird, bereits festgelegt.

(3) Erweiterung der Applikation (Scannen)
Erweitern der Applikation durch einen Modus, in dem Bilder aufgenommen werden können. In weiterer Folge werden ein Titel und Beschreibung festgelegt und an unsere Online-Schnittstelle übertragen.

(4) Erweiterung der Applikation (Erkennen)
Erweiterung um einen Modus, in dem die Datensätze aus der Rest-API geholt werden, und bei der Erkennung eines Objekts, die dazugehörigen Informationen angezeigt werden.

(5) Implementierung der Backend-Services
Implementierung der notwendigen Rest-APIs, um die Kommunikation zwischen Web- und Android-App und Datenbank zu ermöglichen.

(6) Testung und Qualitätssicherung
Umfassendes Testung der Applikation und der Rest APIs, um sicherzustellen, dass sie den Anforderungen entsprechen und fehlerfrei funktionieren.

8.3. Projektinhalt – Überblick

8.3.1. 3D-Modelle einscannen

Durch das Scannen von Objekten, ist es möglich die reale dreidimensionale Struktur, in digitaler Form zu erfassen. Die Technologie bietet eine Vielzahl von Anwendungen in verschiedenen Bereichen, darunter Fertigung, Design, Architektur oder auch in der Medizin. Durch das 3D-Scannen können detaillierte und präzise Modelle von physischen Objekten erstellt werden, die anschließend für eine Vielzahl von Zwecken verwendet werden können, wie beispielsweise die virtuelle Rekonstruktion historischer Artefakte, die Qualitätskontrolle in der Fertigung oder die Anpassung von medizinischen Implantaten.

Im Rahmen unserer Diplomarbeit wird das 3D-Scannen als zentrales Element zur automatischen Objekterkennung und -integration in die Augmented-Reality-Anwendung eingesetzt. Die Nutzung von 3D-Scantechnologien ermöglicht es in Nachhinein präzise und zuverlässig Objekte erkennen zu können, was wiederum die Effektivität und Effizienz der Montagearbeiten verbessert. Zudem wird durch die Möglichkeit, Objekte selbst zu scannen, diese anschließend zu betiteln und hochzuladen, eine flexible und erweiterbare Lösung geschaffen, die den Anforderungen der Diplomarbeit gerecht wird.

8.3.2. Webapplikation für Administratoren

Die Webapplikation für Administratoren ermöglicht es Model-Targets anzuzeigen, zu bearbeiten und zu löschen. Dadurch können alle Model-Targets auf einfache Art und Weise verwaltet, ergänzt und bei Bedarf aus dem System gelöscht werden.

8.3.3. Erkennen der Objekte

In dem Thema geht es hauptsächlich um die Erkennung der Objekte. Die App ist eine Android Augmented Reality App, die Objekte in der realen Welt platzieren kann. Bevor der Installateur ein Objekt erkennen kann, werden ihm alle Model Targets aufgelistet, die in der Vuforia API gespeichert sind. Bei der Auflistung werden auch Modelle aufgelistet, die gerade von der Vuforia API trainiert werden oder Fehler beinhalten. Diese Modelle werden nur angezeigt, können aber nicht ausgewählt werden.

Der Installateur kann das gewünschte Objekt auswählen, dass er erkennen möchte. Wird das Objekt erkannt, so werden erscheinen Informationen oder Arbeitsschritte neben dem Objekt. Das Ziel des Projektabschnitts dieses Kapitel ist es, eine zusätzliche Unterstützung für den Installateur anzubieten.

8.4. Projektumfeld

8.4.1. Projektteam

Unser Team besteht aus 3 Schülern der HTL Perg – Abteilung für Höhere Informatik.

Hofer Benedikt

Projektleiter

Schmid Jakob

Projektmitglied

Haider Florian

Projektmitglied

8.4.2. Auftraggeber

Unser Auftraggeber ist ITPRO Consulting & Software GmbH, welcher sein Hauptbüro in Linz. Zusätzlich befindet sich noch ein weiterer Standort im Softwarepark in Hagenberg im Mühlkreis. Gegründet wurde das lösungsorientierte Unternehmen 1999 und hat sich auf die Softwareentwicklungsbranche spezialisiert. Ihr Spezialgebiet umfasst sowohl App- und Web-Entwicklung als auch Außendienstlösungen, in welchem auch unsere Diplomarbeit eingesetzt wird.

IT | PRO
Software die's einfach macht.

Abbildung 1: Logo der Firma ITPRO
Quelle: [ITPRO CONSULTING & SOFTWARE GMBH]

Abbildung 2: Logo der Firma ITPRO
Quelle: [ITPRO CONSULTING & SOFTWARE GMBH]

8.4.3. Betreuung

Prof. Ing. Dominik Raffetseder, MSc
Kontakt: d.raffetseder@htl-perg.ac.at

Unsere Diplomarbeit wird von Herrn Professor Raffetseder betreut. Seit 2019 unterrichtet Professor Dominik Raffetseder an der HTL in Perg. Im heurigen Schuljahr unterrichtet er uns im Fach Netzwerktechnik und verteilte Systeme (Mobile Computing).

Professor Raffetseder begleitete uns die ersten 3 Klassen im Gegenstand Programmieren und Software Engineering Java. Durch seine guten Kenntnisse in Datenbanken, Mobile Computing und Projektentwicklung ist er uns immer eine große Hilfe bei der Planung und Realisierung unserer Diplomarbeit gewesen. Durch seine jahrelange Berufserfahrung stand er uns stets mit Rat und Tat zur Seite.

9. Theoretische und fachpraktische Grundlagen und Methoden

9.1. Verwendete Technologien

9.1.1. Unity

Unity ist eine Laufzeit- und Entwicklungsumgebung, womit plattformübergreifenden Spiele entwickelt werden. Neben 2D-und 3D-Spielen können auch Tools für Virtual Reality und Augmented Reality verwendet werden. Zusätzlich bietet Unity eine Reihe an Bibliotheken an, die man in einem Projekt benötigt. Wir verwenden Unity für das Entwickeln einer Android-App, die Augmented Reality Inhalte behandelt.

9.1.2. Vuforia

Vuforia ist eine AR-Plattform, die sich auf Unternehmen konzentriert. Diese Plattform beinhaltet mehrere Lösungen mit verschiedenen AR-Technologien. In unserem Projekt verwenden wir „Vuforia Engine“, speziell das sogenannte „Model targets“ Feature[3].

9.1.3. ASP.NET Core Web API

ASP.NET Core Web API ist ein Framework, das von Microsoft entwickelt wurde, um die Entwicklung von RESTful-Webdiensten unter Verwendung von ASP.NET Core zu erleichtern. Es ermöglicht Entwicklern, HTTP-basierte APIs zu erstellen, die von verschiedenen Clientanwendungen, einschließlich Webanwendungen, Mobilanwendungen und anderen Diensten, konsumiert werden können.

[3] [PTC INC.]

9.1.4. Microsoft SQL Server

Microsoft SQL Server ist ein relationales Datenbankmanagementsystem (RDBMS), das von Microsoft entwickelt wurde. Es ist eine umfassende Plattform für die Verwaltung und Analyse von Daten in Unternehmen jeder Größe und wird häufig in Unternehmen und Organisationen weltweit eingesetzt.

9.1.5. Angular

Angular ist ein Open-Source-Framework für die Entwicklung von Single-Page-Anwendungen (SPAs) und dynamischen Webanwendungen. Es wird von Google entwickelt und gewartet und bietet eine umfangreiche Sammlung von Werkzeugen und Funktionen für die Entwicklung moderner, ansprechender Webanwendungen.

9.1.6. Docker

Docker ist eine Open-Source-Plattform, die es Entwicklern ermöglicht, Anwendungen in sogenannten Containern zu erstellen, bereitzustellen und auszuführen. Container sind eine Art standardisierter, leichtgewichtiger, portabler und isolierter Umgebungen, die eine konsistente Ausführung von Anwendungen über verschiedene Rechner hinweg ermöglichen.

9.1.7. Android

Android ist ein mobiles Betriebssystem, das von Google entwickelt wurde. Es wird für Mobilgeräte wie Smartphones und Tablets entwickelt.

9.1.8. Fotogrammetrie Software

Um unser Vorhaben in die Wirklichkeit umsetzen zu können, ist es notwendig Objekte aus der Realität in ein virtuelles dreidimensionales Modell übertragen zu können. Somit ist es im Anschluss möglich, jene wieder zu erkennen. Diese Technik nennt man „Fotogrammetrie“.

9.1.8.1. Anforderungen an die Technologie

Da unsere Applikation diese Bilder an unsere Rest API sendet, welche dann die Verarbeitung übernimmt, ist es wichtig, dass die Software entweder durch die Kommandozeile oder durch eine Einbindung in den Programmcode ausführbar ist. Außerdem sollten die Kosten dessen gering sein, da es sich hierbei um eine Applikation für unseren Kooperationspartner handelt, welcher dafür aufkommen muss. Weiters existieren zahlreiche Fotogrammetrie Softwares, welche eine Grafikkarte zur Verarbeitung voraussetzen. In unserem Fall ist es essenziell, diese auch ohne eine GPU zu verwenden.

Zusammengefasst sind unsere Anforderungen an eine derartige Software folgende:

- Erstellung qualitativ hochwertiger 3D-Modelle
- Bereitstellung einer API oder Ausführung durch Programmcode / Kommandozeile
- keine / geringe Kosten
- Generierung nur mit CPU auch möglich

9.1.8.2. Verworfene Optionen

Bei der Technologie-Suche sind uns einige Alternativen zur ausgewählten Technologie untergekommen. In diesem Kapitel werden die verworfenen Optionen genannt und ihr Ausscheidungsgrund erläutert.

(1) 3DF Zephyr

2014 wurde die erste Version der Software von der Firma 3Dflow SRL entwickelt. Grundsätzlich ist an der Qualität der generierten Modelle nichts auszusetzen und das Programm generiert bereits bei wenigen Eingabebildern ein akzeptables Objekt.

Probleme

Für 3DF Zephyr existiert zwar eine Gratisversion, jedoch besitzt diese ein Limit von 50 Eingabebildern[4]. Außerdem ist diese Applikation nur durch ein User Interface bedienbar und kann somit nicht durch ein Programm oder ähnliches gestartet werden. Da der Source Code nicht frei zugänglich ist, ist es auch unmöglich es durch ein Programm starten zu lassen.

(2) COLMAP[5]

Colmap ist eine Open-Source-Software, welche ursprünglich von Johannes L. Schönberger im Rahmen seiner Doktorarbeit an der ETH Zürich, entwickelt wurde. 2013 wurde die erste Version der Bibliothek veröffentlicht. Da der Quellcode öffentlich zugänglich ist, wurde sie kontinuierlich weiterentwickelt und verbessert. Dadurch dass man den Programmcode einsehen kann, kommt diese auch für unseren Anwendungsfall in Frage. Außerdem ist es ebenfalls möglich das Programm, ohne einer Grafikkarte zu verwenden.

Probleme

Grundsätzlich sind hier alle von uns gesetzten Voraussetzungen getroffen. Jedoch haben wir uns gegen eine Verwendung von COLMAP entschieden, da die Größe der Community und Dokumentation hier etwas geringer ausfallen als bei unserer ausgewählten Technologie. Somit ist unsere Entscheidung auf eine Software gefallen, die sich bereits bewährt und viele gute Rezensionen hat.

9.1.8.3. AliceVision Meshroom (Ausgewählte Technologie)

AliceVision Meshroom ist eine Open-Source-Software die hauptsächlich für die Erstellung von 3D-Modellen aus Fotografien verwendet wird.[6] Das Programm kann durch Bildverarbeitungsalgorithmen und die Anwendung von Strukturrekonstruktionsverfahren hochwertige 3D-Objekte generieren. Es

[4] [3DFLOW SRL]
[5] [SCHOENBERGER, Johannes L.]
[6] vgl. [ALICEVISION]

basiert auf fortschrittlichen Techniken wie Structure-from-Motion (siehe 11.5.3.2 Nummer (5)) oder Multi-View Stereo (erwähnt in 11.5.3.4) um aus mehreren 2-dimensionalen Bilder ein dreidimensionales Modell zu erhalten. Grundsätzlich findet die Software in verschiedenen Bereichen Anwendung. Es erwies sich äußerst nützlich in der Architektur, Archäologie, Spieleentwicklung oder bei Filmproduktionen.

Meshroom ist aufgrund seiner Open-Source-Natur perfekt geeignet, da somit der gesamte Code zur Anwendung öffentlich zugänglich ist. Außerdem kann die Software auch über die Kommandozeile bedient werden, wodurch die Generierung an der Rest API gestartet werden kann.

9.2. Verwendete Entwicklungssysteme

9.2.1. Unity

Unity stellt einen Editor zur Verfügung, in dem man 2D- und 3D-Spiele, Simulationen, interaktive Anwendungen und andere multimediale Projekte erstellen kann. Der Unity-Editor bietet eine umfassende Arbeitsumgebung, die Entwicklern verschiedene Werkzeuge und Funktionen bietet, um Spiele und Anwendungen zu entwerfen, zu entwickeln, zu testen und zu veröffentlichen.

9.2.2. Visual Studio

Unity verwendet C# als Skriptsprache. Daher greifen wir auf Visual Studio für die Codebearbeitung zurück. Visual Studio 2022 ist die neueste Version der integrierten Entwicklungsumgebung (IDE) von Microsoft, die speziell für die Entwicklung von Softwareanwendungen entwickelt wurde. Insbesondere in Bezug auf Unity, einer beliebten plattformübergreifenden Spielentwicklungsplattform, gibt es eine enge Integration zwischen Visual Studio 2022 und Unity.

9.3. Verwendete Bibliotheken und Plug-Ins

9.3.1. Entity Framework Core

Entity Framework Core ist ein Object-Relational Mapping (ORM)-Framework von Microsoft, das die Entwicklung von Datenzugriffsschichten in .NET-Anwendungen vereinfacht.

9.3.2. System.IdentityModel.Tokens.Jwt

System.IdentityModel.Tokens.Jwt ist eine Bibliothek in der .NET-Plattform, die es ermöglicht, JSON Web Tokens (JWTs) zu erstellen, zu validieren und zu manipulieren.[7] JWTs sind ein offener Standard (RFC 7519), der eine kompakte Möglichkeit bietet, Informationen zwischen Parteien als JSON-Objekte sicher zu übertragen. Sie werden häufig in modernen Authentifizierungs- und Autorisierungssystemen verwendet, insbesondere in Webanwendungen und APIs.

[7] [MICROSOFT]

9.3.3. Vuforia

Das “Model targets” feature kann direkt in Unity angesteuert werden. Dazu muss es in Form eines sogenannten „unitypackage“ importiert werden.

9.3.4. Bootstrap

Bootstrap ist ein Open-Source-Framework für die Frontend-Entwicklung von Websites und Webanwendungen. Es wurde von Twitter entwickelt und bietet eine Sammlung von vorgefertigten HTML-, CSS- und JavaScript-Komponenten sowie eine umfangreiche Bibliothek von Designvorlagen und -elementen.

10. Planung und Realisierung

10.1. Projektorganisation

Unser Team besteht aus 3 Schüler der HTL Perg, wobei jede Person zu einem bestimmten Teilbereich des Projektes zugeteilt ist. Jede Woche halten wir ein internes Meeting in der Schule ab, dabei wird besprochen, was jeder einzelne geschafft hat, ob es Probleme gab und was wir bis nächste Woche erledigen werden.
Nach zwei Wochen zeigen wir unseren Fortschritt dem Auftraggeber. Das Meeting wird über Teams abgehalten, es wird der aktuelle Stand des Projektes gezeigt und über zukünftige Themen gesprochen.
Unsere Planung wird über das Online-Tool „Flyingdonut“[8] verwaltet.

10.2. Meilensteine

Grundsätzlich sind für jeden Bereich unserer Diplomarbeit eigene Meilensteine vorhanden. Das bedeutet, dass jedes Projektmitglied einen eigenen Zeitplan verfolgt. Die Zeitpläne sind aufeinander abgestimmt, sodass keine Wartezeiten durch Abhängigkeiten entstehen. Auf der nachfolgenden Seite sind die Meilensteine der jeweiligen Abschnitte aufgelistet.

[8] [FLYING DONUT SOFTWARE P.C.]

10.2.1. Object Recognition

- **Start**
  8. Juli 2023
- **Auswahl der Augmented Reality Technologie**
  15. Juli 2023
- **Anzeigen der erkannten Objekte (Statisch)**
  15. August 2023
- **Einblenden des Textes (Statisch)**
  1. September 2023
- **Erkennung trainierter 3D-Modelle (dynamisch)**
  4. Oktober 2023
- **Einblenden der Beschreibung (dynamisch)**
  10. Oktober 2023
- **Anzeigen verfügbarer Model Targets**
  1. November 2023
- **Aktualisieren der zu erkennenden Objekten**
  22. November 2023
- **Testen der Erkennung**
  22. Dezember 2023
- **Ende**
  1. Jänner 2024

10.2.2. Object Scanning

- **Start**
  8. Juli 2023
- **Machen von Fotos und Zwischenspeichern**
  5. August 2023
- **Eingabe des Beschreibungstextes ermöglichen und Zwischenspeichern**
  28. August 2023
- **Hochladen der Fotos und Texte an die Rest-API**
  12. September 2023
- **Auswahl geeigneter Fotogrammetrie Software**
  24. September 2023
- **Generieren der Objekte (lokal)**
  15. Oktober 2023
- **Erstellung der Generator REST-API**
  5. November 2023
- **Generieren der Objekte (remote)**
  22. Dezember 2023
- **Ende**
  1. Jänner 2024

10.2.3. Rest-API und Datenbank

- **Start**
  8. Juli 2023
- **Eigene Authentifizierung**
  1. August 2023
- **Hochladen des Namens, der Beschreibung und der Fotos an die REST-API**
  1. September.2023
- **Anlegen von Testdaten /-objekten**
  15. September 2023
- **Liefern aller Model Targets mit deren Texte**
  1. Oktober 2023
- **Hochladen der Fotos an einen 3D-Objekt Generator**
  1. November 2023
- **Hochladen des 3D-Objekts an die Model Target Web API**
  15. November 2023
- **Speichern der ID des Objektes und dazugehörigen Text in unsere Datenbank**
  1. Dezember 2023
- **Ende**
  1. Jänner 2024

10.3. Projektzeitplan

Innerhalb unserer Diplomarbeit haben wir uns mit dem Online-Tool „Flyingdonut“[9] organisiert. Dieses Werkzeug haben wir bereits im Rahmen unseres Schulprojektes „Project-AR“ verwendet. Hierbei erstellen wir jeden Monat ein sogenanntes „Scrumban-Board“ welches uns erlaubt, unsere Arbeitspakete übersichtlich darzustellen.

10.3.1. Scrumban

Scrumban ist ein Begriff, der aus den beiden Projektorganisationsformen „Scrum“ und „Kanban“ entstanden ist. Scrumban ist eine agile Projektmanagement-Methode, die beispielsweise sowohl einen Backlog besitzt, was für Kanban üblich ist, und auch die Arbeit nach Nachfrage sowie Komplexität entsprechend vorgereiht wird, was ein Merkmal von Scrum darstellt.[10]

Bei dieser Organisationform gibt es ein sogenannten „Scrumban-Board“, welches eigentlich ein Kanban-Board ist. Dieses enthält einen Product Backlog und einen Sprint Backlog, sowie den derzeitigen Zustand des Arbeitspaketes in den Spalten des Boards. Typisch hierfür sind „To-Do“, „In Bearbeitung“, „Testen“ und eine abgegrenzte Spalte für vervollständigte Aufgaben.

10.3.2. Unsere Boards

[9] [FLYING DONUT SOFTWARE P.C.]
[10] [ASANA, INC.]

In diesem Kapitel werden die ersten unserer Organisations-Boards genauer erläutert. Hierzu ist zu sagen, dass diese Übersicht der Arbeitspakete jeden Monat nach einem Meeting mit unserem Auftraggeber erstellt wird. Unser Ziel besteht darin, alle sogenannten „Tickets“ nun bis zum nächsten Treffen zu erledigen. Als Beispiel kann unser erster Projektabschnitt herangezogen werden, welcher vom 1. August 2023 bis zum 16. August 2023 dauerte. Die Übersicht in unserem Planungs-Tool wird in Abbildung 3 abgebildet.

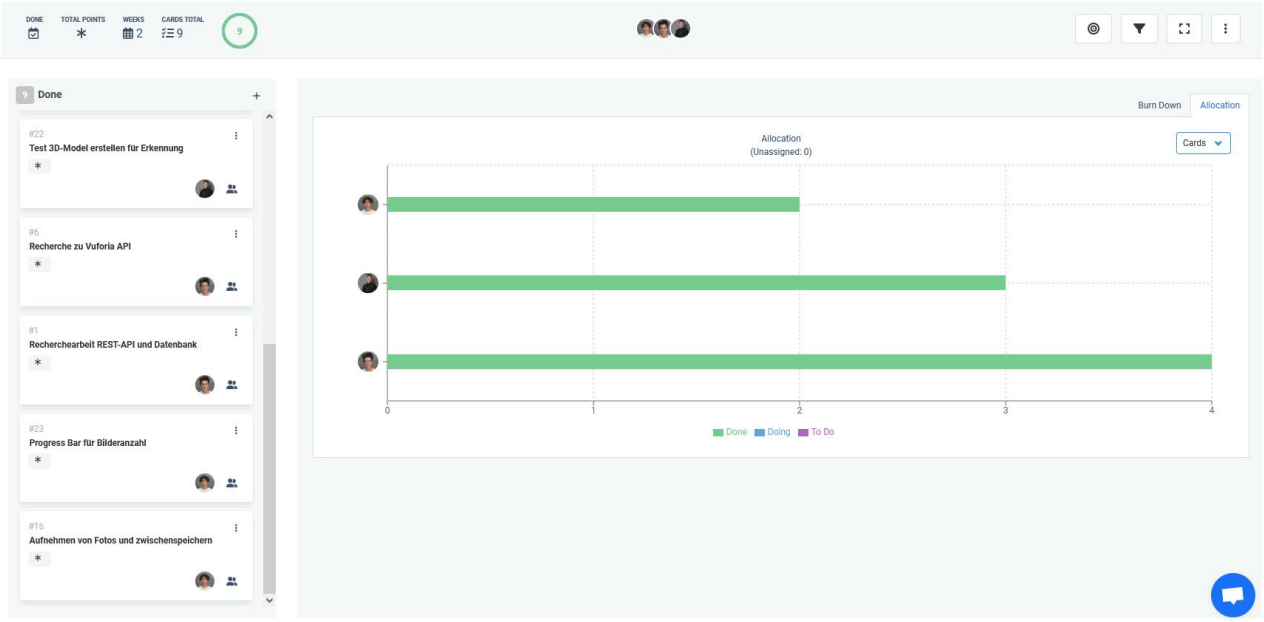


Abbildung 3: Übersicht in "Flyingdonut" Planungs-Tool [FLYING DONUT SOFTWARE P.C.]

Folgende Arbeitspakete wurden in diesem ersten Abschnitt erledigt:

B = Benedikt Hofer
J = Jakob Schmid
F = Florian Haider

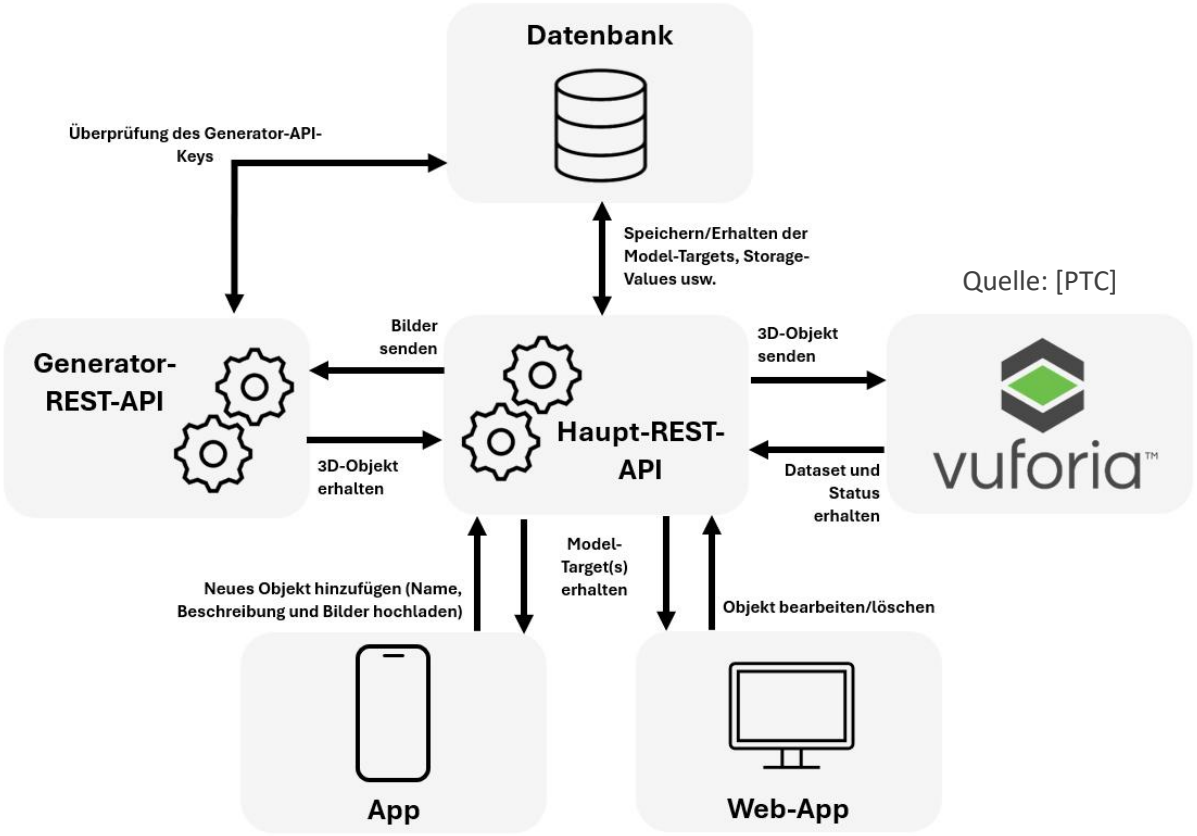
- **(B)** Testdatenlegen
- **(B)** Datenbank ERD erstellen
- **(F)** Einblenden der Texte zu den erkannten Objekten
- **(F)** Implementieren der visuellen Anzeige der erkannten Objekte, ohne Texte
- **(F)** Test 3D-Modell erstellen für Erkennung
- **(B)** Recherche zu Vuforia API
- **(B)** Recherchearbeit REST-API und Datenbank
- **(J)** Progress Bar für Bilderanzahl
- **(J)** Aufnehmen von Fotos und Zwischenspeichern dieser

Hierbei ist zu erwähnen, dass für jeden unserer Projektabschnitte ein solcher Plan erstellt wurde, in unserer Verschriftlichung der Arbeit stellen wir hierbei nur den ersten dar, da alle weiteren den Rahmen dieser sprengen würden.

11. Implementierung

11.1. Technischer Überblick

Aus Implementierungssicht besteht die Anwendung aus der Mobile-App für den Monteur und unserem Backend. Letzteres setzt sich aus einer REST-API, die sich um die allgemeine Business Logic kümmert, einer Datenbank, die zusätzliche Daten wie die Namen und Beschreibungen der Model-Targets speichert, und einer weiteren REST-API, die für das Generieren der 3D-Modelle zuständig ist, zusammen. Zusätzlich wird von der REST-Schnittstelle aus auf die Model Target API von Vuforia zugegriffen. Außerdem existiert noch eine Webapplikation, die die Verwaltung vereinfacht.



Quelle: [PTC]

11.2. REST-API

11.2.1. Einleitung

Diese Schnittstelle sorgt dafür, unsere Business logic zentralisiert zur Verfügung zu stellen. Heruntergebrochen sind die Aufgabenbereiche der REST-API neue 3D-Modelle hinzuzufügen, zu bearbeiten und bereits persistierte zur Verfügung zu stellen. Die API ist unter „https://project-ar.azurewebsites.net“ erreichbar.

11.2.2. Codestruktur und Aufbau

11.2.2.1. Einleitung

Der Strukturierung des Codes kommt vor allem bei wachsenden Applikationsgrößen eine große Rolle zu. Ein durchdachter Aufbau dahingehend resultiert in einer erhöhten Fehlerfreiheit und Wartbarkeit. Aus diesem Grund teilt man den Code auf verschiedene Bereiche auf. Auf den genauen Aufbau der REST-API wird nun in den folgenden Unterkapiteln eingegangen.

11.2.2.2. Controller

Die Controller sind für die Annahme und Beantwortung der HTTP-Anfragen an die Schnittstelle verantwortlich. Sie sind sozusagen der Einstiegspunkt für jede Anfrage. Controller sollten keine Business logic[11] beinhalten, sondern lediglich delegieren und eine entsprechende Antwort an den Aufrufer zurücksenden. Damit ist der HTTP-Statuscode, wie zum Beispiel „200 OK“, „400 Bad Request“ etc. gemeint. Außerdem kann die Antwort auch ein Objekt beinhalten, wie zum Beispiel eine Liste unserer Model-Targets. Wichtig ist zudem, dass die Controller nur für eine Menge an zusammenhängenden Funktionalität zuständig sein sollen. Konkret bedeutet das, dass ein Controller, der sich um Model-Targets kümmert, nicht auch gleichzeitig Endpoints für die Authentifizierung bereitstellen soll. Hierfür ist es ratsam, einen eigenen Controller anzulegen.

11.2.2.3. Interfaces

Interfaces sind Schnittstellen, die Funktionalitäten, beziehungsweise Methoden vorgeben, die wiederum von Services implementiert werden. Die Interfaces tragen üblicherweise den Namen des Services, der sie implementiert mit einem „I“ als Präfix.

Auf die Methoden des Interfaces, beziehungsweise auf die Methoden des implementierenden Services, wird dann in den Controllern zugegriffen. Die Dependency Injection sorgt dafür, dass eine Objektreferenz der Service-Klasse an den Controller übergeben wird.

Über diese Referenz können dann die entsprechenden Methoden aufgerufen werden. Da man die Interfaces per Dependency Injection zur Verfügung stellt und nicht die Service-Klassen selbst, entsteht eine sogenannte „Lose Kopplung“. Allgemein beschreibt dieser Begriff ein Konzept in der Softwareentwicklung, das die gegenseitige Abhängigkeit von Klassen möglichst reduziert. Dadurch kann die konkrete Implementierung des Services ausgetauscht werden, ohne dass abhängige

[11] Die „Business logic“ ist jene Logik, die das eigentliche Vorhaben des Systems abbildet.

Komponenten dadurch beeinträchtigt und überarbeitet werden müssen. Dies sorgt für eine erhöhte Flexibilität und reduziert Aufwand.

11.2.2.4. Services

Die sogenannten Services beinhalten nun die Business logic. Sie implementieren die in den Interfaces vorgegebenen Methoden aus. In unserem Projekt gibt es Services, die für den Umgang mit den Entitäten der Datenbank zuständig sind, aber auch Services, die mit der Vuforia Model Target Web API arbeiten. Hier sind alle Services unserer REST-API aufgelistet und kurz beschrieben:

- Datenbank Services: Verwaltung der jeweiligen Datenbankentitäten, die im Kapitel 11.3.3 beschrieben werden.
  - AirUserDbService
  - ModelTargetDbService
  - RefreshTokenDbService
  - StorageKVDbService
- AuthService
  - Beinhaltet eine Methode zum Einloggen des Benutzers. Liefert eine Fehlermeldung bei falschen Anmeldedaten.
- TokenService
  - Verwaltung aller Token der eigenen Authentifizierung. Beinhaltet Methoden zum Generieren von Access-Token und Refresh-Token und zum Überprüfen des Refresh-Tokens.
- GenerationService
  - Kümmert sich um die Kommunikation mit unserer weiteren REST-API, die für das Generieren der 3D-Objekte aus den geschossenen Bildern zuständig ist.
- VuforiaService
  - Dieser Service ist für die Kommunikation mit der Vuforia Model Target Web API zuständig, die in Kapitel 11.2.4 beschrieben wird.

11.2.2.5. ORM

ORM steht für „object-relational mapping“. Wir greifen auf “Entity Framework Core“ zurück, um unsere objektrelationale Abbildung umzusetzen. Man verwendet Frameworks, die objektrelationales Mapping erlauben, um die relationale Welt in die Welt der Objektorientierung zu überführen, damit man also mit der Datenbank arbeiten kann, ohne selbst die entsprechenden Queries zu entwickeln und die Resultate in eigene Objekte zu geben. Entity Framework kommuniziert direkt mit der Datenbank, während wir in unserer Business logic nur auf den zur Verfügung gestellten „DbContext“ mit den entsprechenden Entitäten zugreifen.

Zum Abbilden der Relationen haben wir einen Database-First Ansatz gewählt, also zuerst die Datenbank entworfen und dann die entsprechenden Klassen für die REST-API dazu erstellt.

Als Beispiel für eine Modell-Klasse, wie sie in Entity Framework definiert ist, ist hier „ModelTarget“ herausgegriffen. Diese Entität stellt das Model-Target dar, speichert also den Namen, die Beschreibung, das Dataset, etc. für das entsprechende Objekt. Eine genauere Beschreibung ist im Abschnitt 11.3.3.2 vorzufinden.

```
[Table("ModelTarget")]
 public class ModelTarget {

     [Key]
     public Guid Id { get; set; }
     public string Name { get; set; }
     public string Description { get; set; }
     public string? VuforiaId { get; set; }
     public string ModelGenerationStatus { get; set; }

 }
```

11.2.2.6. DTOs

Ein “DTO” ist ein “Data Transfer Object“, das dafür verwendet wird, um Daten zwischen zwei Prozessen übermitteln zu können. Im Projekt kommen diese vor allem in der Kommunikation zwischen dem Aufrufer und der REST-API selbst zum Einsatz. Beispielweise wurde das „ObjectRecognitionDTO“ definiert, das - als JSON serialisiert – innerhalb einer Liste an den Aufrufer zurückgesendet wird, wenn alle Model-Targets abgefragt werden.

DTOs bringen den Vorteil, dass interne Speicherstrukturen, wie zum Beispiel die Entity Framework Modell-Klasse für das Model-Target, von den externen entkoppelt werden. Außerdem beinhalten diese Objekte nur die Daten, die für den Aufrufer relevant sind, was den Datenverkehr reduziert.

11.2.2.7. Program.cs

In der „Program.cs“ Datei wird die API konfiguriert. Hier werden die Services angegeben, die später durch die Dependency Injection zur Verfügung gestellt werden sollen. Zusätzlich wird hier die Authentifizierung mit den JSON Web Tokens definiert, wie in Kapitel 11.2.3 beschrieben. Außerdem wird hier festgelegt, dass Swagger zur Dokumentation und visuellen Darstellung der verfügbaren Endpoints verwendet werden soll.

11.2.3. Authentifizierung und Autorisierung

11.2.3.1. Einleitung

Da die REST-Schnittstelle von den mobile Geräten der Monteure aus aufgerufen wird, ist sie öffentlich gehostet. Aufgrund dieser Tatsache sind eine umfassende Authentifizierung und Autorisierung des Benutzers vonnöten, um die gespeicherten Daten abzusichern und das System vor unbefugtem Zugriff zu bewahren.

Hierbei ist die Entscheidung auf eine eigene Implementierung des JSON Web Token Verfahrens gefallen.

11.2.3.2. JWT - Allgemein

JSON Web Tokens (JWTs) sind eine Art von Token, die zur sicheren Übertragung von Informationen zwischen Kommunikationspartnern in Form eines JSON-Objekts verwendet werden[12]. Allgemein werden diese Token nach erfolgreicher Anmeldung des Benutzers generiert und können fortan bei zukünftigen HTTP-Anfragen im Request-Header mitgesendet werden, um den Benutzer zu authentisieren. Dies hat den bedeutenden Vorteil, dass für die Gültigkeitsdauer des Tokens keine sensiblen Anmeldedaten wie Username und Passwort gesendet werden müssen. JSON Web Tokens bestehen aus drei Teilen: Header, Payload und Signature.[13]

Ein typischer JWT-Header besteht aus dem Token-Typ, welcher in diesem Fall „JWT“ ist und dem verwendeten Signaturalgorithmus, beispielsweise „SHA256“.

```
{
    "alg": "HS256",
    "typ": "JWT"
}
```

Im Payload des Tokens befinden sich die sogenannten „claims“. Claims beinhalten Daten über eine Entität, oftmals betreffen diese den Benutzer[14]. Es gibt verschiedene Arten von diesen. „Registered claims“ sind vordefinierte, empfohlene claims wie zum Beispiel „exp“, was für „Expiration Time“ steht, und festlegt, wie lange der Token gültig ist. „Public claims“ und „Private claims“ sind inoffizielle claims, die sich lediglich darin unterscheiden, dass bei Private claims nur intern Kommunikationspartner definieren, was die Bedeutung eines claims ist, während Public claims für alle das gleiche bedeuten. Ein Beispiel für Public claims wäre „roles“, das festlegt, welche Rollen der Benutzer besitzt und wie er dadurch im System operieren darf. Ein Private claim könnte zum Beispiel „dept“ sein, der bestimmt, in welcher Abteilung ein Mitarbeiter arbeitet.

```
{
    "sub": "12345678",
    "name": "John Doe",
    "exp": 1702218483,
    "iat": 1702208483,
    "dept": 1
}
```

Die Signature wird aus dem Header, dem Payload und einer geheimen Zeichenfolge, die nur der REST-API bekannt ist, generiert. Dabei wird der im Header definierte Algorithmus wie zum Beispiel SHA256 auf eine Zeichenkette angewendet, die sich aus dem „Base64“ kodierten Header und dem „Base64“ kodierten Payload mit einem Punkt dazwischen, zusammensetzt. Ein Pseudo-Code Beispiel für das Erstellen einer Signature könnte folgendermaßen aussehen:

```
HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)
```

[12] [OKTA INC.]
[13] [INTERNET ENGINEERING TASK FORCE (IETF)]
[14] [IANA]

„Base64“ ist ein Verfahren zur Umwandlung von Daten zu einer Zeichenkette. In der ursprünglichen Version umfasst es einen Satz von 64 Zeichen. Es wird häufig verwendet, um binäre Daten in einem Textformat übertragen zu können, wie auch im Falle unserer Übertragung des 3D-Modelles an die Model Target API von Vuforia. Mit Bezug auf JSON Web Tokens wird der Algorithmus ebenfalls verwendet, um etwaigen Komplikationen bei der Datenübertragung vorzubeugen.

Bei der Validierung des Tokens wird die Signature erneut errechnet und mit der mitgesendeten im Token verglichen. Dadurch kann sichergestellt werden, dass der Token nicht manipuliert worden ist.

Ein JSON Web Token wird allerdings nicht in der JSON-Darstellung versendet. Alle drei Teile werden wiederum mithilfe des „Base64“ Verfahrens einzeln kodiert und mit einem Punkt dazwischen zu einer Zeichenkette zusammengefügt. Ein Beispiel für einen JWT, wie er tatsächlich versendet wird, könnte so aussehen:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

- Header
- Payload
- Signature

11.2.3.3. Access-Tokens und Identity Tokens

Grundlegend lassen sich JWTs in zwei unterschiedliche Kategorien unterteilen. Die sogenannten Identity Tokens sind dazu da, um Daten über den Benutzer bereitzustellen, während Access-Tokens für Autorisierungszwecke benutzt werden. Das Access-Token regelt, worauf zugegriffen werden darf und wird auf der REST-Schnittstelle bei jeder Anfrage auf einen geschützten Endpoint validiert. Das Identity Token hingegen wird oftmals verwendet, um Informationen über den Anwender auf Benutzeroberflächen anzuzeigen und kann clientseitig validiert werden.

11.2.3.4. Refresh-Token

Ein Refresh-Token wird dazu verwendet, um einen bereits einmal mit Username und Passwort angemeldeten User längerfristig immer wieder anmelden zu können, ohne diese Anmeldedaten erneut abzufragen. Dies passiert allgemein in Form eines sogenannten Silent-Logins. Der Name bezieht sich darauf, dass dem Anwender nicht aktiv bewusst gemacht wird, dass momentan eine neue Anmeldung erfolgt. Dies funktioniert dadurch, dass der Refresh-Token am Gerät des Benutzers gespeichert wird und bei einem abgelaufenen Access-Token verwendet wird, um einen neuen zu erhalten.

11.2.3.5. Authentifizierung – Ablauf

Die Authentifizierung und das damit einhergehende Erhalten eines Access-Tokens kann bei unserer REST-API auf zwei unterschiedliche Arten ablaufen.

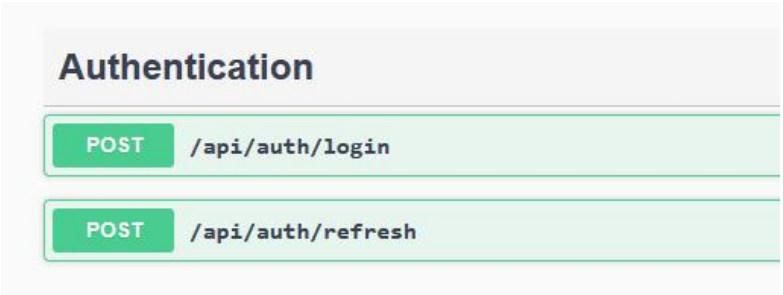


Abbildung 4: Endpoints Authentifizierung

Die erste Möglichkeit, Zugang zu erhalten, besteht darin, den „login“-Endpoint der Schnittstelle aufzurufen. Dieser erwartet einen Request -Body, in dem Username und Passwort enthalten sind:

```
{
    "username": "string",
    "password": "string"
}
```

Bei erfolgreicher Anmeldung erhält man eine Antwort, die folgendermaßen aussehen kann:

```
{
"accessToken":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ1c2VySWQiOiJkNThjMWJjYi04OTJkLTQ1ODgtOTYxZC05ZjE4Mjc3YzcwODEiLCJleHAiOjE3MDIyMj
Y0MjZ9.64JWHIU9oVpt7oquvvkBUSCxqzV89ZiMWDhwQel6j0k",
"refreshToken": "XI5OHhjuqC1pvD13dErw7prD6AOxo4af"
}
```

Die zweite Option ist, den „refresh“-Endpoint zu verwenden:

```
{
    "refreshToken": "string"
}
```

Bei einem gültigen Refresh-Token erhält man wiederum einen neuen Access-Token:

```
{
"accessToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJkNThjMWJjYi04OTJkLTQ1ODgtOTYx
ZC05ZjE4Mjc3YzcwODEiLCJleHAiOjE3MDIyMjczMjV9.AFRfv7gOdaGVEGjJ095OgJ29arH6qBuwmWOPa
7rrCDc"
}
```

Das erhaltene Access-Token kann nun in den Header der nächsten Anfrage gegeben werden, um Zugriff auf die weiteren Ressourcen zu erhalten.

11.2.3.6. JWT - Codebeispiel

Unsere Anwendung implementiert das JSON Web Token Verfahren, um die Benutzer mithilfe des Access-Tokens zu autorisieren. Alle Endpoints der REST-Schnittstelle sind geschützt, abgesehen von den öffentlich zugänglichen Authentifizierungsmethoden. Die Operationen in Bezug auf unsere JWTs funktionieren mithilfe des Microsoft Packages „System.IdentityModel.Tokens.Jwt“.

Die Methode, welche bei der REST-API für das Generieren der entsprechenden Access-Tokens zuständig ist, ist hier abgebildet:

```
public string GenerateAccessToken(Guid userId) {

    var claims = new[]
    {
        new Claim("userId", userId.ToString()),
    };

    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(JWT_SECRET_KEY));
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    var token = new JwtSecurityToken(
        claims: claims,
        expires: DateTime.Now.AddMinutes(120),
        signingCredentials: creds);

    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

Abbildung 5: Codeausschnitt der Tokengenerierung

In diesem Code Ausschnitt werden zuerst die benötigten Parameter für das Erzeugen eines Tokens initialisiert. Hierbei handelt es sich um einen Private claim namens „userId“, welcher die Identifikationsnummer des Benutzers aus der Datenbank repräsentiert. Als nächstes wird der benötigte Schlüssel anhand der Variable JWT_SECRET_KEY erstellt und anschließend in einem „SigningCredentials“ Objekt verarbeitet. Im vorletzten Schritt wird der Token nun mit den vorhin erstellten Variablen generiert und zum Schluss als Zeichenkette von der Methode zurückgegeben.

11.2.4. Vuforia Model Target Web API

11.2.4.1. Einleitung

Die Vuforia Model Target API ist eine öffentlich erreichbare REST-Schnittstelle, die es ermöglicht, 3D-Objekte hochzuladen und mithilfe eines KI-Trainings zu einem Datensatz zu transformieren. Dieser Datensatz kann dann über die verfügbare Schnittstelle wiederum heruntergeladen werden und innerhalb der App verwendet werden, um 3D-Objekte zu erkennen und die dazugehörigen Arbeitsschritte und Beschreibungen einzublenden.

Auf der offiziellen Webseite befindet sich eine Abbildung, die den Flow im Umgang mit der Model Target Web API darstellt.

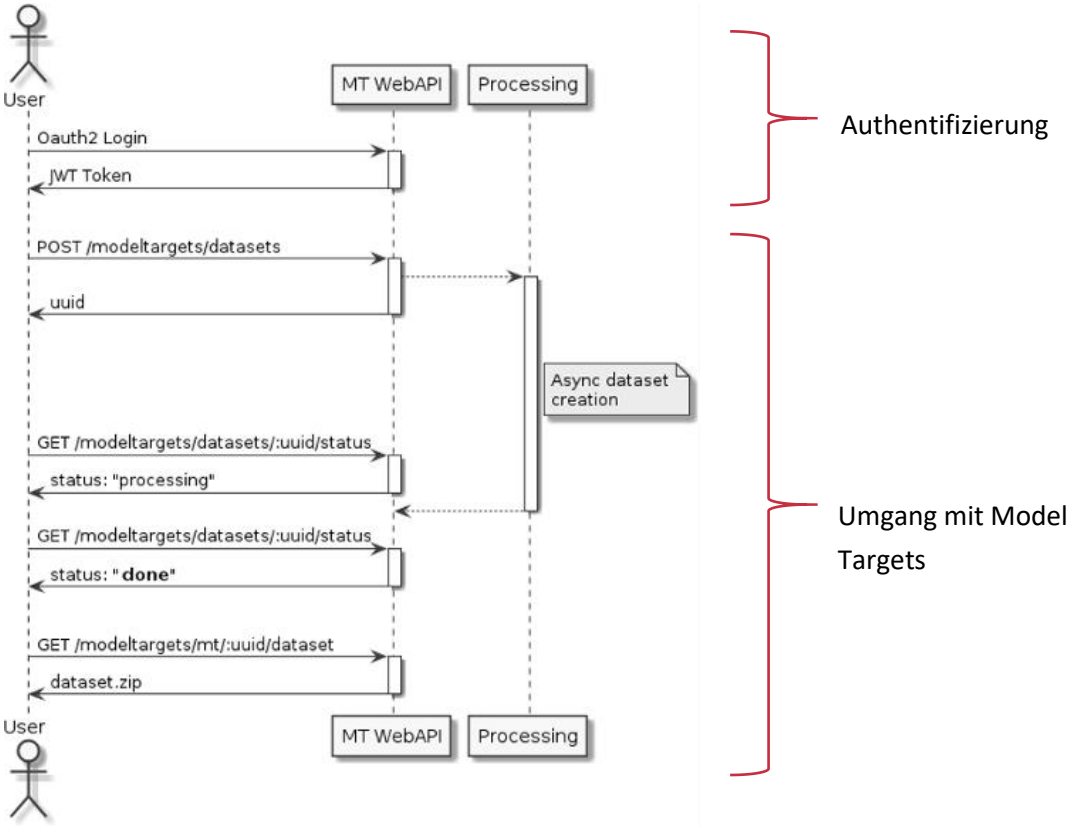


Abbildung 6: Authentifizierungsflow [PTC INC.]

https://developer.vuforia.com/sites/default/files/vuforia-

11.2.4.2. Authentifizierung

Wie auch unsere eigene REST-API, ist auch die Vuforia Model Target API durch ein JSON-Web Token Authentifizierungsverfahren geschützt. Um autorisiert zu werden, also um auf die weiteren Endpunkte der Schnittstelle wie das Hochladen von 3D-Modellen zugreifen zu können, ist es nämlich nötig, einen korrekten, von der API bei der Anmeldung ausgegebenen JSON Web Token im Header der HTTP-Anfrage mitzusenden. Um diesen Token zu erhalten, bietet die Schnittstelle zwei Möglichkeiten an.

In beiden Fällen muss zuerst ein Konto im „vuforia engine developer portal“[15] erstellt werden.

Die erste Möglichkeit besteht nun darin, sogenannte „Client Credentials“ zu erstellen und diese zu verwenden, um den Token zu erhalten.

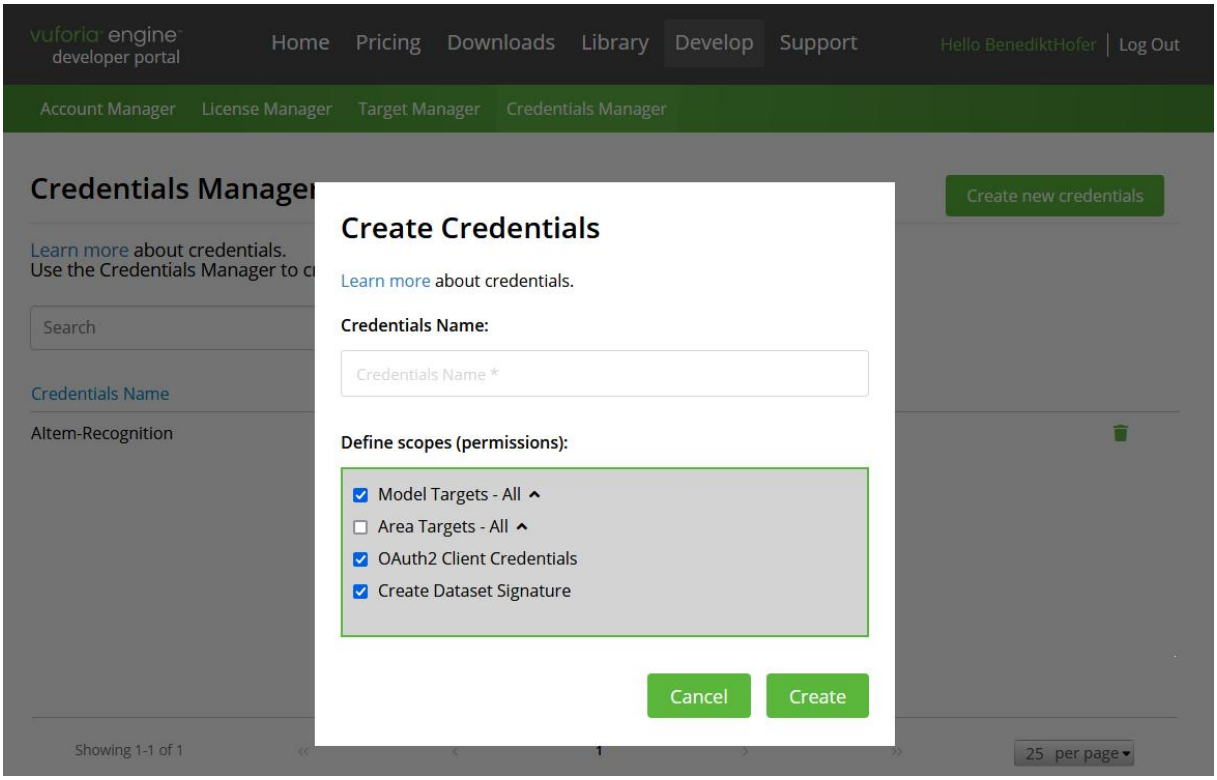


Abbildung 7: Client Credentials erstellen

Bei der zweiten Möglichkeit wird das Passwort des erstellten Kontos in der Anfrage mitgesendet.

In unserem Fall kommen die „Client Credentials“ zum Einsatz, um den Token zu erhalten. Die Anfrage dafür muss einen Header besitzen, der gemäß der „Basic access authentication“ aufgebaut ist.

Der Header einer Pseudo-Anfrage sieht folgendermaßen aus:

```
"Authorization": "Basic base64(‘{client_id}:{client_secret}’)"
```

Bei dem Body der Anfrage wird der media-type „application/x-www-form-urlencoded“ und die Attribute „grant_type“ sowie „scope“ erwartet.

```
"grant_type": "client_credentials"
"scope": "modeltargets.all"
```

[15] [PTC INC.]

Gesendet an die Adresse „https://vws.vuforia.com/oauth2/token“ liefert die obige Anfrage bei korrekten „client credentials“ eine Antwort, die folgendermaßen aussieht:

```
{
    "access_token": "{JWT_TOKEN}",
    "token_type": "bearer",
    "expires_in": 3600
}
```

Der im Response-Body erhaltene Token kann nun, wie bereits erwähnt, in zukünftigen Anfragen an die Model Target Web API verwendet werden, damit eine Autorisierung stattfindet.

11.2.4.3. Model Targets – Allgemein, Arten

Bei der Model Target Web API wird grundsätzlich zwischen zwei Arten von Model-Targets unterschieden. „Model Targets“ und „Advanced Model Targets“.

Model Targets basieren auf 3D-Modellen von echten Objekten. Die Struktur des Objekts wird analysiert und verwendet, um ein Dataset zu erzeugen und dadurch das AR-Tracking zu ermöglichen. Wenn die Kamera eines Geräts das entsprechende Objekt erfasst, kann die AR-Anwendung dieses erkennen und in unserem Fall daraufhin die einzublendenden Beschreibungen genau darauf ausrichten.

Advanced Model Targets bieten zusätzlich die Möglichkeit, das 3D-Objekt innerhalb des definierten Bereiches zu erkennen, ohne dass der Nutzer die Sicht genau auf das physische Objekt ausrichten muss. Aus diesem Grund wird in dem Projekt auch auf diese Art zurückgegriffen.

11.2.4.4. Endpoints - Übersicht

Die Basis-Adresse für alle Endpoints lautet „https://vws.vuforia.com“.

Alle Endpoints, die von der Model Target Web API zur Verfügung gestellt werden, sind hier aufgelistet.

Advanced Model Targets

```
POST /modeltargets/advancedDatasets
```

Mit diesem Endpoint kann man Advanced Model-Targets ausgehend von einem bereits vorhandenem 3D-Modell in den Vuforia Speicher laden. Danach startet das Generieren des Datasets automatisch.

```
DELETE /modeltargets/advancedDatasets/{uuid}
```

Dieser Endpoint ermöglicht es, ein Advanced Model-Target anhand der einzigartigen ID wieder aus dem Vuforia Speicher zu löschen.

```
GET /modeltargets/advancedDatasets/{uuid}/dataset
```

Der Endpoint ermöglicht es, das Dataset, dass später in der App zum Erkennen der Objekte gebraucht wird, herunterzuladen.

```
GET /modeltargets/advancedDatasets/{uuid}/status
```

Mithilfe dieses Endpoints kann der Status des Trainingsprozesses des Datasets abgefragt werden. Die Antwort beinhaltet entweder „done“, „processing“ oder „error“. Im letzten Fall ist eine weitere genauere Fehlerbeschreibung inkludiert.

Analog zu den Advanced Model Targets, existieren die gleichen Endpoints für die Model Targets.

Model Targets

```
POST /modeltargets/datasets

DELETE /modeltargets/datasets/{uuid}

GET /modeltargets/datasets/{uuid}/dataset

GET /modeltargets/datasets/{uuid}/status
```

11.2.4.5. Ablauf

Auf welche Endpoints wurde nun im Projekt zugegriffen? Der Ablauf in Bezug auf die Vuforia Model Target REST API sieht folgendermaßen aus:

Möchte der Benutzer ein neues Model Target hinzufügen, greift unsere REST-Schnittstelle auf den Vuforia Model Target Web API Endpoint „`POST /modeltargets/advancedDatasets`“ zu, um das neue 3D-Modell hochzuladen und zu trainieren. Das Trainieren beziehungsweise das Generieren des Datasets wird wie bereits erwähnt automatisch durch Vuforia initiiert.

Damit alle verfügbaren und erkennbaren Objekte in der App angezeigt werden können, müssen diese zuerst aus dem Speicher von Vuforia geholt werden. Dafür greift unsere API auf die zwei Model Target Web API Endpoints „`GET /modeltargets/datasets/{uuid}/dataset`“ und „`GET /modeltargets/datasets/{uuid}/status`“ zu.

Ersterer Endpoint wird dazu verwendet, um das Dataset in der Antwort unserer eigenen REST-API zu inkludieren.

Der zweite Endpoint liefert den Status den Generierungsprozesses des Datasets, der ebenfalls in der Antwort der eigenen REST-Schnittstelle eingebaut wird.

Außerdem wird noch auf „`DELETE /modeltargets/datasets/{uuid}`“ zugegriffen, wenn ein Objekt gelöscht werden soll.

11.2.4.6. OpenAPI Specification

Der Zugriff auf die Endpoints, wie sie in den obigen Kapiteln beschrieben werden, erfolgt durch die mittels des OpenAPI Generators für C# erstellten Klassen, die direkt im Code unserer eigenen REST-API verwendet werden.

```
public async Task<string> GetDatasetStatus(string uuid) {

    var response = await advancedDatasetsApi.GetModelTargetAdvancedDatasetStatusWithHttpInfoAsync(uuid);

    if (response.StatusCode == System.Net.HttpStatusCode.OK) {

        string responseContent = response.RawContent;

        if (responseContent.Contains("done")) {
            return DATASET_STATUS_DONE;
        }
        else if (responseContent.Contains("processing")) {
            return DATASET_STATUS_PROCESSING;
        }
        else {
            return DATASET_STATUS_FAULTY;
        }

    }
```

Abbildung 8: Codeausschnitt OpenAPI Generator

Auf dieser Abbildung ist ein Codeausschnitt zu erkennen, bei welchem der Status des Dataset über die Vuforia Model Target Web API abgefragt wird. Dies passiert durch Zugriff auf die Service-Klassen die basierend auf der OpenAPI Spezifikation[16] generiert worden sind.

11.2.5. Hinzufügen von Model-Targets

11.2.5.1. Einleitung

Der Bereich „Object Scanning“ greift im letzten Schritt auf die REST-Schnittstelle zu, um die Eingaben des Monteurs zu speichern und damit das eingescannte Objekt in weiterer Folge in das System einzubringen. Der Input des Benutzers bezieht sich hierbei auf den Namen des eingescannten Objektes, der dazugehörigen Beschreibung, die wiederum Arbeitsschritte oder Definitionen in Textform beinhaltet und die Bilder des Objektes.

11.2.5.2. Übermittlung

Diese Daten werden mittels des Media Types „multipart/form-data“ übermittelt. Dieser sogenannte MIME Type ist dafür geeignet, verschiedene Typen von Daten innerhalb einer Anfrage an die Zielschnittstelle zu senden. Ein weiterer verfügbarer Media Type ist „application/x-www-form-urlencoded“, welcher allerdings bei großen Mengen an Binärdaten ineffizient ist. In unserem Kontext ist der gewählte MIME Type also dafür vorgesehen, sowohl den Namen und die Beschreibung als auch die dazugehörigen Bilder in Form von Byte-Arrays, also im binären Format, vom Mobile Gerät des Monteurs an unsere REST-Schnittstelle zu transferieren.

[16] [PTC INC.]

11.2.5.3. Speicherung in der Datenbank

Im ersten Schritt sorgt unsere REST-API dafür, dass die eingegangenen Daten in unserer eigenen Datenbank persistiert werden. Hierbei wird der Name des Objekts und die Beschreibung dazu gespeichert. Zudem wird der Startwert für den Generierungsstatus des Datensatzes festgelegt. Der Generierungsstatus sagt aus, ob das 3D-Modell momentan aus den Bildern generiert wird, beziehungsweise ob ein Fehler aufgetreten ist oder der Prozess erfolgreich abgeschlossen werden konnte. Dieser Status ist notwendig, da diese Operation aufgrund der längeren Dauer nicht synchron abgewickelt werden kann und daher eine vom Prozess unabhängige Abfragemöglichkeit benötigt wird. Nach dieser Operation wird die von der Vuforia Model Target Web API erhaltene uuid des zu trainierenden Models in der Datenbank gespeichert.

11.2.5.4. Anfrage an 3D-Model-Generator

Als Nächstes werden die erhaltenen Bilder des echten Objektes zu einer 3D-Objekt Datei verarbeitet. Dies passiert über eine weitere REST-Schnittstelle, die unabhängig zu der anderen aufrufbar ist.

Der Generator ist unter „https://20.73.144.169/api/Generator“ aufrufbar.

Dazu werden die erhaltenen Bilder nun an diese Schnittstelle gesendet, um den Generierungsprozess anzustoßen. Der verwendete Endpoint dafür ist „`POST /api/Generator`“. Im Body des Requests wird eine Liste an Byte Arrays erwartet.

Der Prozess des Generierens dauert mehrere Minuten, weswegen die Haupt-REST-Schnittstelle in Intervallen von jeweils 60 Sekunden abfragt, ob das Generieren erfolgreich abgeschlossen werden konnte. Dies passiert durch die Abfrage des Endpoints „`/api/Generator/status/{uuid}`“. In diesem Falle wird das erhaltene 3D-Modell als nächstes auf Vuforia hochgeladen, wo dann das entsprechende Dataset durch Training erstellt wird. Wenn das Model-Target

11.2.5.5. Model-Target Training

Um ein Dataset zu erhalten, welches in späterer Folge in der Mobile App zur Erkennung der 3D-Objekte verwendet wird, wird auf die Vuforia Model Target Web API zugegriffen, die für das Training verantwortlich ist.

Dazu wird der Endpoint „`POST /modeltargets/advancedDatasets`“ verwendet. Dieser fordert eine Anfrage mit folgenden Bedingungen.

Header:

```
„Authorization“: „Bearer <access_token>“
```

Da dieser Endpoint vor unbefugtem Zugriff geschützt ist, verlangt er eine Authentifizierung. Wie bereits erwähnt funktioniert dies wiederum mit dem erhaltenen Access-Token im Header.

Body:

```
{
  "name": "<name>",
  "targetSdk": "10.5",
  "models": [
    {
      "name": "<name>",
      "cadDataBlob": "<data>",
      "views": [
        {
          "name": "<name>",
          "targetExtentPreset": "FULL_MODEL",
          "recognitionRangesPreset": "FULL_360"
        }
      ]
    }
  ]
}
```

Der Name in diesem Objekt definiert den Namen, den das Dataset bekommt. „cadDataBlob“ hält die 3D-Objektdatei als Base64 String bereit. Durch dieses Attribut kann also die 3D-Objektdatei an die Vuforia Model Target Web API hochgeladen werden. Die Views, die in diesem Objekt vorhanden sind, definieren, wie das Model aus verschiedenen Blickwinkeln erkannt werden soll.

Zusätzlich können im Request-Body optional folgende Dinge festgelegt werden, um das Erkennen der Model-Targets zu konfigurieren und damit zu erleichtern.

targetExtent
Legt eine Box fest, welche den Bereich des 3D-Modells begrenzt. Dieser wird für die Erkennung herangezogen.

Rotation
Wird mittels eines sogenannten Quaternions angegeben, der die Rotation des Modells darstellt. Quaternionen sind eine mathematische Darstellung von Rotationen in 3D-Raum. Die vier Zahlen repräsentieren die Rotation um die x-, y- und z-Achsen sowie den Winkel, um den rotiert werden soll.

azimRange
Ein Array, das den Bereich der sogenannten Azimutwinkel, welche horizontale Winkel definieren, innerhalb dessen das Modell erkannt werden soll. Dieser Bereich wird in Radiant angegeben, wobei -π bis +π einem vollen Kreis (360°) entspricht.

elevRange
Ein Array, das den Bereich der Elevationswinkel beziehungsweise vertikalen Winkel definiert, innerhalb dessen das Modell erkannt wird. Auch dieser Bereich wird in Radiant angegeben, wobei -π/2 bis +π/2 einem vollen vertikalen Bereich entspricht.

rollRange
Ein Array, das den Bereich der Rollwinkel definiert. Rollwinkel werden angegeben, wenn die Drehung des Modells um seine eigene Achse beschrieben wird. Der Bereich wird ebenfalls in Radiant angegeben.

11.2.6. Erhalten von Model-Targets

11.2.6.1. Einleitung

Damit das zu erkennende Objekt in der Mobile-App ausgewählt werden kann und in weiterer Folge erkannt wird, bedarf es einer Möglichkeit, die gespeicherten Model-Targets, also die Datasets inklusive derer Namen und Beschreibungen von der Schnittstelle zu erhalten.

11.2.6.2. Endpoint

Der Endpoint der für das Erhalten der Model-Targets verantwortlich ist, lautet: „GET /api/model-targets“. Der Endpoint bietet die Möglichkeit, die Datasets als Antwort auf die selbe Anfrage base64-kodiert zu erhalten oder die Datasets in der Antwort zu exkludieren. Umgesetzt wird dies mithilfe eines Query-Parameters namens „includeDatasets“. Ist dieser auf „true“ gesetzt, werden die Datasets inkludiert, ansonsten nicht. Diese Option ermöglicht eine schnellere Antwort. Zusätzlich gibt es die Möglichkeit, „start“ und „end“ als Query Parameter festzulegen. Gemeinsam beschränken diese die zurückgegebenen Objekte. Wird „start“ mit dem Wert 0 und „end“ mit dem Wert 9 übergeben, so werden die ersten 10 gespeicherten Objekte zurückgegeben.

Eine Antwort dieses Endpoints unserer eigenen Schnittstelle kann folgendermaßen aussehen:

```
{
    "total": 20,
    "returned": 2,
    "targets": [
      {
        "uuid": "85636de8-6b00-435a-88f6-7b59b72fe4b8",
        "name": "Testteil 1",
        "description": "Dies ist die Beschreibung des Testteiles 1.",
        "modelGenerationStatus": "DONE",
        "datasetZip": <base64-string>,
        "datasetStatus": "DONE"
      },
      {
        "uuid": "daa3ba6e-24fd-41ea-91e7-ab4c11dbd4d4",
        "name": "Testteil 2",
        "description": " Dies ist die Beschreibung des Testteiles 2.",
        "modelGenerationStatus": "DONE",
        "datasetZip": <base64-string>,
        "datasetStatus": "DONE"
      }
    ]
}
```

11.2.6.3. Erhalten des Datasets

Da die Datasets zu den entsprechenden Model-Targets von Vuforia gespeichert werden, müssen diese bei der Beantwortung der Anfrage zuerst heruntergeladen und dann erst als base64-Zeichenkette weitergegeben werden. Dies passiert über den Endpoint „`GET /modeltargets/advancedDatasets/{uuid}/dataset`“.

11.2.6.4. Abfragen des Trainingsstatus

Da das Training der Model-Targets ein Prozess ist, der mehrere Stunden in Anspruch nimmt und nicht synchron mit dem Hauptprozess läuft, sollte der Status des Trainings abfragbar sein. Somit kann differenziert werden, ob das zurückgegebene Model-Target bereits auf ein Dataset verweist, ob das Training noch andauert, oder ob ein Fehler aufgetreten ist. Dieser Status, welcher in dem obigen Codeausschnitt als „datasetStatus“ bezeichnet wird, ist in dem Antwortobjekt inkludiert, sodass dem Endbenutzer der Mobile-App angezeigt werden kann, was momentan mit dem Model-Target passiert.

11.2.7. Bearbeiten von Model-Targets

11.2.7.1. Einleitung

Die Namen und Beschreibungen der Model-Targets sollten im Nachhinein änderbar sein, damit Anpassungen, beispielsweise an Arbeitsschritten, realisiert werden können.

11.2.7.2. Endpoint

Das teilweise Bearbeiten von Entitäten wird standardkonform über die HTTP-Methode PATCH umgesetzt. Unser Endpoint für das Bearbeiten von Name und Beschreibung eines Model-Targets lautet daher „PATCH /api/model-targets/edit“. Es wird ein Objekt folgenden Aufbaus wiederum im Body der Anfrage erwartet, um das Zielobjekt zu bearbeiten:

```
{
    "uuid": "string",
    "name": "string",
    "description": "string"
}
```

Über das Attribut “uuid”, welches wieder die ID des Objekts darstellt, wird das zu bearbeitende Objekt identifziert. Der Name und die Beschreibung der Anfrage stellen den neuen Namen und die neue Beschreibung des zu aktualisierenden Objekts dar.

11.2.8. Löschen von Model-Targets

11.2.8.1. Einleitung

Manche einst eingescannten Objekte werden möglicherweise nicht mehr vom Unternehmen benötigt, da keine Montage- beziehungsweise Installationsarbeiten dazu mehr ausgeführt werden. In diesem Fall soll es die Möglichkeit geben, bereits gespeicherte Model-Targets wieder zu löschen. Hierbei ist vor allem wichtig, dass nicht nur unser Datenbankeintrag, sondern auch das in der Datenbank von Vuforia gespeicherte Dataset entfernt wird.

11.2.8.2. Endpoint

Zum Löschen von Ressourcen ist im HTTP-Standard die Methode „DELETE“ vorgesehen. Aus diesem Grund lautet unser Endpoint „DELETE /api/model-targets/{uuid}“. Der Ausdruck „{uuid}“ beschreibt einen Pfad-Parameter, der die ID des zu löschenden Objekts repräsentiert. Allgemein sind Pfadparameter ein Teil der URL, der verwendet wird, um spezifische Informationen an einen Webserver zu übermitteln.

11.2.8.3. Umsetzung

Die mittels Pfad-Parameter übergebene ID der zu löschenden Entität wird in der REST-API im ersten Schritt dazu verwendet, um das gesamte Objekt zu erhalten. Hierbei wird wiederum eine Datenbankabfrage mittels Entity Framework und dem eigens implementierten dazugehörigen Service durchgeführt. Die erhaltene Entität enthält schließlich die ID des in der Datenbank von Vuforia gespeicherten Datasets. Dies wird im folgenden Codebeispiel im Attribut „`VuforiaId`“ abgebildet.

Der nächste Schritt ist, das Dataset über die Vuforia Model Target Web API zu löschen. Hierzu wird die erwähnte „`VuforiaId`“ benötigt, die das Dataset identifiziert. In einem HTTP-Request an den Vuforia-Endpoint „DELETE /modeltargets/advancedDatasets/{uuid}“ wird der Ausdruck „{uuid}“ durch die „`VuforiaId`“ ersetzt. Ein Beispiel für eine „VuforiaId“ wäre „`3799f84bf0c4437d8316c854a079be26`“. Diese ID entspricht grundsätzlich dem allgemeinen Standard für UUID, wie im RFC 4122[17] zu erkennen ist, allerdings wird sie in diesem Fall ohne Bindestriche dargestellt. Dies ist seitens Vuforia so vorgegeben.

Nach diesem erfolgreichen Lösch-Prozess wird der Eintrag im letzten Schritt aus unserer Datenbank gelöscht, wieder mittels Rückgriff auf das eigene zuständige Datenbank-Service. Verläuft auch dieser Vorgang fehlerfrei, so wird der entsprechende Status an den Aufrufer zurückgegeben.

[17] [INTERNET ENGINEERING TASK FORCE (IETF)]

11.3. Datenbank

11.3.1. Einleitung

Um alle Model-Targets samt der dazugehörigen Namen, Beschreibungen und Datasets – mittels der VuforiaId - speichern zu können, greifen wir auf unsere ebenfalls in Azure gehostete Microsoft SQL Server Datenbank zurück[18]. Dies ist eine relationale Database-Engine, die sich nahtlos in das Azure und Microsoft Ökosystem integriert.

Da in unserem Fall nur strukturierte Daten kleiner Quantitäten gespeichert werden und die Sicherstellung von Integrität und Konsistenz eine Anforderung darstellt, eignet sich die Verwendung des relationalen Modells am besten.

11.3.2. ERD

11.3.2.1. Begriffserklärung

Nachdem alle Sachverhalte in Bezug auf das Projekt mit unserem Auftraggeber geklärt worden sind, wurde im ersten Schritt des Datenbankentwurfes ein sogenanntes „ERD“ erstellt. ERD[19] steht für Entity Relationship Diagram. Der Name lässt bereits darauf schließen, dass es sich hierbei um ein Diagramm handelt, das alle Sachverhalte in der Datenbank abbilden soll. Konkret bedeutet das, dass alle benötigten Entitäten und wie diese zueinander in Beziehung stehen, dargestellt werden.

Es gibt verschiedene Notationen, wie diese Entitäten und Beziehungen abgebildet werden können. Die bekanntesten Notationen sind die Chen-Notation[20], und die Crow's Foot-Notation[21]. Im Projekt haben wir auf letztere zurückgegriffen.

In dieser Notation werden Tabellen und die dazugehörigen Attribute in Rechtecken abgebildet. Wie diese Tabellen zueinander abhängen, wird durch die Linien abgebildet. Die Kardinalitäten werden durch die Striche am Ende der Linie beschrieben.

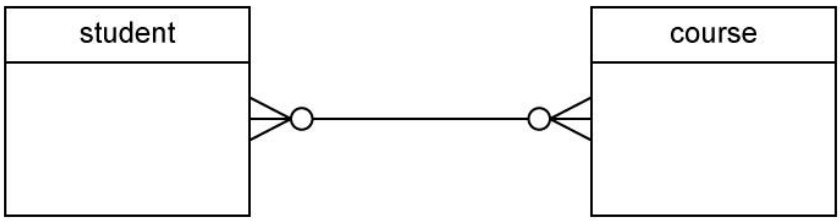


Abbildung 9: Crow's Foot Notation [VERTABELO SA]

[18] [MICROSOFT]
[19] [TECHTARGET, INC.]
[20] [BLOC SHOP]
[21] [BLOC SHOP]

11.3.2.2. Umsetzung

Auf folgendem Diagramm ist unser Datenbankentwurf abgebildet.

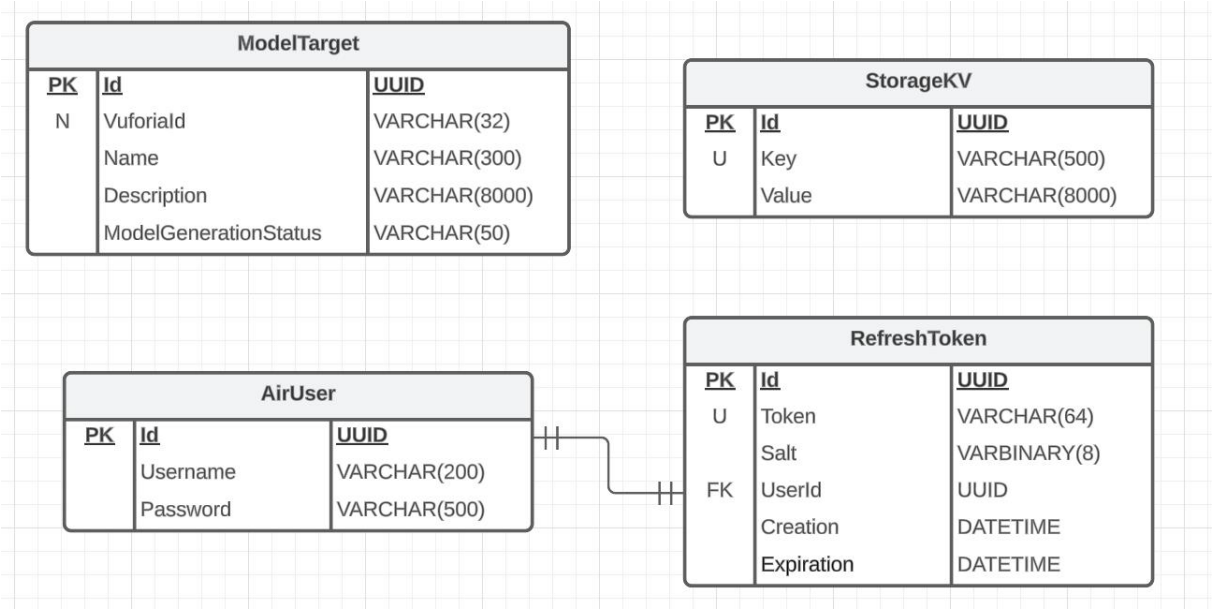


Abbildung 10: Datenmodell

Diese Abbildung schafft einen Überblick über die vorhandenen Entitäten und deren Zusammenhänge. Im Folgenden wird nun auf die einzelnen Bestandteile im Detail eingegangen.

11.3.3. Struktur im Detail

11.3.3.1. AirUser

Alle Benutzer des Systems sind in dieser Tabelle gespeichert. Hierbei ist anzumerken, dass die Benutzer zum Zeitpunkt der Entwicklung nicht bereits vorhanden sind, sondern von ITPRO zum Startzeitpunkt des Produktivbetriebes automatisiert eingespielt werden.

Die Entität besteht aus folgenden Attributen:

Name des Attributs	Datentyp	Beschreibung
Id	UNIQUEIDENTIFIER	Die „Id“ ist wiederum der Primärschlüssel dieser Tabelle und identifiziert jeden Benutzer daher eindeutig.
Username	VARCHAR(200)	Dieses Attribut speichert den Benutzernamen des Anwenders.
PasswordHash	VARCHAR(500)	Speichert den Hash-Wert (SHA256) des Passworts des Anwenders.

Diese Tabelle orientiert sich an den Vorgaben von ITPRO, beziehungsweise an der bereits existierenden, sich lokal am Gerät des Monteurs befindlichen SQLite Datenbank.

11.3.3.2. ModelTarget

Die Tabelle “ModelTarget” ist für die Speicherung der Model-Targets zuständig und enthält folgende Attribute:

Name des Attributs	Datentyp	Beschreibung
Id	UNIQUEIDENTIFIER	Dieses Attribut ist der Primärschlüssel der Tabelle, identifiziert also jeden Datensatz eindeutig. Dieses wird benötigt, um im Umgang mit unserer REST-API beispielsweise um einzelne Model-Targets zu löschen, zu bearbeiten, etc.
VuforiaId	VARCHAR(32)	Die „VuforiaId“ ist eine UUID, die das DataSet, welches über die Vuforia Model Target Web API aufrufbar ist, identifiziert. Dieses Attribut ist „NULLABLE“, kann also anfangs, wenn das Model-Target noch nicht hochgeaden wurde, keinen Wert beinhalten.
Name	VARCHAR(300)	Speichert den Namen des Model-Targets.
Description	VARCHAR(8000)	Stellt die Beschreibung des Model-Targets dar. Hier können Arbeitsschritte oder detaillierte Informationen gespeichert werden.
ModelGenerationStatus	VARCHAR(50)	Dieser Status kann entweder „DONE“, „PROCESSING“, „FAULTY“, oder „NOT_STARTED“ sein. Dadurch ist festzustellen, ob das 3D-Modell bereits aus den hochgeladenen Bildern generiert werden konnte, oder ein Fehler aufgetreten ist, etc.

11.3.3.3. RefreshToken

“RefreshToken” ist eine Tabelle, die dazu benötigt wird, um bei Ablauf des letzten gültigen Access-Token einen neuen anfordern zu können, damit ein sogenannter „Silent-Login“ entsteht und der Benutzer seine Anmeldedaten nicht erneut eingeben muss, damit er weiterhin Zugriff auf die REST-API besitzt. Konkret speichert die Tabelle den Refresh-Token pro Benutzer, um später bei dem Client-Versuch einen neuen Access-Token über den Refresh-Token anzufordern, zu überprüfen, ob dieser auch gültig ist.

Name des Attributs	Datentyp	Beschreibung
Id	UNIQUEIDENTIFIER	Die „Id“ ist der Primärschlüssel dieser Tabelle und identifiziert jedes Refresh-Token daher eindeutig.
TokenHash	VARCHAR(64)	Speichert den Hash-Wert (SHA256) des Refresh-Token. Über dieses Attribut (gemeinsam mit „Salt“) kann überprüft werden, ob der übergebene Refresh-Token valide ist.
Salt	VARBINARY(8)	Das Attribut „Salt“ speichert eine 8-Bit-Zeichenfolge, die dazu dient, die Sicherheit zu erhöhen. Konkret wird ein Salt dazu verwendet, um Rainbow-Table Attacken zu erschweren.
UserId	UNIQUEIDENTIFIER	Die „UserId“ ist ein Fremdschlüssel auf die Tabelle „AirUser“ und bestimmt daher, zu welchem Benutzer dieser Refresh-Token gehört.
Creation	DATETIME	Speichert den Erstellzeitpunkt des Refresh-Tokens.
Expiration	DATETIME	Speichert den Ablaufzeitpunkt des Refresh-Tokens. Wurde dieser bei einer „refresh“-Anfrage bereits erreicht, ist die Anfrage ungültig und es muss ein erneuter „Login“ mit den Anmeldedaten durchgeführt werden, woraufhin das Refresh-Token durch ein Neues ersetzt wird.

11.3.3.4. StorageKV

Diese Tabelle dient dazu, um etwaige Dinge wie die Zugangsdaten zum Vuforia-Account sicher speichern zu können. Im Wesentlichen ist diese Tabelle ein „Key-Value-Store“, also speichert nur einen Wert für einen Schlüssel pro Datenzeile.

Name des Attributs	Datentyp	Beschreibung
Id	UNIQUEIDENTIFIER	Die „Id“ ist der Primärschlüssel dieser Tabelle und identifiziert jeden Eintrag-eindeutig.
Key	VARCHAR(500)	„Key“ ist der Schlüssel des Eintrages. Über diesen wird auf den Wert zugegriffen, daher ist dieses Attribut „UNIQUE“, die gleiche Zeichenkette darf also nur einmal in der Tabelle vorkommen.
Value	VARCHAR(8000)	Legt den Wert des dazugehörigen Schlüssels fest. In diesem Attribut sind also die Nutzdaten des Dateneintrages, wie zum Beispiel der Vuforia Benutzername gespeichert.

11.4. Webapplikation für Administratoren

11.4.1. Einleitung

Aufgrund des hervorragenden Zeitmanagements konnte das Projekt noch um eine zusätzliche Webapplikation erweitert werden, die es den Administratoren von ITPRO ermöglicht, die gespeicherten Model Targets über eine intuitive Benutzeroberfläche zu bearbeiten und zu löschen. Dieses Tool war nicht im ursprünglichen Projektumfang enthalten, wurde aber im Projektverlauf hinzugefügt. Zur Umsetzung dieser Webapplikation haben wir das Web-Framework Angular[22] verwendet. Da diese Applikation eine Ergänzung zur eigentlichen Projektarbeit darstellt, wird sie in den folgenden Kapiteln oberflächlich beschrieben.

11.4.2. Aufbau

Die wesentlichsten Teile der Applikation sind folgende:

- Anmeldungsseite
- Übersichtsseite
- Bearbeitungsseite

[22] [GOOGLE LLC]

Wie es für das Framework Angular üblich ist, wurden diese einzelnen Seiten in eigene sogenannte Komponenten ausgelagert und sind mittels Angular Routing[23] ansteuerbar.

Die übergreifende Logik der Interaktion mit unserer REST-API wurde in eigenen Services implementiert, auf welche dann wiederum in den Komponenten zugegriffen wird.

11.4.3. Anmeldung

Um keinen unbefugten Zugriff auf die Ressourcen des Systems zu ermöglichen, ist, wie bereits in Kapitel 11.2.3 erwähnt, eine Authentifizierung notwendig. In der grafischen Benutzeroberfläche äußert sich diese durch eine „Login“-Seite, auf der der Benutzer seinen Username und sein Passwort eingeben kann, um sich nach dem Klick auf den „Anmelden“-Button zu authentifizieren.

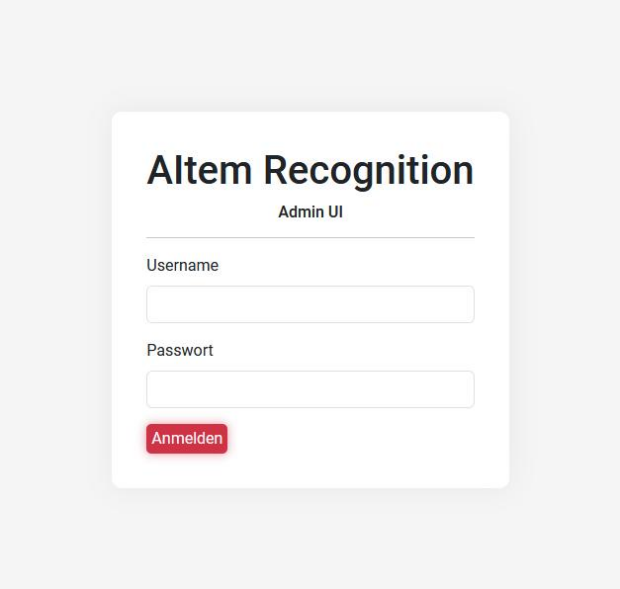


Abbildung 11: Admin UI Login

11.4.4. Anzeige der Model-Targets

Die Hauptseite der Webapplikation stellt die Anzeige aller gespeicherten Model-Targets dar. Hier können die Namen, Beschreibungen und auch Status aller Entitäten auf einen Blick angesehen werden. Außerdem befindet sich noch eine Anzeige des angemeldeten Benutzers und ein Button, mit dem man sich wieder vom System abmelden kann, rechts oben auf dieser Seite.

[23] [GOOGLE LLC]

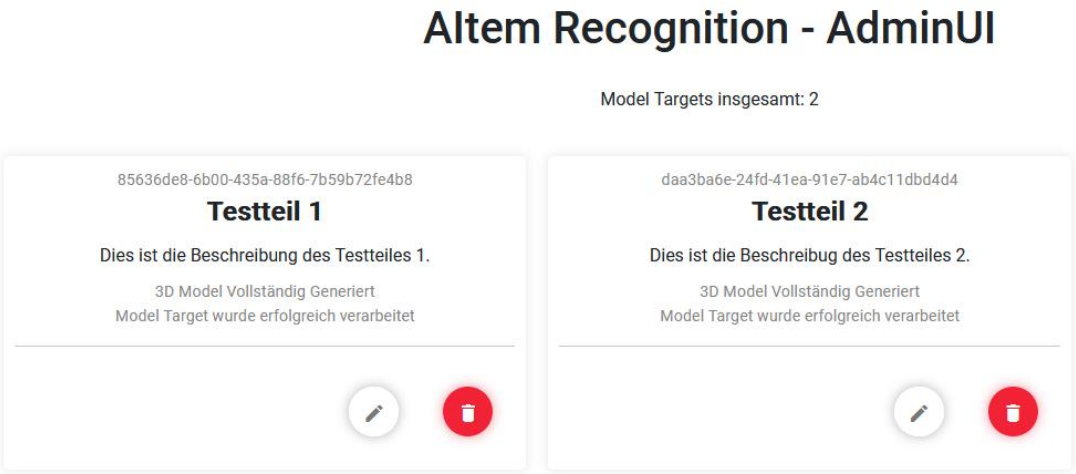


Abbildung 12: Admin UI Anzeige der Model-Targets

11.4.5. Bearbeiten

Der Bearbeitungsmodus eines Model-Targets kann durch einen Klick auf den Bearbeitungs-Button, welcher mit dem Bleistift-Icon gekennzeichnet ist, initiiert werden. Daraufhin öffnet sich ein Pop-Up, in welchem man den Namen und die Beschreibung ändern kann. Mit einem Klick auf den „Speichern“-Button werden die Änderungen an die REST-Schnittstelle gesendet, dadurch wird die Entität aktualisiert, und das Pop-Up schließt sich wieder, woraufhin die Model-Targets neu geladen werden und die Änderungen direkt auf der Übersichtsseite dargestellt werden.

Abbildung 13: Admin UI Model-Target bearbeiten

11.4.6. Löschen

Das Löschen eines Model-Targets funktioniert auf der Hauptseite durch einen Klick auf den roten Löschen-Button, der mit einem Abfalltonnen-Icon gekennzeichnet ist. Nach dem Löschen wird die Übersichtsseite erneut aktualisiert, sodass wiederum nur die jetzt gespeicherten Model-Targets angezeigt werden.

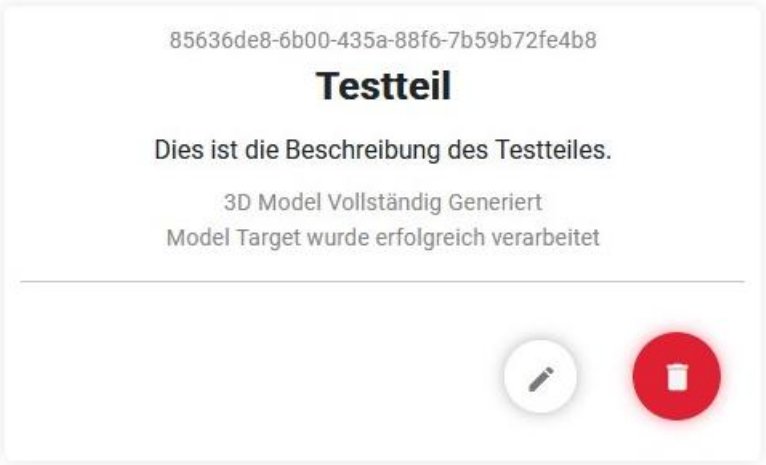


Abbildung 14: Admin UI Löschen des Model-Targets

11.5. Object Scanning

11.5.1. Einleitung

Grundsätzlich umfasst dieses Thema sowohl den Bereich von Rest-APIs, die Generierung der 3D Modelle und dem Front-End in Unity. Ein großer Part zur weiteren Erkennung der Objekte, spielt hierbei Vuforia und das Training der 3-dimensionalen Darstellung, welcher in Kapitel 11.2.5.5 behandelt wird. Allgemein kann der Workflow des Object-Scanning in folgender Abbildung 15 erkannt werden.

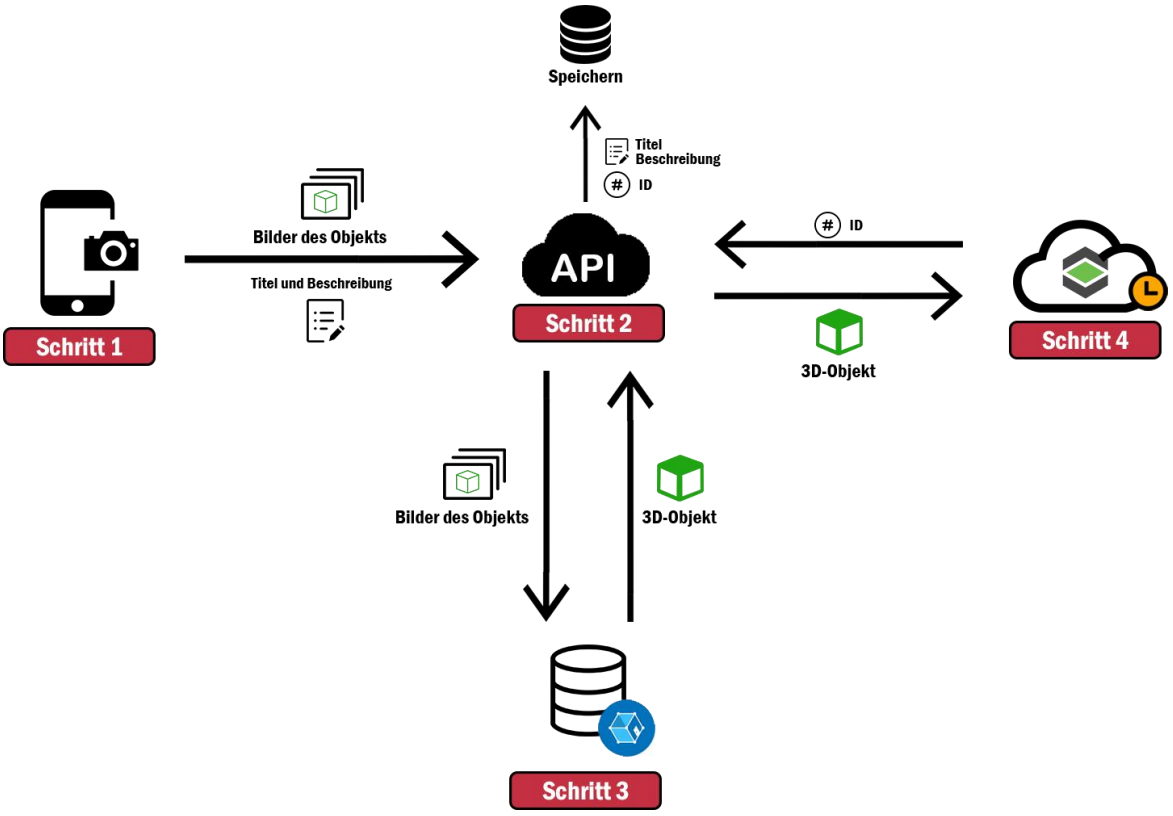


Abbildung 15: Object-Scanning Workflow Digramm siehe 18.2

1. In der Android App, welche mit der Spiel-Engine Unity programmiert wird, wird der Benutzer bzw. die Benutzerin dazu aufgefordert Bilder eines Objekts zu machen und einen Titel und eine Beschreibung zu vergeben.
2. Diese Bilder werden auf die Rest API (siehe 11.2) gesendet.
3. Anschließend werden sie auf die ausgelagerte Model-Generator-Rest-API weitergeleitet. Dort wird mithilfe der Open-Source 3D-Rekonstruktionssoftware „Meshroom“ ein 3D-Objekt aus den Bildern erzeugt. Die erzeugte .OBJ-Datei und deren Texturdateien wird in Form einer ZIP-Datei wieder auf die Rest-API gesendet.
4. Abschließend wird das Objekt an die Vuforia Model-Target API gesendet, hierbei wird sie trainiert, um nachher funktionsfähig zu sein. (siehe 11.2.5.5)

Nach Abschluss wird eine eindeutige Identifikationsnummer zurückgesendet, um das Objekt dem Titel und der Beschreibung zuzuordnen.

11.5.2. Object Scanning in Unity

Um ein Objekt zu erzeugen, muss der User oder die Userin vorerst das Objekt aus verschiedenen Winkeln fotografieren. In diesem Kapitel wird erläutert, welchen Prozess man als Benutzer oder Benutzerin der Applikation durchläuft, um schlussendlich die Generierung eines 3D-Modells anzustoßen.

11.5.2.1. Aufbau der Benutzeroberfläche

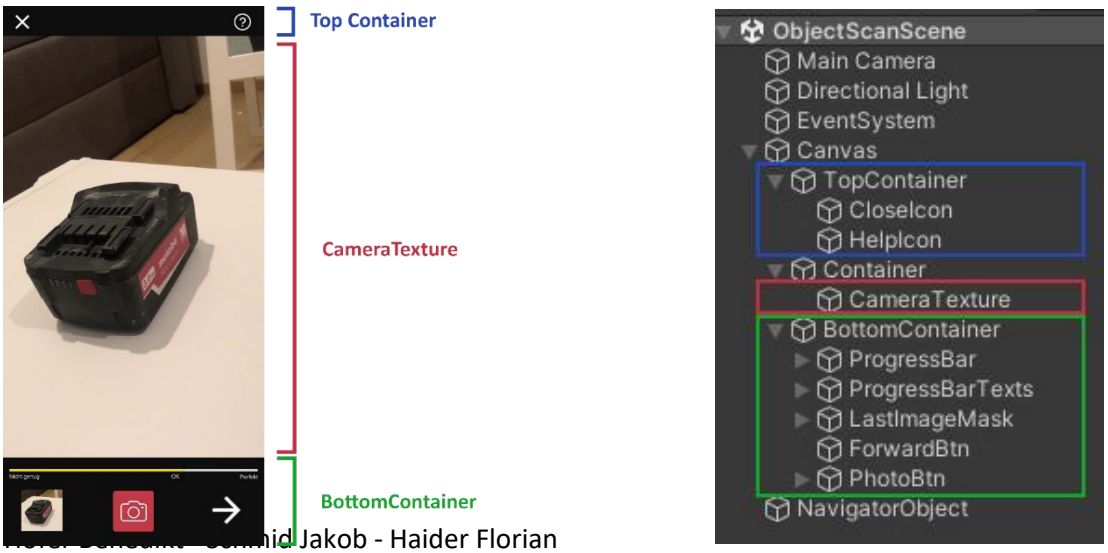
Grundsätzlich besteht eine Unity Applikation aus mehreren Szenen. So eine Szene ist eine virtuelle Umgebung, in der alle Elemente organisiert sind. Jede Szene repräsentiert eine spezifische Anwendungsumgebung. Durch die Nutzung von Szenen ermöglicht Unity den Entwicklern eine klare Strukturierung und Organisation ihrer Projekte.

In unserem Fall haben wir nur 3 Szenen, welche jeweils als UNITY-Datei gespeichert sind:

- **ObjectScanScene**: Hierbei handelt es sich um die Darstellung, die es dem Benutzer oder der Benutzerin ermöglicht die Bilder des Objektes zu machen
- **ImagesTakenScene**: Diese Szene macht es möglich bereits aufgenommene Bilder anzusehen und gegebenenfalls zu löschen
- **AddTitleAndDescScene**: genannte Szene kommt nach der Aufnahme der Bilder und bietet die Möglichkeit den Titel und Beschreibung des Objekts einzugeben. Anschließend werden die Daten an die Rest-API gesendet.

Die wohl wichtigste hierbei ist vermutlich die „**ObjectScanScene**“.

Eine Szene ist hierarchisch aufgebaut. Der Canvas kann hier als Leinwand beschrieben werden, auf der unsere UI-Elemente platziert sind. Die Szene kann grob in 3 Bereiche unterteilt werden. Links ist ein Screenshot aus der App abgebildet, rechts dazu sind die einzelnen Komponenten der Benutzeroberfläche abgebildet.



11.5.2.2. Darstellung der Kamera in Unity

Die wichtigste Komponente in unserer Szene ist die **CameraTexture**, hierbei wird die Kamera des Gerätes auf die Textur übertragen. Unity unterstützt dies bereits standardmäßig mithilfe der UnityEngine.WebCamTexture Klasse.

Grundsätzlich ist es möglich in einer Unity Szene ein Skript hinzuzufügen. Diese sind in der Programmiersprache C# geschrieben und enthalten standardmäßig folgende Methoden:

Update … Diese Methode wird in jedem Einzelbild (Frame) des Systems aufgerufen

Start … Wird aufgerufen, bevor die erste Update-Methode aufgerufen wird

Erstere ist für die Darstellung der Kamera essenziell. Denn in dieser Methode wird in regelmäßigen Abständen das derzeitige Bild der Geräte-Kamera auf den Hintergrund, also der CameraTexture, gelegt. Wird auf den Knopf zur Aufnahme eines Bildes gedrückt, wird die derzeitige Textur als Bild im JPG-Format gespeichert und gemerkt.

11.5.2.3. Übertragung der Bilder

Nach der Aufnahme der Bilder, kann der Benutzer oder die Benutzerin weitere Informationen zum Objekt angeben, siehe Abbildung 16. Bei dieser Grafik handelt es sich um die **AddTitleAndDescScene**-Szene, welche in Kapitel 11.5.2.1 erläutert wurde. Diese werden dann in weiterer Folge bei der Erkennung des Objektes, angezeigt (siehe 11.6).

Abbildung 16: Screenshot aus letztem Schritt des Scannens

Um mit der Rest API, welche eine Schnittstelle zum Hochladen der Objekte bereitstellt, kommunizieren zu können, muss der Benutzer in erster Linie authentifiziert werden. Laut unserem Auftraggeber befindet sich bereits auf dem Gerät eine lokale Datenbank, in welcher sich die benötigten Zugangsdaten (Benutzername und Passwort) befinden. Grundsätzlich existiert hierfür ein Endpunkt, und ein HTTP Request vom Typ POST wird gesendet. Nähere Informationen zur Authentifizierung findet man in Kapitel 11.2.3.

Erhält man nun diesen Token können im Skript der Unity Applikation alle Daten gesammelt werden und an den dafür zur Verfügung gestellten Endpunkt (siehe Kapitel 11.2.5) gesendet werden. Der zuvor erhaltene „Token“ (welcher für die Authentifizierung verwendet wird) kann mittels dieser Zeile am Request angefügt werden. Hierbei ist anzumerken, dass es sich bei dem Objekt „client“ um einen HttpClient aus der .NET-Umgebung handelt. Diese Klasse stellt uns vorgefertigte Methoden zur Verfügung, um http-Requests abzusenden.

```
client.DefaultRequestHeaders.Add("Authorization", "Bearer " + token);
```

Nachdem dieser angefügt wurde, können unsere Daten durch ein „multipart/form-data“ übertragen werden. Im Kapitel 11.2.5.2 wird dieser Typ genauer erklärt. Außerdem wird hierbei der Titel und die Beschreibung, welche vom Bediener bzw. der Bedienerin der Applikation eingegeben werden, folgendermaßen an ein Objekt vom Typ „MultipartFormDataContent“, welcher einfach den Inhalt dieses Requests beschreibt, angefügt.

```
formData.Add(new StringContent(title), "name");
formData.Add(new StringContent(desc), "description");
```

Die Variablen „title“ und „desc“ enthalten hierbei die zuvor getroffenen Eingaben. Es handelt sich um ein „Key-Value-Pair“. Das bedeutet, ein eindeutiger Key, in unserem Fall „name“ für den Titel und „description“ für die Beschreibung, identifiziert den Inhalt. Dieser ist ein „StringContent“, also eine Zeichenkette, welche die gewünschten Informationen enthält. Durch den festgelegten Schlüssel, können an der Rest API die eingegebene Information identifiziert werden, und somit aus diesem „Multipart-Form-Data“ wieder entnommen werden.

Um nun die aufgenommenen Bilder, welche als Bytes gespeichert sind, anzufügen, werden diese als „ByteArrayContent“ angefügt. Das bedeutet, dass jedes Bild aus einem Array, also einer Liste mit mehreren Bytes, besteht und diese für jedes einzelne, angefügt wird. Dieser Code wird für jedes aufgenommene Bild ausgeführt, die Variable imgName, ist hierbei lediglich eine 4-stellige Zahl, welche fortlaufend erhöht wird (0001.jpg, 0002.jpg, 0003.jpg etc.):

```
formData.Add(new ByteArrayContent(file), "file", $"{imgName}.jpg");
```

Der zuvor erwähnte Key, ist hierbei „file“.

Anschließend wird der Post-Request durch den http-Client abgesetzt und auf eine Antwort erwartet. Der Benutzer oder die Benutzerin bekommt während der Übertragung der Daten, Meldungen zu dessen Status angezeigt.

Ist die Übertragung erfolgreich, wird eine Erfolgsmeldung ausgegeben und man kann zurück zum Hauptmenü navigieren.

11.5.3. Generierung der 3D Objekte

Um aus den 2-dimensionalen Bildern ein 3D Objekt zu erhalten, verwendeten wir die Open-Source 3D Rekonstruktionssoftware Meshroom, welche das AliceVision Framework verwendet (siehe 9.1.8.3). „AliceVision ist ein quelloffenes Gemeinschaftsprojekt, an dem sich Vertreter aus der Industrie, wie auch aus dem akademischen Bereich beteiligen“. [24] Grundsätzlich bietet es viele Funktionen, für unsere Arbeit war der Teil der Fotogrammetrie am wichtigsten.

11.5.3.1. Fotogrammetrie

[24] [DAVID G. LOWE, 2004]

„Im wörtlichen Sinne beschreibt die Photogrammetrie die Erfassung präziser Messdaten aus Fotos. Konkret beinhaltet sie, sich überlappende Fotos eines Objekts, eines Gebäudes, einer Person oder einer Umgebung aufzunehmen und diese mittels verschiedener Algorithmen computergestützt in ein 3D-Modell zu überführen.“ [25]

In unserer Arbeit machen wir von dieser Funktion gebrauch, um so die Modelle zum Erkennen der realen Dinge zu erstellen. Hierbei wird der Benutzer in der Applikation am Handy aufgefordert Bilder von einem Objekt aus allen Winkeln zu machen. Somit kann ein Modell erstellt werden. Gehen wir einmal genauer auf den Prozess der Objekt-Erstellung ein.

Aufnahme der Bilder

Wichtig bei der Bildaufnahme ist die Überlappung der erfassten Fotografien des gewünschten Objektes. Dies ist deshalb essenziell, da der Fotogrammetrie-Prozess durch verschiedene Algorithmen erkennen muss von welchem Winkel die Aufnahmen stammen, um so ein realistisches Objekt generieren zu können.

Laut Formlabs[25] reicht eine Smartphone Kamera mit 8 Megapixel Auflösung bereits aus, um ein gutes Ergebnis erzielen zu können. Unser Kooperationspartner ITPRO verwendet intern für ihre Monteure, welche das System in späterer Folge auch verwenden, Handys des Herstellers „Zebra“. Das Modell TC22/TC27[26] zum Beispiel hat eine Kamera mit einer Auflösung von 16 Megapixel, welche völlig ausreichend ist.

Innerhalb der App wird der Benutzer dann aufgefordert mindestens 20, am besten noch mehr, Bilder aufzunehmen. Grundsätzlich gilt, je mehr Bilder, desto besser. Außerdem werden dem App-Benutzer oder der App-Benutzerin Tipps gegeben, um ein optimales Ergebnis erzielen zu können.

Nach der Aufnahme der Bilder, wird man aufgefordert einen Titel und eine Beschreibung einzugeben. Diese wird beim erneuten Scan des Objekts angezeigt und soll somit die wichtigen Informationen darüber enthalten.

Kamera Eigenschaften

Eine entscheidende Rolle bei der Erstellung der 3-dimensionalen Objekte sind die Metadaten der Kamera. Als Beispiel kann man in Abbildung 17 unter Kamera erkennen, dass dieses Bild mit dem Namen IMG_0145.JPEG mit einem IPhone X aufgenommen wurde. Darunter finden sich die Informationen, welche für die Fotogrammetrie wichtig sind.

An der Abbildung 17 ist zu erkennen, dass der Brennweite bei 4mm liegt, jedoch müsse dieser laut blende-zeit-iso[27] mit einem Faktor multipliziert werden, um die tatsächliche Brennweite herauszufinden. Dieser umgerechnete Wert kann unten bei „35mm Brennweite“ gefunden werden.

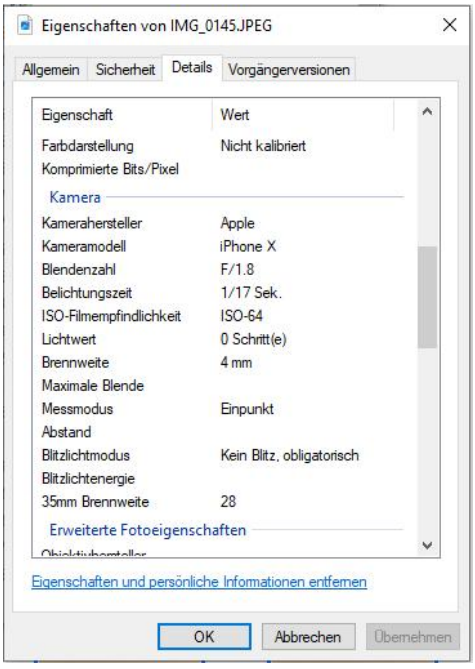


Abbildung 17: Eigenschaften eines JPEG-Bildes in Windows

[25] [FORMLABS GMBH]
[26] [ZEBRA TECHNOLOGIES CORP.]
[27] [BLENDE ZEIT ISO]

Wichtig ist diese Information deshalb, da wie in Abbildung 18 zu erkennen, ein Objekt unter Umständen verzerrt, dargestellt werden könnte, falls die Kamera eine kleine Brennweite besitzt. Grundsätzlich werden diese Parameter durch den Auto Fokus des Handys bestimmt.

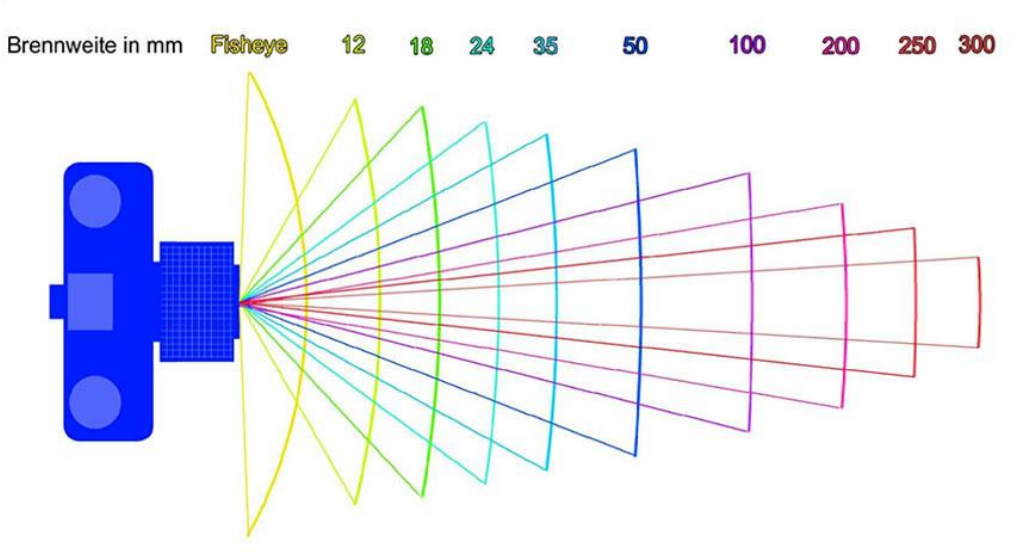


Abbildung 18: Brennweite Vergleich [SKETCH.MEDIA GMBH]

Um aus den 2D Bildern ein brauchbares 3D Objekt generieren zu können, versucht die Software, Verzerrungen zu korrigieren, indem die Brennweite herangezogen wird. Außerdem kann dadurch der Abstand zum jeweiligen Objekt berechnet werden.

Auch die Blendezeit und ISO Einstellungen zählen zu eine der wichtigsten Metadaten eines Bildes, da durch diese Metadaten die Qualität des Bilds beinflusst wird. Bilder welche für die Generierung eines 3D-Objektes verwendet werden, müssen auf jeden Fall qualitativ hochwertig sein. Wie die Generierung im Detail abläuft wird in nachfolgendem Kapitel (11.5.3.2) beschrieben.

11.5.3.2. AliceVision Meshroom

Wie dem Kapitel 9.1.8 zu entnehmen, fiel die Entscheidung unserer Technologie fiel auf Meshroom. Diese ist eine kostenlose Open-Source 3D-Rekonstruierungssoftware der Firma AliceVision[28]. Mehr Informationen zur Technologie können im Kapitel 9.1.8.3 **Fehler! Verweisquelle konnte nicht gefunden werden.** gefunden werden. Die Software funktioniert so, dass einzelne Funktionen abgekapselt sind und als Node dargestellt werden. Um das besser zu verstehen, schauen wir uns die Generierung eines 3D Objektes genauer an.

Grundsätzlich werden bereits vorgefertigte Pipelines angeboten, hierbei handelt es sich um eine feste Anordnung der erwähnten Nodes. Die „Photogrammetry Draft Pipeline“ besteht aus folgenden Knoten:

(1) CameraInit
(2) FeatureExtraction
(3) ImageMatching

[28] [ALICEVISION]

(4) FeatureMatching
(5) StructureFromMotion
(6) PrepareDenseScene
(7) Meshing
(8) MeshFiltering
(9) Texturing

Hierbei ist zu erwähnen, dass auch eine normale „Photogrammetry Pipelien“ exisitert, jene können wir jedoch nicht anwenden. Was genau das Problem hierbei ist, wird in Kapitel 11.5.3.4 „Einschränkungen im Projekt“ erläutert.

Was diese obigen Knotenpunkte genau tun sehen wir uns nachher an. Zu diesen Nodes sei gesagt, dass diese Daten produzieren und diese wiederum in andere als Eingabe Daten weiter gegeben werden können. Wie hier in der Abbildung 19 am Beispiel der Knoten „StructureFromMotion“ und „PrepareDenseScene“ zu erkennen. Im Node wurde eine „SfMData“ generiert, welche, nachdem dieser Knoten erfolgreich durchgelaufen ist, weitergegeben wird an den nächsten Knoten.

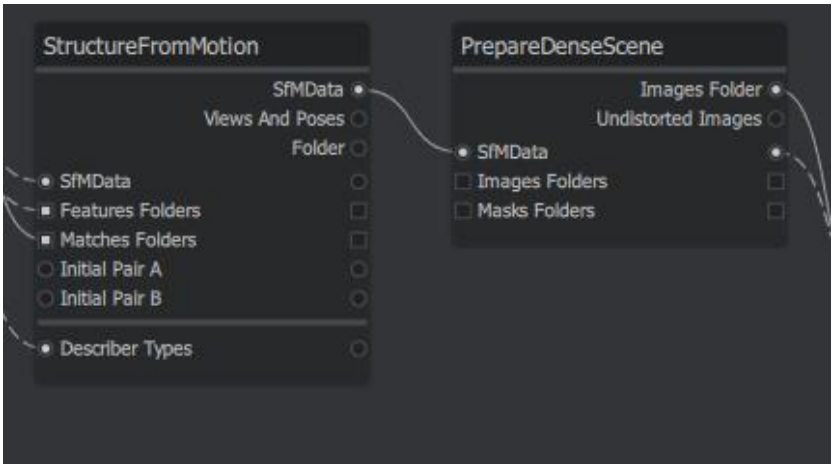


Abbildung 19: Nodes Ansicht im Meshroom User Interface

Der Vorteil so einer Architektur ist es, dass flexibel die einzelnen Funktionen ausgetauscht oder konfiguriert werden können. Weiters gibt es noch viele weitere Nodes um den Erstellungsprozess der 3D-Objekte nach spezifischen Anforderungen anzupassen. Auch wir machen von diesem Vorteile gebrauch und personalisieren diese Pipeline (nachzulesen in Kapitel 11.5.3.3 „Anpassungen an der Pipeline“)

Abbildung 20: Akku als Beispiel-Objekt

Sehen wir uns nun an, wie die einzelnen Funktionen der Pipeline zusammenarbeiten, sodass man nach Durchführung aller Schritte ein 3D Objekt erhält. Um die Schritte besser nachvollziehen zu können, werden wir versuchen einen Akku (Abbildung 20**Fehler! Verweisquelle konnte nicht gefunden werden.**) anhand von Bildern aus allen Winkeln, zu rekonstruieren:

(1) CameraInit

Diese Funktion lädt alle Eingabebilder. Durch Analyse der Bildmetadaten, werden alle verwendeten Kameras identifiziert. Grundsätzlich wird zwar empfohlen alle Bilder mit denselben Einstellungen, wie die Brennweite oder ähnlichem, zu machen, jedoch ist es trotzdem möglich verschiedene zu mischen, hierbei werden dann diese Bilder gruppiert nach Kamera.

Wenn wir nun das mit den Bildern des Akkus ausführen, wird eine „cameraInit.sfm“ Datei erstellt. In dieser finden wir zu jedem Bild einen Eintrag im JSON-Format, in der die Metadaten aufgelistet sind. Hier ist ein Eintrag für einen dieser eingegebenen Bilder:

```
{
    "viewId": "2139873408",
    "poseId": "2139873408",
    "frameId": "1092",
    "intrinsicId": "224489008",
    "path": "C:\/Users\/schmj\/Documents\/diplomarbeit\/akku\/01092.jpg",
    "width": "720",
    "height": "1280",
    "metadata": {
        "AliceVision:SensorWidth": "4.890000",
        "DateTime": "2024:03:02 14:59:21",
        "Exif:ApertureValue": "1.69599",
        "Exif:BrightnessValue": "1.97159",
        // ...
        "Model": "iPhone X",
        "ResolutionUnit": "none",
        "Software": "16.7.5",
        "XResolution": "72",
        "YResolution": "72",
        "jpeg:subsampling": "4:2:0",
        "oiio:ColorSpace": "sRGB"
    }
}
```

Das vierte Attribut mit dem Namen „intrinsicId“ verweist auf die Kamera, denn am Ende des Dokuments sind alle Kameras aufgelistet. Hier steht nur diese eine, da alle Bilder mit derselben aufgenommen wurden.

```
{
    "intrinsicId": "2139873408",
    "width": "1536",
    "height": "2048",
    "sensorWidth": "4.8899999999999997",
    "sensorHeight": "3.6674999999999995",
    "type": "radial3",
    "initializationMode": "estimated",
    "initialFocalLength": "5.333333333333333",
    "focalLength": "5.333333333333333",
    "pixelRatio": "1",
    "pixelRatioLocked": "true",
     // ...
}
```

(2) FeatureExtraction

Wie der Name bereits verrät, extrahiert er charakteristische Merkmale (Features) aus den Eingabebildern.[29] Durch solche Merkmale können die Bilder nachher richtig zugeordnet und identifiziert werden, von welchem Winkel, welche Bilder aufgenommen wurden. Der SIFT-Algorithmus (ausgeschrieben: **S**cale-**I**nvariant **F**eature **T**ransform) ist eine weit verbreitete Methode zur Merkmalsdetektion. „Der SIFT-Algorithmus beschreibt ein Verfahren, welches aus einem Bild vergleichbare rotations- und skalierungsinvariante Schlüsselpunkte findet, um diese in einem weiteren Verfahren auf anderen Bildern wiederzufinden, um somit Objekte mit bekannten Merkmalen auf anderen Bildern zu finden“[30]

In Abbildung 21 kann ein Beispiel dazu gefunden werden in dem dasselbe Objekt von verschiedenen Winkeln dargestellt wird. Hier werden diese Features gefunden und zugeordnet. So kann in späterer Folge die Kamera Position der beiden Bilder errechnen werden. Dieser Algorithmus ist ein wichtiger Bestandteil in der Generierung des 3D Objektes.

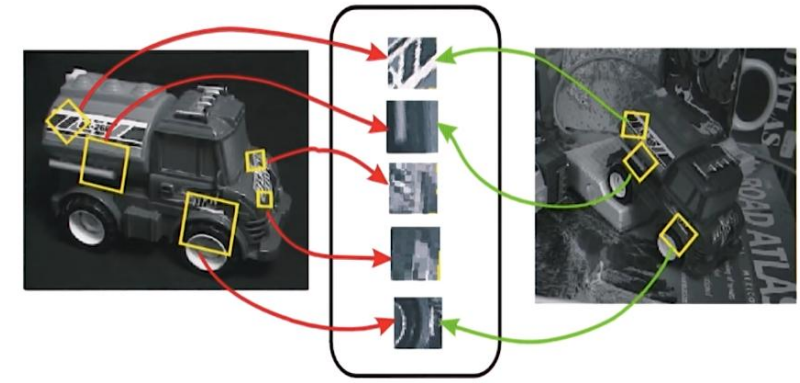


Abbildung 21: Extrahierte Features durch SIFT-Algorithmus [DAVID G. LOWE, 2004]

Grundsätzlich muss jeder sogenannte „Interest Point“[31] erkannt werden, auch wenn sie eine unterschiedliche Größe, Belichtung oder Orientierung haben. Wie in Abbildung 21 zu erkennen, ist das zweite Bild aus einer größeren Entfernung aufgenommen als das Erste. Außerdem sind die Punkte auch gedreht, im Vergleich zu ersterem und die Gemeinsamkeiten können trotzdem identifiziert werden. Außerdem ist bei der Berechnung dieser Punkte wichtig, dass auch wirklich markante Merkmale ausgewählt werden.

Diese Schlüsselpunkte werden auch „blob“ genannt. Um diese finden zu können werden sogenannte Oktaven-Ebenen aus dem Ausgangsbild, in unserem Fall dem Akku, generiert. Eine Oktave besitzt eine Skalierung und besteht aus ebenen, welche fortlaufend immer unschärfer sind. In Abbildung 22 kann erkannt werden, dass die Oktaven immer ein Viertel der Skalierung des vorherigen Bildes besitzen und somit immer kleiner werden. Die Ebenen werden durch den „gaußschen Weichzeichner“[30] immer mehr verschwommen.

[29] [ALICEVISION]
[30] [JÖREN CARSTENS UND HAUKE MARTENS, 2014]
[31] [NAYAR, Shree K., 2022]

Abbildung 22: SIFT-Ebenen einer Oktave

Hat man nun alle Oktaven und Ebenen berechnet, wird aus allen Ebenen der sogenannte „Difference of Gaussian“ gebildet. Dieser entsteht „[…] indem die Differenz der Grauwerte zweier benachbarter Ebenen gebildet wird.“[30] Wie in Abbildung 23 zu erkennen, entsteht darauf quasi ein 3-dimensionaler Raum (die gestapelten schwarzen Flächen), indem dann in Abbildung 24 Punkte gefunden werden können. Wie zu erkennen, hebt dieser Algorithmus die Kanten hervor. Grundsätzlich wird dieses Verfahren verwendet um eben solche Merkmale wie Kanten, Ecken und Texturänderungen in einem Bild zu identifizieren.

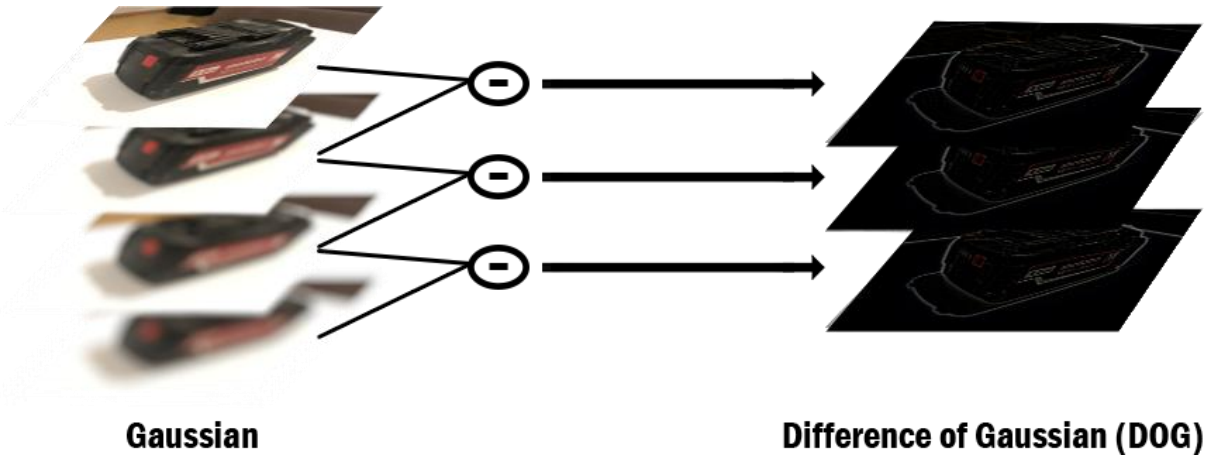


Abbildung 23: Berechnung Difference of Gaussian

Um nun die Extrempunkte und somit unsere „Feature Points“ zu erhalten wird für jeden Punkt auf den DOGs ein Vergleich durchgeführt. Da dieser Raum im DOG eine X, Y und Z-Achse besitzt, wird ein Pixel mit allen benachbarten verglichen, ob er den kleinsten oder größten Grauwert besitzt.

Das bedeutet, jeder Punkt wird mit den 8 benachbarten auf der eigenen Eben, und mit jeweils 9 Pixels von der oberen und unteren Ebene, verglichen (siehe Abbildung 24). Da die Extrempunkte, welche in eine der herunterskalierten DOGs (Difference Of Gaussian) gefunden werden, ja nicht genau die Pixel-Koordinate entsprechen wie die der, mit der vollen Auflösung, werden diese anschließen entsprechend hochskaliert, um an der richtigen Position zu sein.

Außerdem können nicht alle gefundenen Punkte verwendet werden. Denn es gelten bestimmte Kriterien, da ansonsten auch irrelevante Stellen gespeichert werden. Ein Kriterium hierbei wäre zum Beispiel ein zu geringer Kontrast.

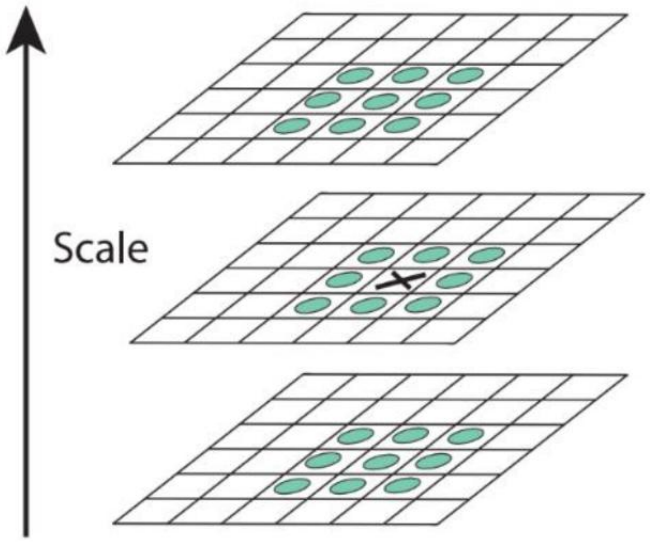


Abbildung 24: Identifizierung der Extrempunkte [DAVID G. LOWE, 2004]

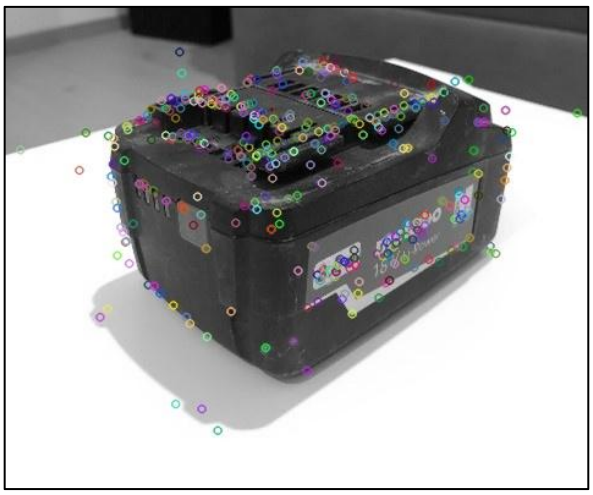


Abbildung 25: Erkannte Schlüsselpunkte des Akkus

In Abbildung 25 können die gefundenen Schlüsselpunkte erkannt werden. Dieses Bild wurde mit der Open Source Computer Vision Library „OpenCV“ erstellt.

Um jedoch nachher die Schlüsselpunkte wieder richtig zuordnen zu können, spielt auch die Rotation und Skalierung der Punkte eine wesentliche Rolle. Da diese auch erkannt werden müssen, wenn beispielsweise eine Aufnahme mit geringerer Distanz dazu gemacht wird und dadurch die Punkte größer sind als sonst. Ebenfalls sollte es kein Problem sein, wenn sich die Orientierung der Punkte ändert. Um diese Orientierung zu bestimmen, wird sich der Umkreis eines jeden Punktes angesehen und entschieden, in welche Richtung der Farbverlauf der Pixel verläuft.

Die genaue Winkelanzahl wird nachher in ein Histogramm, wie in Abbildung 26 zu sehen, eingetragen. Die Winkel werden in 10er-Schritten angegeben, überschreitet eines der Balken den Grenzwert von 80%, bestimmt es die Orientierung für dieses Feld. Sind mehrere Werte über der Grenze, werden alle diese angenommen.

Verläuft der Gradient beispielsweise in einem Winkel von 67°, wird er in dem Balken mit der Beschriftung 60-70 zugeordnet. Dieses Histogramm wird für den nachfolgenden Schritt benötigt.

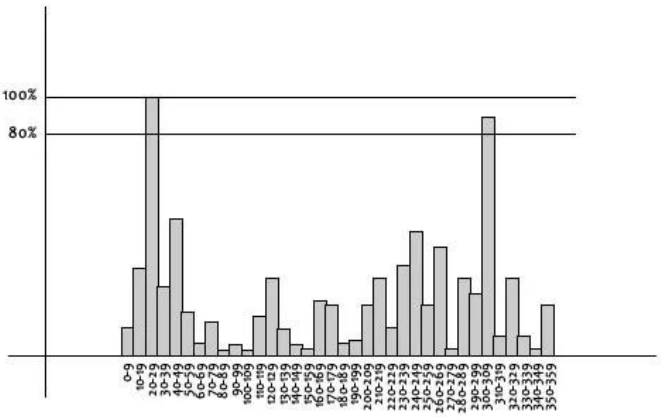


Abbildung 26: Histogramm der Orientierung der Schlüsselpunkte [JÖREN CARSTENS UND HAUKE MARTENS, 2014]

Dieser beinhaltet, dass dem Schlüsselpunkt ein sogenannter „Fingerprint“ gegeben wird, um ihn nachher mit anderen Punkten vergleichen zu können. Hierbei wird entlang der vorher bestimmten Rotation ein 16x16 Feld gezogen, wie in Abbildung 27 zu erkennen. Die blaue Linie stellt die zuvor bestimmte Orientierung dar. Auf dieser Abbildung ist zu erkennen, dass der markierte Bereich wieder in 16 Blöcken zu je 16 Pixeln eingeteilt wird. Die Orientierung dieser Pixel wird wieder in ein Histogramm eingetragen und gespeichert. Aus diesen Daten, welche in Form eines „Feature Vectors“[30] gespeichert werden, besteht schließlich der Fingerabdruck, wodurch in späterer Folge wieder so ein Punkt identifiziert werden

kann.

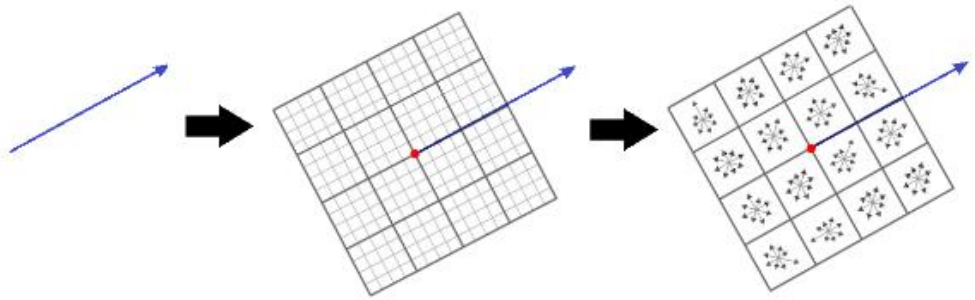


Abbildung 27: 16x16 Feld um den Schlüsselpunkt [JÖREN CARSTENS UND HAUKE MARTENS, 2014]

Wenn wir nun den SIFT-Algorithmus an unserem Bild aus Abbildung 20 anwenden erhalten wir folgendes Ergebnis:

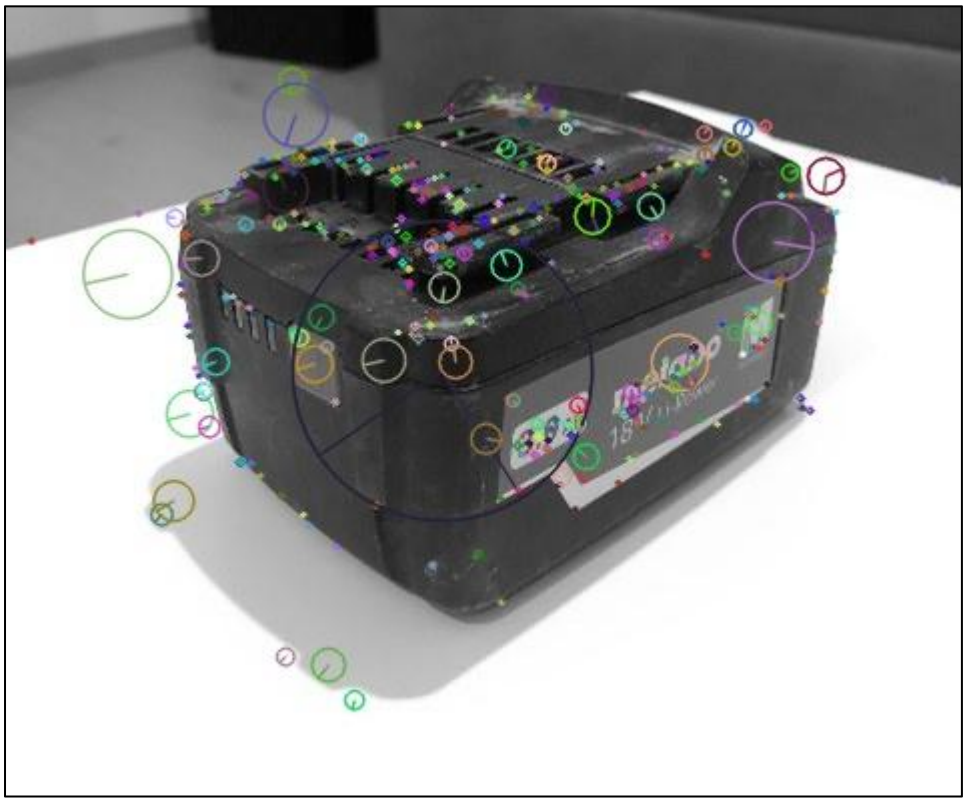


Abbildung 28: Akku nach Anwendung des SIFT-Algorithmus generiert durch „OpenCV“

Zu erkennen sind alle Punkte und deren Größe, außerdem kann auch die Orientierung, in Form eines Striches im Kreis, erkannt werden. Außerdem kann man feststellen, dass diese auch, wie bereits erwähnt, mehrere Richtungen besitzen können. Jeder einzelne „Feature Point“ besteht nun aus einem Vector mit 128 Dimensionen.

Somit ist die „Feature Extraction“ abgeschlossen und es kann mit den gefundenen Punkten und deren Fingerabdruck fortgesetzt werden beim nächsten Schritt.

(3) ImageMatching

Dies ist ein Vorverarbeitungsschritt, der herausfindet, welche Bilder sinnvoll sind, um sie miteinander abzugleichen. Das Ziel dieses Abschnitts besteht darin, Bilder zu finden, die auf ähnliche Bereiche der Szene fokussieren. Hierfür werden Techniken der Bildrückgewinnung genutzt, um Bilder zu entdecken, die gewisse Inhalte teilen, ohne alle Merkmalsübereinstimmungen im Detail auflösen zu müssen. Dies sparen einige Zeit, da somit nicht jeder Feature Point eines Bildes mit jedem, der anderen Bilder verglichen werden muss, sondern hier zuerst bestimmt wird, welche Aufnahmen, Inhalte einer anderen enthalten.

Laut Meshroom wird hier der Ansatz von David Nistér und Henrik Stewénius aus ihrer Veröffentlichung „Scalable Recognition with a Vocabulary Tree“[32] angewendet, um das Ergebnis erzielen zu können. Die bei Schritt (2) FeatureExtraction gefundenen Merkmale werden hierbei in ein sogenanntes „Codebook“ eingefügt. Es können einige der Merkmale auch zusammengefasst werden. Durch den sogenannten „K-Means Algorithmus“ werden die 128 Dimensionen eines „Feature Points“ in Cluster eingeteilt.

Der **K-Means Algorithmus** ist eine der am häufigsten verwendete Clustering-Algorithmus. Er zielt darauf ab, ähnliche Datensätze in solche Cluster, also Gruppen, zu organisieren und zusammenfassen. Außerdem ist es wichtig zu erwähnen, dass dieses Verfahren „unsupervised“ ist. Das bedeutet, dass keine Vorkenntnisse über die Datensätze nötig sind, um diese zuordnen zu können.

Zuerst wird eine Anzahl von Clustern definiert, welche am Ende herauskommen sollen. Da dieses Beispiel in Abbildung 29 etwas kleiner skaliert ist, um das Prinzip des Algorithmus zu verstehen, wählen wir eine Größe von 3 aus. Der ImageMatching Knotenpunkt wählt hierbei eine Zahl von 200.

Anschließend werden zufällig Punkte ausgewählt. So viele wie die zuvor bestimmte Clustergröße. Danach werden alle Punkte einem der ausgewählten zugeordnet, basierend darauf, wie nahe sie zueinander sind. (Schritt 2 aus Abbildung 29)

Anschließend wird für alle zusammengehörigen Gruppen der Mittelpunkt ermittelt und anschließend wieder mit Schritt 2 fortgefahren, so lange bis sich die gefundenen Gruppen nicht oder nur mehr ganz leicht ändern.

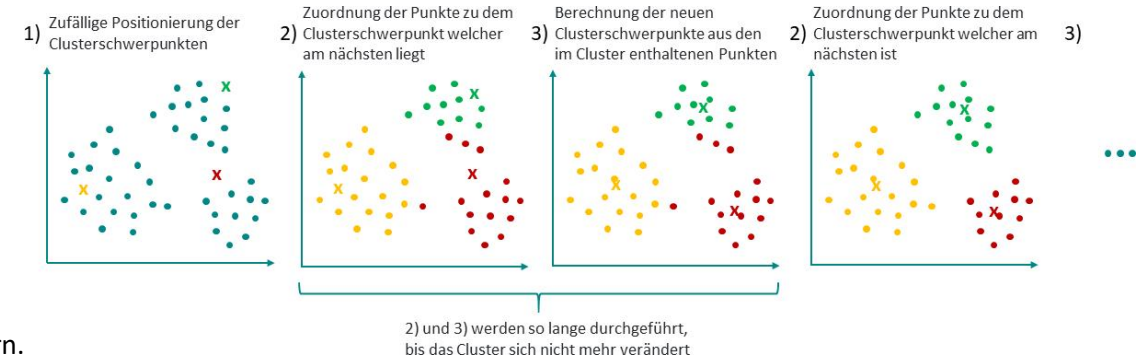


Abbildung 29: Ablauf des k-means Algorithmus [DATATAB E.U. GRAZ]

[32] [DAVID NISTÉR AND HENRIK STEWÉNIUS, 2006]

Beim ImageMatching-Node verläuft dies natürlich etwas komplizierter. Im obigen Beispiel haben wir nur eine x und eine y-Achse. Meshroom verwendet hierbei jedoch 128-Dimensionen. Da diese sehr schwer darstellbar sind, wurden im Beispiel nur 2 Dimensionen verwendet, um das Grundprinzip des Algorithmus einfacher verstehen zu können.

Nach Durchführung des Algorithmus erhalten wir 200 Cluster, in denen die Punkt eingeteilt wurden. Diese Cluster werden laut David Nistér und Henrik Stewénius auch „Visual Words“[32] also „visuelle Wörter“ zu Deutsch, genannt. Für jedes dieser wird nun ermittelt, wie oft sie in dem Bild vorkommen. Daraus kann ein Histogramm erstellt werden (siehe Abbildung 30). Hierbei kann abgelesen werden, welche dieser Cluster am häufigsten im Bilder vertreten sind.

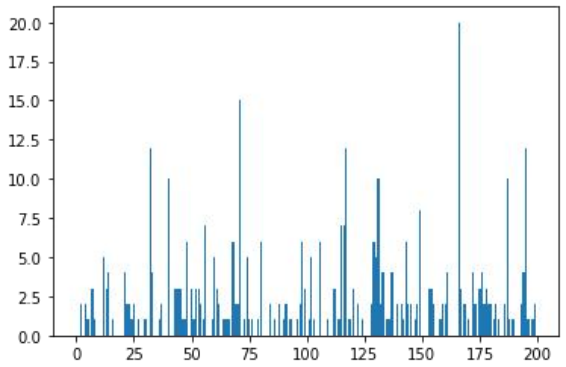


Abbildung 30: Histogramm der "Visual Words" [PINECONE SYSTEMS, INC.]

Durch einen Vergleich dieser im Diagramm dargestellten Bildmerkmale ist es möglich festzustellen, ob verschiedene Bilder denselben Inhalt teilen. Das Ergebnis dieses Nodes ist eine Datei mit dem Namen „imageMatches.txt“. Hierbei werden in jeder Zeile zusammengehörige Bilder, in Form ihrer ID dargestellt. Bedeutet also, dass alle Bilder einer Zeile, bestimmte Punkte des Objektes beinhalten. Somit können im nächsten Schritt die einzelnen Feature Points richtig zugeordnet werden.

(4) FeatureMatching

Hierbei werden die Zusammenhänge des Bildes, basierend auf die im Schritt (2) errechneten Fingerabdrücke, dem sogenannten „Feature Descriptor“[29], gefunden. Auch hier kommt der k-means Algorithmus zum Einsatz. Man erhält man von jedem Punkt eine Liste an Kandidaten. Grundsätzlich wird eine Grenze gesetzt, sodass falsche Punkte entfernt werden, falls diese eine zu große Distanz besitzen. Da der k-means Algorithmus bei höheren Dimensionen sehr rechenintensiv ist, kommt hierbei eine Abwandlung, der „Approximate Nearest Neighbor“, kurz „ANN“, zum Einsatz. Zur Erinnerung, bei uns handelt es sich um einen Vektor mit 128 Dimensionen.

Anstatt alle Datenpunkte im Raum zu durchsuchen, verwendet der „ANN“ eine Reihe von effizienten Datenstrukturen und Algorithmen, um die Suche zu beschleunigen, ohne dabei zwangsläufig den genauen nächsten Nachbarn zu finden. Es wird die „Best-Bin-First (BBF)“[33]-Implementierung des ANN angewandt. Hierbei wird mit hoher Wahrscheinlichkeit der „Nachbar“ gefunden, welcher am nächsten ist. Laut Lowe[24] werden bloß die nächsten 200 Kandidaten geprüft, danach wird es einfach abgebrochen. Er erwähnt außerdem, dass dadurch nur in weniger als 5% eine falsche Übereinstimmung gefunden wird. Dieser BBF eignet sich deshalb besonders gut für dieses Problem, da wir nur Übereinstimmungen in Betracht ziehen, bei denen der nächstgelegene Nachbar weniger als 0,8-mal die Entfernung zum zweitnächsten Nachbarn beträgt. Somit besteht keine Notwendigkeit, die schwierigsten Fälle genau zu lösen, in denen viele Nachbarn sehr ähnliche Entfernungen haben.

[33] [JEFFREY S. BEIS AND DAVID G. LOWE, 1997]

Da der Algorithmus auch fehlerhafte Werte oder Ausreißer enthält, müssen solche auch beseitigt werden. Im Rahmen des „Feature Matchings“ wird laut Meshroom Dokumentation[29] der „Random Sample Consensus“, kurz „RANSAC“, angewandt. Dieses Framework lässt uns Merkmalspaare suchen, die der oben erwähnte, „BBF“-Algorithmus als Zusammengehörig identifiziert hat, welche fehlerhaft zugeordnet wurden. Um das zu tun, werden zufällig ein paar Merkmale ausgewählt und versucht, eine grundlegende Matrix zu erstellen. Diese ist dann eine Art Vorhersage, wie die Merkmale in den beiden Bildern miteinander verbunden sind. Danach wird überprüft, wie viele Merkmale wirklich zu dieser Vorhersage passen. Wenn die meisten Merkmale übereinstimmen, wird diese Vorhersage als richtiges Paar verwendet. Wenn nicht, wird es mit einem anderen Satz von Merkmalen probiert und der Prozess wird wiederholt. So wird sichergestellt, dass nur gültige Merkmale gefunden und falsche aussortieren werden. Wenn eine passende Vorhersage getroffen wurde, werden alle, die nicht dazu passen, entfernt.

Um die Zuordnung der Punkte veranschaulichen kann man in Abbildung 31 die beiden Eingabe-Bilder, welche von einem Unterschiedlichen Winkel aufgenommen wurden, sehen und wie die deren Feature Points, durch den Algorithmus verbunden wurden.

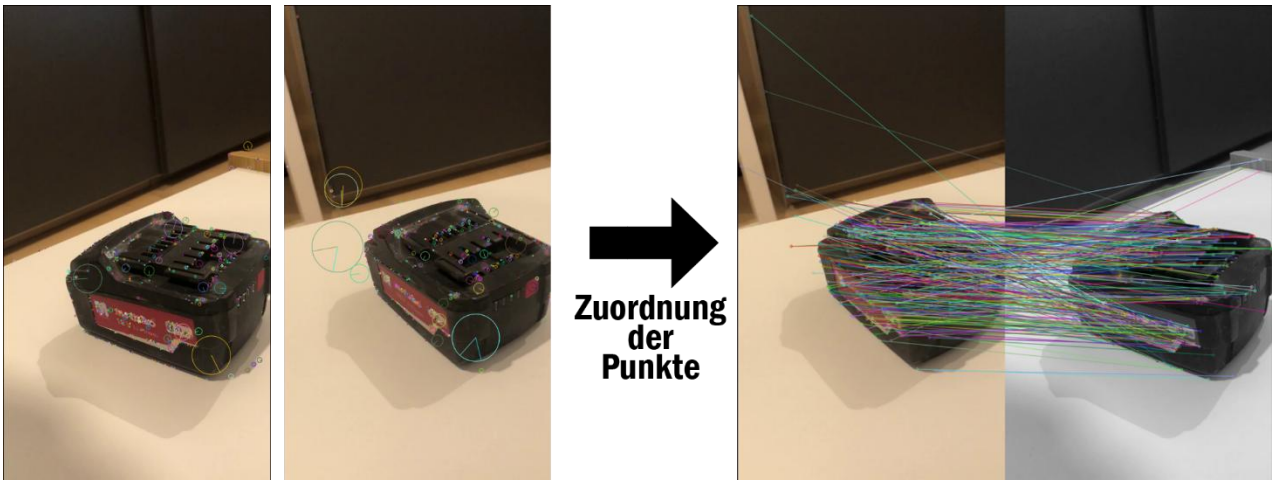


Abbildung 31: Ergebnis des Feature Matching durch „OpenCV“ anhand des Akkus

Als Ergebnis erhalten wir eine TXT-Datei, welche alle zusammenpassenden Feature Points beinhaltet. Diese können im nächsten Schritt weiterbearbeitet werden.

(5) StructureFromMotion

Hier werden 3D-Punkte aus den Eingabebildern generiert. Ein sogenanntes „Point Cloud“, also Punkte-Wolke, entsteht, woraus in späterer Folge das finale 3D-Objekt gerendert werden kann. Durch die vorher in Schritt (4) gefundenen zusammengehörigen Feature Points können die Kameras rekonstruiert werden. Das bedeutet es wird für jedes Bild errechnet, aus welcher Position dieses aufgenommen wurde.

Grundsätzlich startet dieser Prozess damit, dass laut „Meshroom Dokumentation“[29] alle Punkte, welche von mehreren Bildern geteilt werden, in einem „Track“ zusammengefasst werden. Der

Algorithmus sucht nun das optimale Bilderpaar aus, diese werden dann als Ausgangsbilder herangezogen. Die Auswahl soll folgende Kriterien erfüllen:

- Übereinstimmung mehrere Feature Points
- Großer Winkel zwischen den Bildern *(nicht direkt nebeneinander aufgenommen)*

Bei unserem Beispiel, dem Akku, kann in den Logs folgender Eintrag gefunden werden (Ausgewählte Bilder siehe Abbildung 32):

```
[19:48:24.303381][info] Initial pair is:
    - [A] view id: 873990423, filepath: C:/Users/schmj/Documents/diplomarbeit/akku/00312.jpg
    - [B] view id: 963811534, filepath: C:/Users/schmj/Documents/diplomarbeit/akku/00372.jpg
```

Abbildung 32: Initial Pair bei "Structure From Motion" Generierung

Das Ziel ist es nun die Position, an der das zweite Bild erfasst wurde, in Relation zu dem ersten Bild zu finden. Mithilfe der „Epipolargeometrie“ kann diese Beziehung zu einem 3D Punkt gefunden werden, dargestellt in Abbildung 33.

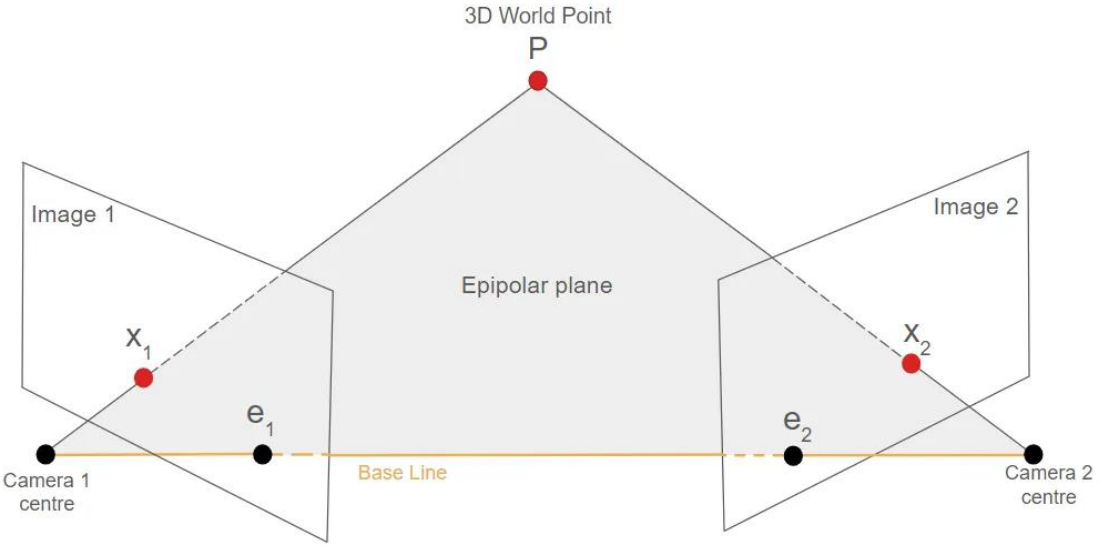


Abbildung 33: Abbildung der Epipolargeometrie zweier Bilder zu einem 3D Punkt [LOBO, Teresa]

Mithilfe dieser Informationen kann eine „fundamentale Matrix“ erstellt werden. Diese wird, laut Teresa Lobo[34], dazu verwendet um alle Punkte, aus den zwei Bildern, mit denen aus den verschiedenen Ansichten, zu verknüpften. Das bedeutet hier wird festgehalten, wenn Punkte auch in anderen Bildern vorkommen. Um anhand dieser, die Epipolargeometrie (siehe Abbildung 33) der Bilder zu bestimmen. Somit wird deren Position in der 3-dimensionalen Umgebung erfasst.

Danach wählen wir alle Bilder aus, die ausreichend den Merkmale des Ausgangspaares teilen, welche bereits rekonstruiert wurden. Basierend auf diesen 2D-3D-Assoziationen führt er eine sogenanntes „resectioning“[29] für jede dieser neuen Kameras durch. Dieses „resectioning“ ist ein Perspective-n-Point-Algorithmus (PnP) in einem RANSAC-Framework, um die Pose der Kamera zu finden, die die meisten Merkmale zuordnen kann.

Der „Perspective-n-Point-Algorithmus“ zielt darauf ab, die Position und Orientierung einer Kamera zu schätzen, die eine Szene beobachtet, wenn eine bestimmte Anzahl von 3D-Punkten in der Szene und ihre 2D-Projektionen in einem Bild bekannt sind. Das RANSAC-Framework wird, wie in (4) FeatureMatching, verwendet um Ausreißer und Bilder, die nicht zugeordnet werden können, zu identifizieren und anschließend zu entfernen.

Außerdem wird nach dem Hinzufügen einer neuen Bild-Position, die Positionen der anderen verfeinert, um am Ende alle Kameras möglichst genau rekonstruieren zu können. Grundsätzlich besteht dieser Prozess also aus folgenden Schritten:

- Bildung der Tracks
- Epipolargeometrie
- PnP-Algorithmus
- Verfeinern

[34] [LOBO, Teresa]

Dieser wird so lange iteriert, bis keine neuen Ansichten mehr gefunden werden können.

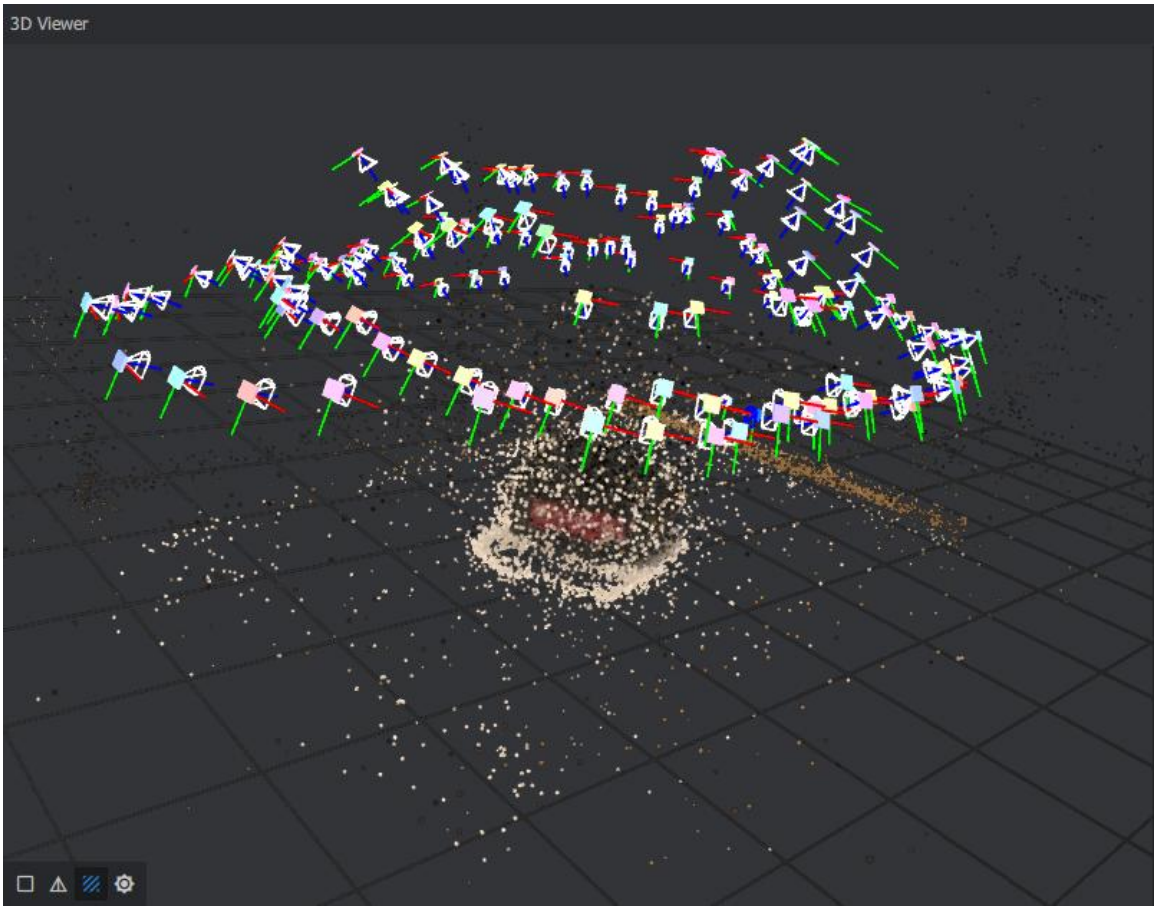


Abbildung 34: Darstellung der Kameras nach Structure From Motion im Meshroom 3D-Viewer

In der obigen Abbildung 34 kann man die rekonstruierten Kameras erkennen und deren errechnete Position. Außerdem sind die Punkte mit welchem die Position bestimmt werden konnte in einer „Point-Cloud-Darstellung“ visuell abgebildet. Hier kann man schon leicht unser Eingabebild, den Akku, erkennen. Nachher wird dieses Bild noch verfeinert und ein echtes 3D-Modelle entsteht.

(6) PrepareDenseScene

Dieser Schritt entzerrt die Bilder, was bedeutet, dass er Verzerrungen korrigiert, die durch die Linse der Kamera entstanden sind. Diese Verzerrungen können dazu führen, dass gerade Linien gekrümmt oder Objekte verzerrt erscheinen. Durch die Entzerrung werden die Bilder so angepasst, dass sie dem ursprünglichen Sichtfeld der Szene entsprechen, was zu einer präziseren Darstellung führt.

Außerdem werden die Bilder in das EXR-Format[35] konvertiert, das speziell dafür entwickelt wurde, hochdynamische Bilddaten und zugehörige Metadaten genau und effizient zu speichern, wobei Unterstützung für verschiedene Kanäle und Teile bereitgestellt wird. Somit werden hochwertige Bilder mit genauen Farb- und Helligkeitsinformationen erzeugt, die für weitere Bildverarbeitungs- oder Analysezwecke verwendet werden können.

[35] [OPENEXR]

(7) Meshing

Hierbei wird das vom Schritt (5) StructureFromMotion, erstellte Point Cloud (siehe Abbildung 34) in ein sogenanntes Mesh umgewandelt. Ein Mesh ist schon das gewünschte 3D Modell. Jedoch besitzt dieses noch keinerlei Textur oder ähnliches, aber die Form unseres Objektes schon zu erkennen.

Hierbei kommt eine Technik namens „3D Delaunay-Triangulierung“ zum Einsatz. Das Ziel ist es „[…] aus einer Punktmenge [siehe Abbildung 34] ein Dreiecksnetz [das Mesh] zu erstellen.“[36] Das bedeutet also, dass die Punkte sinnvoll verbunden werden müssen, um so Oberflächen darzustellen.

Abbildung 35: Point-Cloud des Akkus im Meshroom 3D-Viewer

Um das Prinzip der Technik verstehen zu können schauen wir uns eine Möglichkeit an wie man diesem implementiert. Hierbei verwenden wir einen kleineren Datensatz von gerade einmal 4 Punkten, um diesen zu demonstrieren.

Hierbei wenden wir den „Bowyer-Watson Algorithmus“ an Wie bereits erwähnt haben wir nur einen Datensatz von vier Punkten, welche wir durch Dreiecke verbinden wollen. Folgende Schritte müssen angewandt werden:

1. Erstellen eines „Super-Dreickes“ welcher alle Punkte umschließt (siehe Abbildung 36)
2. Auswählen eines zufälligen Punktes, falls sich dieser in einem Dreieck befindet, verbindet man die Ecken dieses mit dem Punkt (so entstehen 3 weitere Formen)
3. Nun sieht man sich die Dreiecke an und zeichnet den Umfang dessen, falls sich innerhalb dessen ein Punkt befindet, wird dieses Dreieck in weitere Formen aufgeteilt
4. Falls in Schritt 3 eine Aufteilung passiert ist, wird Schritt 3 mit allen wiederholt, bis sich keine Punkte mehr in der Fläche befinden.

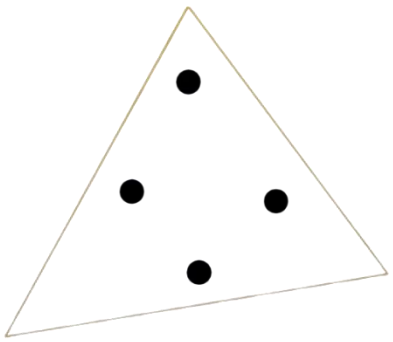


Abbildung 36: Delaunay-Triangulierung Super-Dreieck

Somit geht man hierarchisch vor und verbindet alle Punkte letztendlich durch die Wiederholung von Schritt 3.

Das hervorgehende Mesh wird laut Meshroom Dokumentation[29] noch durch einen sogenannten „Graph Cut Max-Flow“-Algorithmus, welcher von Yuri Boykov and Vladimir Kolmogorov entwickelt wurde, und einer Methode von Michal Jancosek, optimiert. Unter anderem werden dadurch auch irrelevante Punkte herausgefiltert.

Somit entsteht das erste richtige 3D-Modell, als Ausgabe erhalten wir bereits eine .OBJ-Datei. Diese beinhaltet das Objekt aus Abbildung 37. In der Abbildung ist ebenfalls ein langer Balken im

[36] [GISWIKI]

Hintergrund zu erkennen. Dieser wird in Schritt (8) MeshFiltering entfernt. Er resultiert daher, dass sich bei der Aufnahme der Bilder, im Hintergrund eine Holzleiste befindet. Diese ist beispielsweise auch in Abbildung 32: Initial Pair bei "Structure From Motion" Generierung zu erkennen. Beim Objekt rechts ist zu erkennen, dass es sich um den Akku handelt, dieser wird in den letzten Schritten noch besser Erkennbar.

Abbildung 37: 3D Objekt nach "Mesh" Node

(8) MeshFiltering

In dem vorletzten Knotenpunkt der Modell-Generierung werden unnötige Elemente des Objektes entfernt. Hierbei wird sichergestellt, dass nur das größte Mesh behalten wird. Das bedeutet, dass nur die größte „Ansammlung“ an Dreiecken weiter geschickt wird zur nächsten Node. Wenn also wie in unserem Beispiel in Abbildung 37 der Streifen im Hintergrund, weniger Dreiecke besitzt, was er definitiv hat, wird dieser verworfen.

Auf derselben Abbildung ist zu erkennen, dass ein paar entkoppelte Formen sich unterhalb des Objekts auf der rechten Seite befinden. Jene werden ebenfalls entfernt um ein sauberes 3D-Modell zu erhalten.

(9) Texturing

Der abschließende Schritt besteht darin die Textur an das Objekt anzufügen. Das bedeutet, dass es nicht wie in Abbildung 37: 3D Objekt nach "Mesh" Node keine Farbe besitzt, sondern quasi so aussieht wie in echt. Hierbei kommen die in Schritt (6) PrepareDenseScene konvertierten „EXR-Dateien“ zum Einsatz, um das Aussehen der Oberflächen zu bestimmen.

Laut Meshroom Dokumentation[29] wird hier der Ansatz von Adam Baumber aus seinem wissenschaftlichen Paper „Blending images for texturing 3D models“[37] verfolgt, um die Texturen auf das drei-dimensionale-Objekt anzuwenden.

Die grundsätzliche Idee ist es, für jede Kamera eine Gewichtung zu errechnen. Hierbei wird jedes Bild, in Frequenzbänder unterteilt und jedes Band auf die Fläche der Texturen, gemeinsam mit seiner Gewichtung, gelegt. Auf diese wird dann für jeden Pixel einzeln, basierend auf der Gewichtung, verschiedene Filter angewandt.

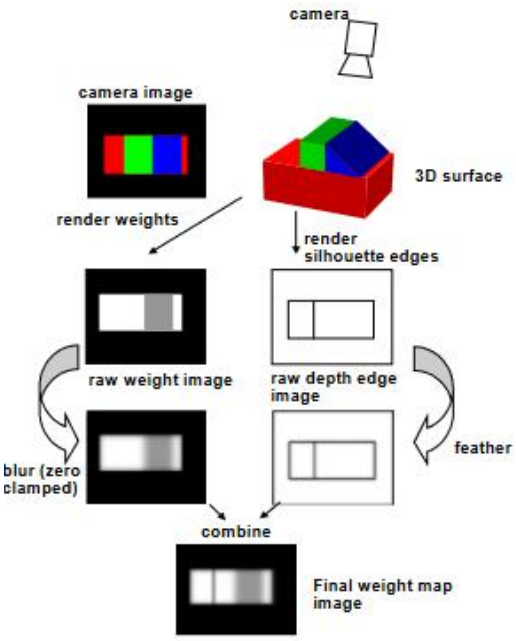
Das Wichtigste an diesem Algorithmus ist wohl das Gewichtsbild der Kamera. Diese wird für jeden in Schritt (7) „Meshing“ erstellten Dreieck einzeln berechnet. Diese Formel ist recht einfach, und zwar wird einfach ausgerechnet, wieviel der Fläche des Dreiecks auf der Kamera zu sehen ist, basierend auf wie groß die Fläche insgesamt ist. Also ergibt sich folgende Formel[37]:

$$w = \frac{area\ of\ projected\ triangle}{surface\ area\ of\ triangle}$$

- Als nächsten Schritt um ein Gewichtsbild zu erhalten, wird jedes Dreieck in das Eingabebild gerendert, mit einer Intensität, welche dem Gewichtsfaktor „w“ entspricht.
- Als letzten Schritt wird das Bild noch geglättet mittels dem Gaußschen Weichzeichner (siehe (2) FeatureExtraction).

Nun haben wir also den sichtbaren Teil des Eingabebilds geglättet und in der Gewichtungsabbildung berücksichtigt. Um die unsichtbaren Teile, also welche die auf der Kamera nicht zu erkennen sind, sich aber unter oder hinter dem auf dem Bild sichtbaren befinden, berücksichtigen zu können muss das Ausgangsbild wieder verwendet werden, um folgende Schritte durchzuführen:

- Erstellen einer Silhouette des Bildes, um Kanten zwischen den Dreiecken erkennen zu können
- Rendern der unverdeckten Silhouetten Kanten als schwarze Linien auf weißem Hintergrund)
  Position dessen, werden dem 3D Modell entnommen
- Glätten des Bildes (in der Abbildung wird der Prozess feather genannt und enthält einige Zwischenschritte[37])



In Abbildung 38 wird der Prozess zur Erstellung der Gewichtungsfläche, basierend auf den obigen Schritten, illustriert.

Abbildung 38: Generierung der Gewichtungsfläche [BAUMBERG, Adam, 2002]

Um nun die Textur auf das Bild anzuwenden, müssen folgende Schritte durchgeführt werden:

- Für jedes Eingabebild wird die Gewichtsfläche wie zuvor beschrieben generiert
- **Generieren des Bildes in niedriger Frequenz**
  weichzeichnen durch Gaußschen Weichzeichner, siehe (2) FeatureExtraction
- **Generieren des Bildes in hoher Frequenz**
  subtrahieren des niedrig frequenten Bildes vom Originalen
- Anwenden eines Durchschnittsfilter basierend auf dem Gewichtsfaktor beim Bild mit **niedriger Frequenz**
  Verwendung eines nichtlinearen Filters mit einem festgelegten maximalen Gewicht beim Bild mit **hoher Frequenz**
- Kombinieren der beiden Frequenzen

Dadurch kann für das gesamte Objekt eine Textur generiert werden, welche dann auf das 3D-Modell gelegt wird.

Als Ergebnis für unseren Akku erhalten wir das Modell aus der Abbildung rechts. Warum die Generierung nicht komplett einwandfrei läuft, und kein makelloses Objekt erstellt wird, wird in Kapitel 11.5.3.4 „Einschränkungen im Projekt“ klargestellt. Trotzdem ist der Akku durch die Textur und der Form klar erkennbar.

Als Ergebnis erhalten wir 3 Dateien:

- .OBJ – diese stellt die Form des Modells dar, also ist das wirkliche 3D Objekt
- .PNG – legt die Textur fest
- .MTL – Informationen wie z.B. Farbe, Glanz, Reflexion oder Transparenz

11.5.3.3. Anpassungen an der Pipeline

Um eine verlässliche Modell-Generierung gewährleisten zu können werden wir die Standard-Pipeline etwas modifizieren. Anschließend wird diese als Meshroom-Datei (.md) exportiert und für die Generierung verwendet, siehe dazu den `templatePath` in Kapitel 11.5.4 unter (1) POST /api/Generator.

Grundsätzlich bleiben die Knotenpunkte bestehen, jedoch werden an folgenden Punkten Einstellungen getroffen:

(2) FeatureExtraction
Hierbei wird die „Describer Density“ und die „Describer Quality“ von „normal“ auf „high“ gesetzt. Dadurch werden mehr Feature Points aus den Bildern extrahiert, besonders bei kleinen Datensätzen ist dies wichtig, da ansonsten unter Umständen zu wenige Gemeinsamkeiten zwischen den Bildern gefunden werden können.

(4) FeatureMatching
Hierbei aktivieren wir die Funktion „Guided Matching“. Dies bewirkt, dass die Features zwischen den Bildern noch ein zusätzliches Mal durchgelaufen werden, um Gemeinsamkeiten zu erkennen. Somit wird das Ergebnis-Objekt genauer.

(8) MeshFiltering
Hierbei wird die Option „Keep Only The Largest Mesh“ eingeschaltet. Die Auswirkung der Einstellung wird oben bei der Beschreibung des Knotenpunktes genauer erläutert.

(9) Texturing
Hierbei verwenden wir für die Texturdatei anstatt das standardmäßig ausgewählte EXR-Dateiformat ein PNG. Diese Einstellung beruht darauf, dass die Vuforia Model Target API die Textur ansonsten nicht verarbeiten kann.

11.5.3.4. Einschränkungen im Projekt

Grundsätzliche haben sich bei der Entwicklung der Applikation einige Hürden aufgetan, die wohl größte Hürde beim Teil der Objekt-Generierung war, die Voraussetzung einer CUDA fähige NVIDIA Grafikkarte für eine optimale Erstellung der Modelle. Da ein Server, der über eine solche Grafikkarte verfügt, das Budget sprengen würde, mussten einige der Schritte, bei der Überführung von 2D-Bildern in ein 3-dimensionales Objekt ausgelassen und kompensiert werden.

Aber klären wir zuerst was überhaupt diese „NVIDIA CUDA Beschleunigung“ ist und warum Meshroom diese für eine Einwandfreie Benutzung der Software voraussetzt. Grundsätzlich wurde die „Compute Unified Device Architecture“, kurz CUDA, 2006 von NVIDIA[38] entwickelt und macht es möglich Aufgaben auf einer Grafikkarte parallel auszuführen. Laut Incredibuild Software Ltd.[39] sei es so möglich Aufgaben, welche der Prozessor verarbeiten sollte, auf die GPU auszulagern, welcher diese selbst parallel zu seinen anderen Tasks ausführt, und das Ergebnis zurücksendet.

[38] [NVIDIA CORPORATION]
[39] vgl. [INCREDIBUILD SOFTWARE LTD.]

Um das besser zu verstehen, muss man wissen, dass die Grafikkarte dafür entwickelt, wurde, um zum Beispiel die Grafiken in Videospielen zu rendern. Wenn man sich nun vor Augen führt, dass auf einem Standard Monitor, mit einer Auflösung von 1920x1080 Pixel und 60 Bilder pro Sekunde, jeder einzelne Pixel, 60-mal in der Sekunde, berechnet werden muss. Dafür wird eine Hardware benötigt, welche viele Matrix-Berechnungen und Vektor Transformationen, die für die Berechnung des nächsten Bildes gebraucht werden, in kürzester Zeit, parallel, ausführen kann.

Das ist der Grund, warum eine Grafikkarte überhaupt benötigt wird. Diese unglaubliche Geschwindigkeit der Grafikkarte kann man sich durch die CUDA-Technologie nun vom Prozessor aus, zu Nutzen machen. Dies funktioniert so, dass die Daten, welche zur Berechnung benötigt werden, auf die GPU kopiert werden, dort die Aufgabe ausgeführt und das Ergebnis zurückgegeben wird. Diese Methode findet heutzutage in vielen Bereichen Anwendung, wie zum Beispiel Deep Learning, Künstliche Intelligenz oder Analysen von Big Data. Also in allen Bereichen, in denen große Datenmengen in kürzester Zeit verarbeitet werden müssen.

Meshroom selbst schreibt „To fully utilize Meshroom, a NVIDIA CUDA-enabled GPU is recommended.“[28] Ansonsten würde eine Berechnung rein mit der CPU schlicht viel zu lange dauern und ist deshalb kaum möglich. Grundsätzlich enthält die von der Rekonstruierungssoftware vorgegebene Pipeline für die Durchführung von Fotogrammetrie, zwei weitere Schritte, und zwar „DepthMap“ und „DepthMapFilter“.

Nach Schritt (6) PrepareDenseScene und vor Schritt (7) Meshing befinden sich diese beiden Knotenpunkte. Hierbei werden zuerst in „DepthMap“ die Tiefenwerte eines jeden Pixels aller Kameras durch eine Technik namens Multi-Stereo-View (kurz MSV), welche durch Schritt (5) StructureFromMotion ermittelt wurden, extrahiert. Im nächsten Schritt „DepthMapFilter“ werden diese Punkte noch optimiert und verdeckte Kanten identifiziert.

Auswirkung

In unserem Fall fehlen diese Nodes und im Schritt (7) Meshing werden die Daten aus dem Structure From Motion Knoten verarbeitet. Diese sind jedoch nicht optimal und werden durch den DepthMap Schritt ersetzt, da dieser die Tiefe jedes Pixels ermittelt und so zu viel genaueren Ergebnissen führt.

Wenn wir nun den Akku (siehe Abbildung 20 aus Seite 54) vollständig durch beide Pipelines durchlaufen lassen, kann folgender Vergleich in Abbildung 39 erstellt werden:

Abbildung 39: Vergleich Generierung mit und ohne CUDA-Beschleunigung

Aufgrund dieser Auswirkungen kann es teilweise zu Komplikationen beim Erkennen von Objekten kommen, da einige markante Punkte nicht optimal erkannt werden können.

11.5.4. Generator Schnittstelle

Dieses Kapitel handelt einerseits von der Schnittstelle an sich, welche sich aus einer Rest API zusammensetzt, andererseits von der Containerisierung dieser und wie die Umsetzung durch einen Docker Container erfolgte.

11.5.4.1. Generator Rest API

Da der Generator separat und abgekoppelt von der „Haupt“-Rest API läuft, benötigt dieser ebenfalls eine Schnittstelle, um mit ihm kommunizieren zu können, welches ebenfalls eine Rest API ist. Jedoch besitzt diese lediglich 3 Endpoints.

Übersicht

POST	`/api/Generator`	Senden der Eingabebilder für die Generierung
GET	`/api/Generator`	(Testzwecke) Zur Überprüfung, ob die API ansprechbar ist
GET	`/api/Generator/status/{uuid}`	Abfragen des Status eines Models

Autorisierung

Um nur Berechtigte Benutzer einen Zugang zu dieser Schnittstelle zu ermöglichen, wird bei einer Anfrage ein Verifizierungsschlüssel überprüft. Dieser muss im Head des Requests hinterlegt werden, und wird durch die Bezeichnung „GENERATOR-API-KEY“ identifiziert. Der Inhalt wird mit dem, in der Datenbank festgelegten Kennwort verglichen und falls diese übereinstimmen, wird die Anfrage zugelassen.

Für diesen Zweck existiert eine Middleware[40], welche vor jedem Aufruf aktiviert wird. Falls der gültige Schlüssel gegeben ist, wird die Anfrage weitergesendet zum Endpoint. Falls nicht wird dem Absender ein HTTP Status Code „401 Unauthorized Access“ zurückgesendet. Dies bedeutet, dass er keine Berechtigung hat auf diese Ressource zuzugreifen.

(1) POST /api/Generator

Hierbei handelt es sich, wie oben beschrieben um jene Schnittstelle, die es ermöglich Bilder zu senden, welche dann auf dem Server zu einem 3D Model verarbeitet werden. Grundsätzlich wird nach der Übertragung der Bilder, die Generierung gestartet und danach sofort eine eindeutige Identifikationsnummer zurückgegeben. Mit dieser kann man jederzeit den Status der Herstellung des Objektes abgefragt werden (siehe (3) GET /api/Generator/status/{uuid}).

Um Meshroom auf der Schnittstelle benutzen zu können, wird ein Docker Image von AliceVision verwendet. Mehr Informationen dazu findet man im Kapitel 11.5.4.2. Da die 3D-Rekonstruktionssoftware auch ein durch die Kommandozeile bedienbar ist, machen wir davon Gebrauch, um mittels Programm Code die Software auszuführen. Hierbei werden zuerst alle notwendigen Dateien und Ordner erstellt, welche dann als Argument übergeben werden:

```
meshroom_batch --pipeline {templatePath} --input {inputDir} --output {outputDir}
```

Das Argument `inputDir` stellt hierbei den Pfad jenes Verzeichnises dar, in welchem sich die Eingabebilder befinden. In unserem Fall sind es die, die beim Absenden der POST-Anfrage übergeben werden. Diese werden in einem Ordner mit der Bezeichnung „input“ abgelegt. Siehe Ordnerstruktur rechts

`outputDir` verweist auf einen leeren Ordner, in welchem nach Abschluss der Generierung, das Ergebnis gespeichert werden kann.

Das `templatePath` steht hierbei für den Pfad einer Meshroom (.MG)-Datei, welcher die Informationen über die Pipeline enthält. Hierbei verwenden wir bewusst nicht die standardmäßige „Photogrammetry Draft“ Pipeline, da noch einige Schritte angepasst wurden. Siehe 11.5.3.3 Anpassungen an der Pipeline.

```
|-- WorkingDirectory
|   |-- 12345
|   |   |-- input
|   |   `-- output
|   |       |-- texture_1001.png
|   |       |-- texturedMesh.mtl
|   |       `-- texturedMesh.obj
|   `-- e5af772e-a99a-4b7c-9d78-5d2e18b30dfa
|       |-- input
|       |   |-- Image_1.jpg
|       |   |-- Image_10.jpg
|       |   |-- Image_100.jpg
|       |   |-- Image_101.jpg
|       |   |-- Image_102.jpg
|       |   |-- Image_103.jpg
|       |   |-- Image_104.jpg
|       |   |-- Image_105.jpg
|       |   |-- Image_106.jpg
|       |   |-- Image_107.jpg
|       |   |-- Image_108.jpg
|       |   |-- Image_109.jpg
|       |   |-- Image_11.jpg
|       |   |-- Image_110.jpg
|       |   |-- Image_111.jpg
```

Abbildung 40: Darstellung der Ordnerstruktur innerhalb des Containers

In Abbildung 40 ist der Ordner namens „WorkingDirectory“ dargestellt. Eine Ebene darunter liegen alle Ordner, welche eine Modellgenerierung darstellen, gekennzeichnet durch eine UUID. Sie besitzen jeweils einen „input“ und einen „output“ Ordner. Um Speicher zu sparen wird der Eingabeordner nach Fertigstellung der Generierung gelöscht, weshalb der erste „input“-Ordner leer ist. Ebenfalls abzulesen ist, dass der untere Ordner, da er noch einige JPG-Dateien enthält, derzeit

[40] Software die als Vermittler zwischen Anwendungen oder Systemen fungiert

noch ein Objekt generiert. Außerdem kann erkannt werden, dass der Ausgabeordner die drei Dateien enthält welche in Kapitel 11.5.3.2 unter (9) Texturing erwähnt wurden.

Nachdem nun der obige Kommandozeilenbefehl in einem Thread gestartet wurde, kann die generierte UUID zurück an den Absender der Bilder gesendet werden. Außerdem wird der Ablauf, welcher nun parallel zu anderen Aktivitäten innerhalb der Rest API läuft, in einem „Dictionary“ gespeichert, um nachher bei der Statusabfrage identifizieren zu können, ob dieser Prozess bereits abgelaufen ist oder nicht. Ein Dictionary speichert zu einem bestimmten Typen, in unserem Fall „Thread“ einen Key. In dieser Applikation ist der Key die UUID, welche den Prozess eindeutig identifizieren kann.

(2) GET /api/Generator

Wie bereits erwähnt dient dieser Endpunkt nur zu Testzwecken. Es gibt lediglich eine Nachricht zurück, welche die derzeitige Version der API beinhaltet.

Beispielrückmeldung

```
Server is reachable: v2.0
```

(3) GET /api/Generator/status/{uuid}

Hierbei handelt es sich um einen Endpunkt, welcher Auskunft über den Status einer 3D-Modellierung gibt. Grundsätzlich gibt es verschiedene Status Codes, welche zurückgegeben werden können:

[404] Not Found

```
Object with ID 1dac9988-6912-187g-420-413376ca43e doesnt exist
```

Diese Statusmeldung wird zurückgegeben, wenn eine Identifikationsnummer eingegeben, welche nicht im System registriert ist. Außerdem ist es bei einem Statuscode 404 auch möglich, dass folgende Nachricht angefügt ist:

```
Object with ID 1dac9988-6912-187g-420-413376ca43e had an Error while compiling
```

Dies bedeutet, dass die Generierung des Modells nicht erfolgreich war, vermutlich da aus den Eingabebildern, kein vernünftiges 3D-Objekt generiert werden konnte.

[202] Accepted

```
Object with ID 1dac9988-6912-187g-420-413376ca43e is still processing
```

Falls das zu generierendes Objekt derzeit noch generiert wird, kommt oben genannte Meldung zurück. Das bedeutet, man muss noch etwas warten, bis das finale Objekt fertig ist.

```
[200] OK
FileStreamResult
```

Ist die Generierung nun fertiggestellt, wird der Status Code 200 OK durch ein „FileStreamResult“-Objekt zurückgesendet. Dieses beschreibt die Datei, welche zur weiteren Verarbeitung an die Vuforia Web API gesendet wird (siehe 11.2.5.5).

Grundsätzlich besteht eine 3D Modell nach Fertigstellung der Generierung aus 3 Dateien, wie in Abbildung 40 zu sehen. Um nun diese auf die Haupt-RestAPI senden zu können werden diese Dateien vorher in einen Zip-Ordner komprimiert und dieser dann als FileStreamResult, wie oben beschrieben, zurückgegeben.

11.5.4.2. Containerisierung

Um unser finales Produkt im Produktivbetrieb so flexible wie möglich einsetzen zu können, wurde diese Schnittstelle durch die Technologie Docker (siehe 9.1.6) in einen Container ausgelagert. Um nun unser Programm funktionsfähig innerhalb des Containers laufen zu lassen, muss ein Dockerfile erstellt werden, welches beschreibt welche Schritte ausgeführt werden müssen, um den Container zu erstellen. Bei der Erstellung dessen müssen folgende Dinge beachtet werden:

(1) Funktionsfähige Meshroom Installation

Die Voraussetzung dafür, ist, dass innerhalb der Umgebung eine lauffähige Version von Meshroom installiert ist. Diese kann dann, wie im Kapitel zuvor erwähnt, durch unsere Generator Applikation mittels der Kommandozeile bedient werden.

Unsere Grundlage ist ein von AliceVision bereit gestelltes Docker Image, welches einen Linux Container mit vorinstalliertem Meshroom, liefert. Unser Container verwendet ein Abbild dessen, welches am 11. Dezember 2023 veröffentlich wurde mit dem Tag „2023.3.0-av3.2.0-centos7-cuda11.3.1“[41] versehen ist. Dies bedeutet, dass die aktuelle Meshroom Version 2023.3.0[42] vollständig installiert ist, und bedient werden kann. Durch diesen Punkt können wir die erste Zeile unseres Dockerfiles festlegen.

```
FROM alicevision/meshroom:2023.3.0-av3.2.0-centos7-cuda11.3.1 AS meshroom
```

Dies legt fest, dass als Basis das eben beschriebene Image verwendet wird.

[41] [ALICEVISION DOCKERHUB]
[42] stand 24. März 2024

(2) Ausführbare und von außen Ansprechbare Generator Applikation

Um die Rest API verlässlich im Container laufen lassen zu können, muss dieses zuerst kompiliert werden. Da die Umgebung des in Schritt (1) festgelegten Containers, Linux ist, müssen vorerst einige Dateien installiert werden.

Zuerst wird mittels folgenden Eintrags das Linux Tool „rpm“ ausgeführt. Dieses lädt ein Microsoft Paket herunter und führt dieses auch aus, wodurch die nötigen .NET Core-Pakete installiert werden, mit welchem wir unsere Applikation in weiterer Folge kompilieren können.

```
RUN rpm -Uvh https://packages.microsoft.com/config/rhel/7/packages-microsoft-prod.rpm
```

Außerdem benötigen wir noch die .NET SDK. Ein solches Software Development Kit bezeichnet eine Sammlung von Werkzeugen, Ressourcen und Dokumentationen, die von Entwicklern verwendet werden können, um Anwendungen für eine bestimmte Plattform, Software oder Technologie zu erstellen. Hierbei handelt es sich um die .NET SDK, welches eine Vielzahl von Tools, unter anderem auch einen Compiler, umfasst. Die Installation erfolgt durch den Paketmanager „yum“ welcher unter Linux vorinstalliert ist. Der Paramter ‚-y‘ stellt sicher, dass die Installation ohne Benutzerinteraktion erfolgt. Da bei einer standardmäßigen Installation eine Benutzereingabe erforderlich ist.

```
RUN yum install dotnet-sdk-7.0 -y
```

Mithilfe des Compilers können wir unsere Applikation kompilieren. Dadurch erhalten wir eine ausführbare DLL-Datei mit welchem wir unsere API ganz einfach starten können. Dazu wird im Dockerfile mithilfe des Befehls „dotnet publish“ die Vorbereitung zum Deployment angestoßen und eine, wie zuvor beschrieben, .dll-Datei entsteht.

Außerdem wird am Ende des Dockerfiles ein sogenannter „Entrypoint“ festgelegt. Das bedeutet, dass bei jedem Start des Containers ein Befehl ausgeführt wird. In unserem Fall wollen wir unsere Applikation, bzw. die zuvor generierte ausführbare Datei, starten. Dies funktioniert folgendermaßen:

```
ENTRYPOINT ["dotnet", "GeneratorRestAPI.dll"]
```

Außerdem muss eine sichere Übertragung gewährleistet werden, deshalb setzten wir eine Umgebungsvariable, sodass unsere ASP.NET Core Applikation auf HTTPS läuft. Weiters muss dieser Port auch nach außen offen sein. Deshalb wird dieser mit dem „EXPOSE“ Schlüsselwort nach außen geöffnet.

```
ENV ASPNETCORE_URLS=https://*:443
EXPOSE 443
```

Nun ist unsere Schnittstelle nach Außen über den HTTPS Port 443 ansprechbar.

(3) Übertragen der Pipeline Datei (siehe 11.5.3.3)

Weiters muss unsere .mg-Datei in den Container kopiert werden, sodass die richtigen Schritte bei der Generierung ausgeführt werden. Dazu wird in das Verzeichnis „/src“ gewechselt, und diese dorthin kopiert. Dies funktioniert über einen COPY-Eintrag im Dockerfile.

(4) Aufsetzen der Umgebung

Um das Dateisystem unseres Containers sauber zu halten, werden im „home“ Verzeichnis zwei Ordner erstellt. Zuerst einen „installation“-Ordner in welchem alle Installationsdateien liegen. Weiters wird auch ein „WorkingDirectory“. Dieses wird in Kapitel 11.5.4.1 (1) genauer erläutert.

11.6. Object Recognition

11.6.1. Einleitung

Bei diesem Kapitel handelt es sich um die Erkennung der zuvor gescannten 3D-Objekte. Dies soll dem Installateur als Hilfestellung dienen, indem Arbeitsschritte oder Informationen neben einem erkannten Objekt angezeigt werden. Hierbei hat die Android-App Zugriff auf die Rest-API und kann sich somit, die zuvor trainierten 3D-Modelle herunterladen.

Für das Erkennen von Objekten wird auf die Vuforia Web API zugegriffen. Die Vuforia API ist zuständig, um 3D-Modelle zu trainieren und sie dann zu speichern. Das Unity Programm bekommt über eine, von uns programmierte, ASP.NET API alle Objekte/Modelle aufgelistet, die in der Vuforia API gespeichert sind. Davon kann man ein Modell auswählen. Bei der Auswahl eines Objektes werden die nötigen Dateien (.xml und .dat) heruntergeladen.

Mit diesen Dateien kann nun ein Model Target erstellt werden. Das Model Target ist für die Erkennung eines Objektes notwendig. Wenn ein Objekt erkannt wurde, dann wird ein Text/Objekt in der AR-Umgebung eingeblendet. Vor allem sind das Informationen über das Gerät oder Arbeitsschritte, die man zur Reparatur/Installation eines Gerätes benötigt.

11.6.2. Augmented Reality

11.6.2.1. Einleitung

Technologien wie Augmented Reality oder Virtual Reality beeinflussen immer mehr unseren alltäglichen Alltag. Besonders in der Industrie wird Augmented Reality als Unterstützung für einen Mitarbeiter/in verwendet. Bei unserer Diplomarbeit wird AR als Hilfestellung für Installateure der Firma ITPRO angewendet.

11.6.2.2. Augmented Reality vs. Virtual Reality

Die Idee beider Technologien ist, eine reale Umgebung durch eine simulierte Umgebung zu erweitern oder zu ersetzen. Dabei wird bei VR die reale Umgebung komplett durch eine simulierte Welt ersetzt und bei AR werden digitale Objekte in der realen Welt platziert. Für die Diplomarbeit ist AR am besten geeignet, denn bei Fragen über eine Maschine kann der/die Arbeiter/in mittels Verwendung der Kamera des Firmen-Handy digitale Informationen in der realen Welt angezeigt bekommen.

11.6.2.3. Funktionsweise AR

[43]Grundsätzlich kann AR auf verschiedensten Geräten wie Smartphones, Tablets oder Brillen angewendet werden. Die Hauptidee dabei ist, dass digitale Objekte oder Informationen über die physische Welt gelagert werden. Dabei benötigen die Hardware folgende Komponenten, um eine realistische Welt abbilden zu können.

[43] [REYDAR]

Eingabegeräte
Kameras und Sensoren, in unserem Fall die Sensoren der Handykamera, erfassen Daten der realen Umgebung und senden diese dann an das Programm weiter.

Prozessoren
Um ein Objekt in der realen Umgebung platzieren zu können, müssen vorher die berechneten Daten mittels eines Algorithmus verarbeitet werden. Zum Beispiel werden dabei der Standort und die Ausrichtung eines Objektes berechnet, damit digitale Elemente so genau wie möglich platziert werden können.

Ausgabegeräte
Die Ausgabegeräte können Bildschirme, Projektoren oder andere verschiedenste Displays sein, die nun die physische Welt und die digitalen Objekte kombiniert anzeigen. In unserem Fall ist es das Smartphone-Display, welches als Ausgabegerät verwendet wird.

11.6.3. Auswahl der Augmented Reality Art[44]

11.6.3.1. Einleitung

In diesem Kapiteln erwähne ich die verschiedenen Arten die Augmented Reality zur Verfügung stellt.

11.6.3.2. Allgemeinen Arten

Markerbasierte AR
Wie man vom Namen schon ablesen kann, werden mit dieser Technologie Markierungen verwendet, um digitale Elemente in einer bestimmten Umgebung zu platzieren. Die Markerbasierte Lösung erfordert weniger Leistung als andere Methoden, hat aber dafür wegen der einfachen Implementierung auch Einschränkungen. Zum Beispiel müssen Objekte im Vorhinein mit virtuellen Punkten markiert werden (händisch in einem AR-Editor-Programm), dass ermöglicht keine dynamische Erkennung von Objekten und ist meist sehr aufwendig.

Markerlose AR
Mit dieser Methode ist man auf die Sensoren und die Leistung der Geräte angewiesen. Der Vorteil gegenüber dem markerbasierten AR ist, dass man auf kein bestimmtes Bild, Objekt oder Muster angewiesen ist, daher ist es auch dynamischer einsetzbar.
Die Sensoren erkennen eine physische Umgebung, berechnen daraus eine digitale Fläche, wo sie AR-Objekte in der realen Welt mittels Anker platzieren können.
Diese Technologie ist nicht für jedes Gerät verfügbar, da es bestimmte Sensoren/Systeme benötigt.

11.6.3.3. Software

Unsere Diplomarbeit ist eine Erweiterung des Projektes. In unserem Projekt haben wir bereits Augmented Reality verwendet und dabei ARCore von Google benutzt. ARCore bietet keine Funktion, um Objekte mittels generierten 3D-Objekte automatisch zu erkennen, daher ist dieses Package für

[44] [REYDAR]

unsere Diplomarbeit ungeeignet.
Nach längeren Recherchen haben wir uns für die Vuforia Engine entschieden, die ebenfalls von Untiy unterstütz wird.

Vuforia bietet eine Reihe an Augmented Reality Funtionen an.

11.6.4. Vuforia AR-Arten[45]

11.6.4.1. Image-Targets

Mit dem Tool des Image Targets können Elemente in einer Augmented Reality Welt platziert werden, aufgrund eines Bildes. Hier werden nur 2D-Bilder als Zielobjekt verwendet. Ein Objekt wird erst gesetzt, wenn das Bild von der Vuforia Engine erkannt wurde. Images können folgende Formate besitzen: JPG oder PNG. Image Targets werden häufig als Marketing-Zweck eingesetzt, um Kunden mit neuen Features zu beeindrucken.

11.6.4.2. Multi Targets

Multi Targets ist die erweiterte Funktion von Image-Targets. Hier wird als Zielobjekt kein 2D-Objekt verwendet, sondern Quadrate und Rechtecke. Jede Fläche des Objektes besitzt nun ein bestimmtes Bild, somit kann die Erkennung des Objektes von jeder Seite durchgeführt werden. Das AR-Element erscheint, wenn das 3D-Modell von der Vuforia Engine erkannt wird.

11.6.4.3. Cylinder Targets

Ähnlich zu den Methoden davor, kann auch hier ein AR-Objekt aufgrund eines Bildes platziert werden. Bei Cylinder Targets können nun zylindrische Formen erkannt werden.

11.6.4.4. Barcode Scanner

Wie es der Name schon vermuten lässt, stellt Vuforia hier ein Tool zu Verfügung, um Barcode- und QR-Code mittels Handys lesen zu können. Dabei können Produkte erkannt werden, in dem sie eine Seriennumer eingelesen haben.

11.6.4.5. Area Targets

Mittels Area Targets können größere Räume, wie der Arbeitsplatz im Büro oder Produktionshallen, in eine Augmented Reality Welt umgewandelt werden. Davor muss ein 3D-Scan vom gesamten Bereich erstellt werden und danach können AR-Punkte erstellt werden, wo nun 3D-Elemente angezeigt werden. So können Räume/Firmenabteilungen mittels AR-Geräten wie ein AR-Brille in einer ganz anderen Weise vorgestellt werden.

11.6.4.6. Ground Plane

[45] [VUFORIA, PTC]

Ground Plane ist ein AR-Funktion, die es ermöglicht horizontale Flächen/Objekte zu erkennen und AR-Objekte zu platzieren. Die AR-Elemente werden mit „Anchor Points“ platziert und können somit auf der erkannten Fläche beliebig verschoben werden. Es ist sehr gut geeignet für Produktvisualisierungen, wie zum Beispiel das Einrichten eines Büros mithilfe von Augmented Reality.

11.6.4.7. Model Targets

Mit dieser Technologie können erstellte 3D-Modelle in der realen Welt erkannt werden und dann wichtige Details oder Anweisungen zu einem Objekt anzeigt werden. Model Targets können entweder statisch oder dynamisch im Unity Projekt einfügt werden.

11.6.5. Entscheidung der AR-Art

Für unsere App benötigen wir eine Augmented Reality App, die 3D-Modellen erkennen kann. Somit fallen die Image-basierten Arten raus. Außerdem soll ein Objekt horizontal sowie vertikal erkannt werden. Deswegen ist Ground Plane für uns nicht infrage gekommen, da die Erkennung eines Models nicht vertikal funktioniert.
Zum Schluss haben wir uns für die Model Targets Variante entschieden, weil sie alle erforderten Komponenten (generieren eines 3D-Modells, Erkennen eines 3D-Modells 360 Grad) mitbringt, die wir für unsere AR-App benötigen.

11.6.6. Statische AR-Anwendung

11.6.6.1. Einleitung

In diesem Kapitel geht es um eine Beispiel Anwendung, die prüft soll, ob das Erkennen eines Objektes funktioniert. Außerdem gibt es einen Einblick, wie Vuforia mit Model Targets umgeht und wie diese in ein Unity Projekt eingebunden werden können.

11.6.6.2. Voraussetzungen

In diesem Kapitel behandeln wir die Vorschriften, wie ein 3D-Modell am besten erkannt wird und welche Plattformen unterstützt werden.

Allgemein kann ein reales Objekt nur mittels einem 3D-Modell erkannt werden, dass zum Beispiel ein 3D-CAD-Modell aber auch ein 3D-Scan des Objektes sein kann. Dabei ist zu beachten, dass glänzende Oberflächen nicht erkannt werden und es sollen ausreichend viele Eckpunkte des Objektes vorhanden sein.

Mit dem Model Target Tool können grundsätzlich kleine und große physische Geräte erkannt werden. Schwieriger wird es, wenn sich das Objekt während der Erkennung Phase bewegt, denn dabei können Punkte des Modells verloren gehen. Deswegen sollte man das Objekt nicht bewegen oder in der Einstellung den „Default Tracking Optimization“ aktivieren.

Wie vorher schon angesprochen sollen Objekte ausreichende geometrische Eckpunkte besitzen, damit sie sich von anderen Objekten unterscheiden können. Außerdem sollen symmetrische Objekte vermieden werden.

Eine weitere Regel gibt vor, dass die erstellten 3D-Modelle möglichst Realitätsnah sind, damit physische Modelle schneller und effektiver erkannt werden. Besonders die Größe des CAD-Modells soll gleich dem echten Objekt sein.

Um eine noch effektivere Erkennung zu schaffen, sollte 3D-Modelle nicht einfach einfarbig sein, sondern mit farbigen Mustern bemerkt sein, um 3D-Modelle besser zu unterscheiden.

Grundsätzlich wird Model Targets von den meisten Geräten unterstützt. Die Vuforia Engine können sowohl iOS-Geräte als auch Android Geräte verwenden. [ENGINE, PTC Vuforia]

11.6.6.3. 3D-Model Generierung

In unserem Projekt übernimmt Meshroom, die Generierung der 3D-Modelle. An diesem Abschnitt wird gerade gearbeitet, deswegen verwenden wir für das erste Projekt den Model Target Generator von Vuforia. Hierbei ist anzumerken, dass der MTG eine „Database“ in Vuforia erstellt, worauf die Unity-App zugreifen kann. Später soll das nicht mehr über die Vuforia-Database laufen, sondern über sogenannte .xml und .dat Dateien, die wir von der API zurückbekommen.

Mit dem Model Target Generator können spezielle Eigenschaften wie „Guide Views“ erstellt werden, diese haben wir mit Meshroom nicht.

11.6.6.4. Model Targets vs. Advanced Model Targets

Für unsere AR-Anwendung verwenden wir Advanced Model Targets, mit diesen Modellen kann das Modell an allen Seiten getrackt werden und der Benutzer muss die Ansicht nicht manuell am physischen Objekt ausrichten.

11.6.6.5. Guide View

Eine Guide View ist eine Unterstützung zum Erkennen des Objektes. Wenn eine Guide View aktiv ist, dann erstellt die App eine ähnlich gezeichnete Struktur vom Model. Während des Erkennens muss das Geräte so bewegt werden, dass die Struktur über dem echten physischen Objekt liegt. Ist dies erfolgt, so wird das Objekt erkannt.

11.6.6.6. Erstellen eines Model Targets (MTG)[46]

Zuerst muss man sich den Model Target Generator von der offiziellen Seite von Vuforia herunterladen. Beim Start des Target Generator muss man sich mit seinem Developer-Account anmelden. Vuforia bietet eine kostenlose Registrierung an.

[46] [VUFORIA, PTC]

Wenn man die Software startet, kommt man zu einer Übersichtsseite, wo man nun ein 3D-Model

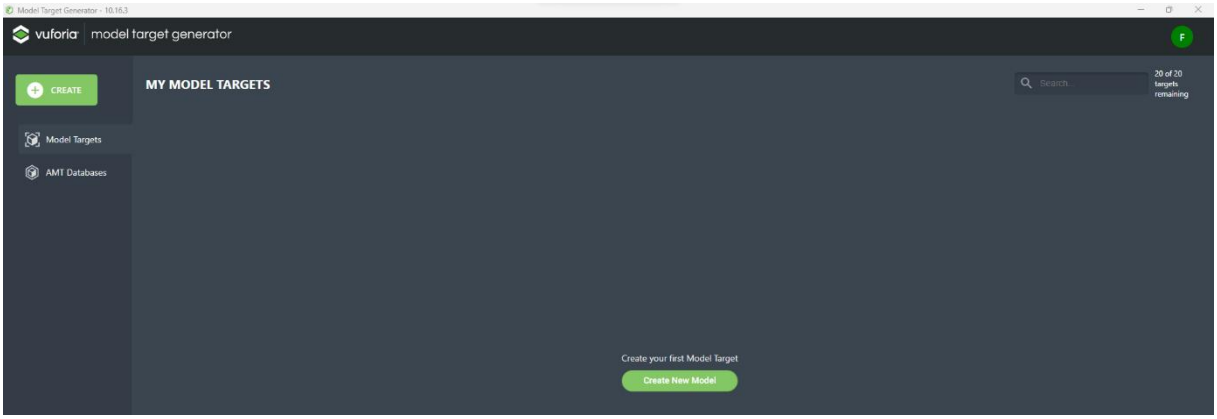


Abbildung 41: Hauptseite Model Target Generator

anlegen kann. Mit einem Klick auf „Create new Model“ kann ein neues Model Target erstellt werden.

Zuerst benötigt man ein 3D-Model, bevor man ein Model Target erstellen kann.
Als 3D-Model verwenden wir ein Handy.
Dazu haben wir uns ein einfaches 3D-Model gratis vom Internet gedownloadet.

Als nächstes wählt man das 3D-Model aus, wir vergeben dem Model den Namen „mobile_phone“ und zum Schluss, wo das generierte Model Target gespeichert werden soll.

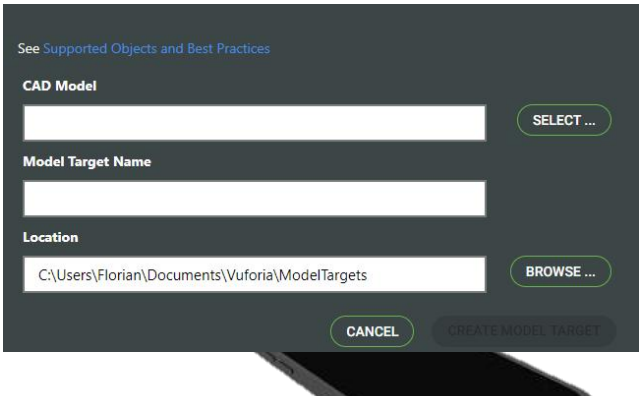


Abbildung 42: Eingabe eines CAD-Models im MTG

Danach können verschiedenste Einstellungen unternommen werden.

Model Up Vector

Einstellung in welcher Richtung das 3D-Modell trainiert werden soll.
Querformat oder Hochformat usw.

Model Units

Angabe der Maßeinheit
hier sieht man bereits, wie hoch und breit das Modell ist

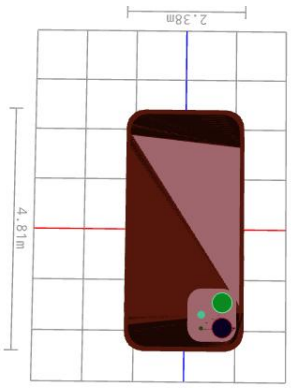


Abbildung 44: Anzeige des 3D-Modells im MTG

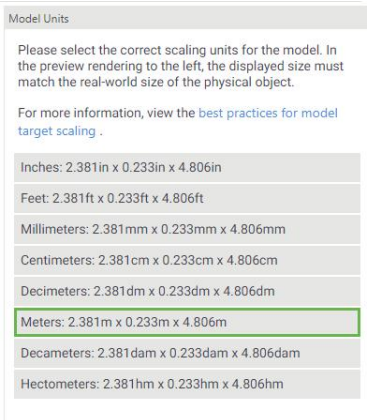


Abbildung 43: Auswahl der Model Units im MTG

Colorin

Es gibt zwei Auswahlmöglichkeiten:

Realistic Appearance
Model Targets werden nur erkannt, wenn auch die Farben vom Model Target zum realen Objekt übereinstimmen.

NON-Realistic Appearance
Farben und Texturen eines Model Targets wirken sich nicht auf die Erkennung eines realen Objektes aus

Optimize Tracking

Beim Optimize Tracking kann zwischen „Default“ und „Low Feature Objekt“ ausgewählt werden.

Default
Beste Tracking Performance für alle Model Targets anwenden

Low Feature Object:
Es können Objekte mit wenigen Eckpunkten und geringen Texturen leicht erkannt werden. Nicht einsetzbar bei Objekten, die bewegt werden.

Guide Views

Create Guide View
Guide View muss über dem Objekt liegen, um erkannt zu werden

Create Advanced View
Das Objekt kann von jedem Winkel erkannt werden, auch wenn es nicht mit der Guide View exakt übereinstimmt. Dafür wird ein Training des Models in der Vuforia Cloud benötigt.

Ausrichtung Guide View
Es kann eine Guide View als Landscape oder Portrait erstellt werden oder für „Digital Eyewear“, wie z.B. HoloLens, Magic Leap

Wenn man eine Guide View ausgewählt hat, dann kann man sich das Model Target generieren lassen.

Für unser Model Target haben wir folgende Einstellungen getroffen:

```
Up Vector: Y
Model Units: Meter
Coloring: NON-Realistic
Optimize Tracking: Low Feature Object
Guide View: Advanced Guide View
```

Wir haben uns für ein Low Feature Object entschlossen, weil das Handy-Modell nicht genügend Eckpunkte und Texturen hat und damit wir verschiedenste Handyarten verwenden können.

Das Model Target benötigt ungefähr vier Stunden Zeit, bis es fertig ist, dies hängt von der Rechenleistung des Gerätes (PC, Laptop) ab.

Unter dem Bereich „AMT Databases“ werden alle generierten Datenmodelle aufgelistet.

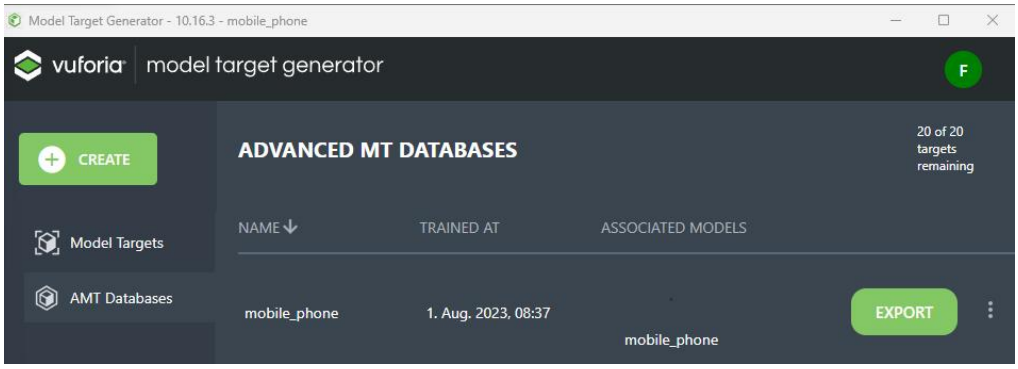


Abbildung 45: Auflistung aller trainierten Modelle im MTG

Um das Model Target im Unity Projekt verwenden zu können, muss man es vorher noch exportieren. Beim Exportieren vom „Mobile_Phone“ Model Target bekommt man folgendes zurück.

- *.unitypackage*
- *.dat*
- *.xml*
- *GuideViews als .png*

Für die statische Erkennung benötigen wir die „mobile_phone_unitypackage“ Datei.

11.6.6.7. Erstellung eines Model Targets (Target Manager)

Neben dem MTG gibt es die Möglichkeit eine „Database“ online zu erstellen. Dazu meldet man sich auf der Vuforia Webseite an und wechselt zur Target Manager Seite.

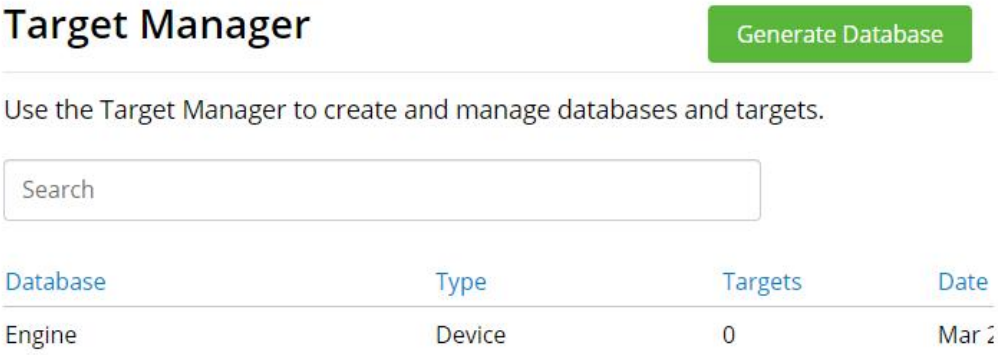


Abbildung 46: Auflistung aller Datenbanken im TM

Beim Target Manager können mehrere „Datenbanken“ angelegt werden, diese können wiederum mehrere Modelle beinhalten.

Im Target Manager können unterschiedliche Targets hochgeladen werden. Image, Multi Images und auch cylinderförmige Modelle können erstellt werden. Die Object Methode wird ab der Vuforia Engine 10.5. nicht mehr weiter unterstützt. Für unsere App ist der Target Manager nicht sinnvoll einsetzbar, da keine Model Targets erstellt werden können. Für Anwendungen mit Bildern kann der Target Manager eingesetzt werden.

Abbildung 47: Auswahlbaren Möglichkeiten im TM

11.6.6.8. Erstellen einer statischen AR-Unity App

Zuerst haben wir uns eine leeres 2D-Unity Projekt angelegt und die Vuforia Engine mittels einem Vuforia Assets in das Unity-Projekt geladen.
Als nächstes löscht man die Main Camera und fügt eine AR-Camera ein. Die Kamera wird benötigt, um Elemente in der AR-Umgebung platzieren zu können.

Damit man ein Vuforia Objekt einfügen kann, benötige ich eine Lizenz von Vuforia. Dafür meldet man sich auf der Vuforia Webseite an und erstellt einen Basic-License.

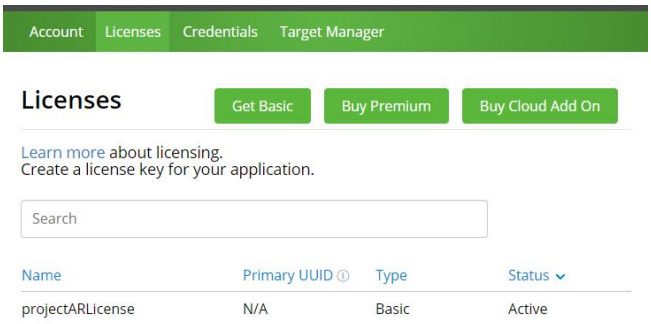


Abbildung 48: Auflistung aller Lizenzen bei Vuforia

Der Aufbau einer statischen App sieht folgendermaßen aus.

ARCamera:
Kamera, die eine AR-Umgebung realisiert

ModelTarget:
GameObject, dass zum Laden eines ModelTargets verantwortlich ist

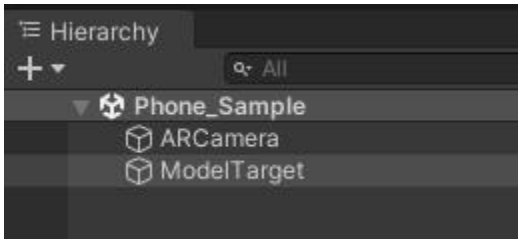


Abbildung 49: Aufbau AR Example

Das zuvor generierte Model Target wird mit dem Unitypackage importiert. Dazu geht man in Unity auf „Assets->Import Package -> Custom Package“ und wählt das Mobile_Phone.unitypackage aus.

Im Model Target GameObject wird ein ModelTargetBehaviour erstellt. Dort kann die Database und das bestimmte Model Target ausgewählt werden. Außerdem ist es besonders wichtig, dass das

Model mit der Größe des physikalischen Objektes übereinstimmt. Deswegen kann die Länge, Breite und die Höhe einstellen. Zum Schluss kann noch die GuideView aktiviert werden oder nicht.

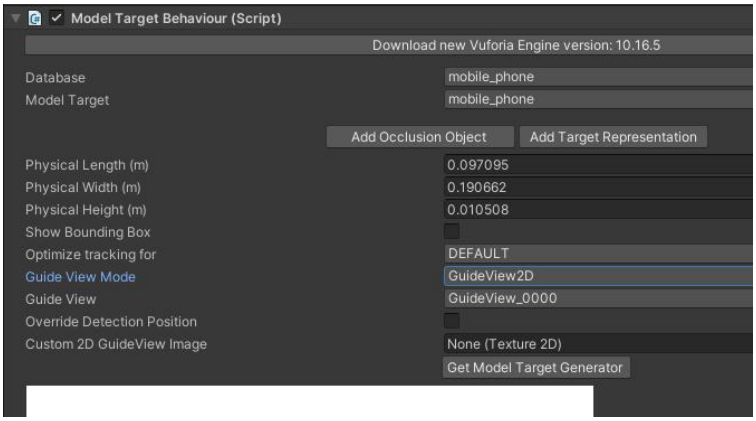


Abbildung 50: Aufbau ModelTargetBehaviour Komponente

Damit wir nachverfolgen können, ob das Handy erkannt wurde, haben wir einen Textblock erstellt. Dieser soll angezeigt werden, wenn das Objekt erkannt wird.
Nach der Ausführung des Programmes in Unity, sieht es folgendermaßen aus.
Das statische Erkennen eines Objektes hat somit funktioniert

11.6.6.9. Problem – statische Erkennung

Bei statischen Beispielen kann immer nur ein Model Target zur Laufzeit erkannt werden und es müsste jedes Mal ein neues Model Target über die Assets importiert werden. Will man ein neues Model Target verwenden, so muss das Alte gelöscht werden und das neue Model eingefügt werden.
Genau deswegen gibt es die dynamische Erkennung, die mit .xml und .dat Dateien arbeiten und nicht mit Unity Paketen.

Abbildung 51: statische Erkennung des Handys

11.6.7. Aufbau dynamische Erkennung

Der Unterschied zur statischen Erkennung ist, dass nun mehrere Model Targets zur gleichen Laufzeit verwendet werden. Wir benötigen für das Kapitel Object Recognition folgende Szenen und Klassen.

- **DatabaseController.cs**
  Schnittstelle zur SQLite Datenbank, um auf das Passwort und den Username zugreifen zu können.
- **ModelView**
  Darstellung aller Objekte, die zur Verfügung stehen. Objekte, die derzeit trainiert werden, werden auch angezeigt
- **ModelTargetScene**
  Diese Szene ist für die Erkennung eines Objektes zuständig

11.6.8. Zugriff SQLite-Datenbank

Der Installateur besitzt eine interne SQLite-Datenbank auf seinem Handy. Dort werden wichtige Information, wie zum Beispiel sein UserName, seine UserID und sein Passwort gespeichert. So kann er auf seine Apps automatisch zugreifen, ohne sich vorher anmelden zu müssen. Zugriff auf unsere API hat man nur mit einem Bearer-Key, den man bekommt, wenn man sich anmeldet. Für diese Anmeldung muss vorher auf die SQLite Datenbank zugegriffen werden, um das Passwort zu erhalten.

11.6.8.1. Installer-Klasse

In der Klasse „Installer“ werden Information über das Gerät und über den Mitarbeiter gespeichert. Die Tabelle und die Bespieldaten wurden von unserem Auftraggeber ITPro zur Verfügung gestellt. Die Abbildung ist ein Ausschnitt von der zur Verfügung gestellten Tabelle.

Attribute

```
[PrimaryKey]
[Column("Id")]
public string Id { get; set; }

[Column("UserName")]
public string UserName { get; set; }

[Column("Password")]
public string Password { get; set; }

[Column("MessgerätSeriennummer")]
public string? MessgerätSeriennummer { get; set; }

public short? IsExtern { get; set; }

[Column("PasswordEncrypted")]
public short? PasswordEncrypted { get; set; }
```

11.6.8.2. DatabaseController-Klasse

Die Klasse “DatabaseController” erlaubt es uns auf die interne Datenbank zugreifen. Für unsere Testphase wird die Datenbank „MasterData.db“ automatisch erstellt, wenn sie noch nicht vorhanden ist. Diese Klasse verwendet das SQLiteConnection Package von Unity, um die SQLite Datenbank aufrufen zu können. Der DatabaseController verwendet zusätzlich die UserDatabaseController.cs, weil dort alle Methoden für den User und den Installateur implementiert sind. Um nun den UserName und das Passwort vom Installateur zu bekommen, gibt es die Methode „getInstallerFromDatabase()“. Als Rückgabewert bekommt man ein Installer-Objekt zurück, wo nun Username und Passwort gespeichert sind.

```
public Installer getInstallerFromDatabase() {
    Debug.Log(173);
    return _userDatabaseController.mechanicUser;
}
```

11.6.9. Auflistung der Model Targets in Unity

11.6.9.1. Einleitung

Bevor wir die dynamische Erkennung eines Model Targets implementieren, erstellen wir eine Seite, wo der Installateur alle Model Targets aufgelistet bekommt. Zusätzlich wird der Status eines Models angezeigt, denn ein Model kann gerade noch von Vuforia trainiert werden oder es könnte auch ein Fehler bei der Generierung eines Model Targets aufgetreten sein.
Der Status kann folgende Eigenschaften aufweisen:

- Verfügbar
- Wird noch trainiert
- 3D-Modell wird erstellt

11.6.9.2. Aufbau ModelView.Scene

Grundsätzlich muss man in Unity für jede Seite eine eigene Scene anlegen. Eine Scene besteht dann aus mehreren GameObjects, die auf der Seite angezeigt werden.
Ein GameObjects kann verschiedenste Komponenten besitzen und diese dann mit C# Dateien verbinden.
Die ModelView.Scene ist folgendermaßen aufgebaut.

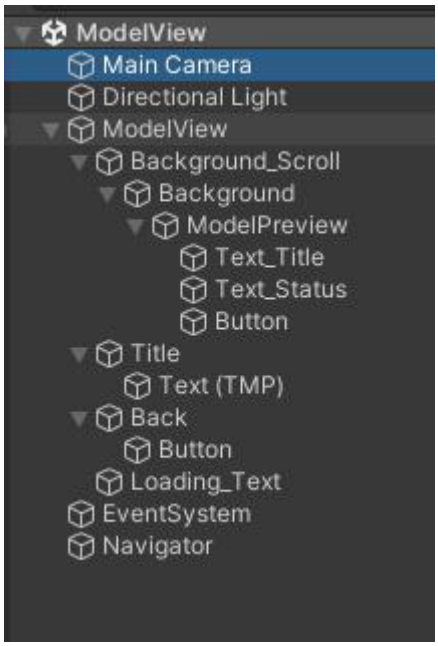


Abbildung 52: Aufbau ModelView in Unity

Main Camera:

Beinhaltet eine Kamera, die den Canvas/Seite der ModelView anzeigt.

Modelview:

Beinhaltet die Canvas und Canvas Scaler Komponente. Diese beiden Komponenten erstellen eine Oberfläche, wo Formen wie Quadrate, Rechtecke aber auch Texte können in diesem GameObject platziert werden.
Der Canvas hat verschieden Modi, wie die Seite am Gerät angezeigt werden kann.

Overlay:
Die Größe des Canvas passt sich hierbei automatisch an die Größe oder die Auflösung des Bildschirms an.

Kamera:
Diese Einstellung ist ähnlich zum Overlay, hier passt sich der Canvas and den Kameraeinstellungen an. Alle UI-Komponenten werden von der Main-Kamera gerendert. Die Größe des Canvas wird automatisch geändert, wenn sich die Größe des Bildschirms oder die Auflösung geändert wird.

Weltraum:
Bei diesem Modus legt man eine fixe Größe mittels eines Rect-Transforms an. Alle Objekte kann man nun in dieser Fläche einfügen und an der Größe anpassen. Aufgrund der Platzierung werden die Objekte entweder vor oder hintereinander gerendert.

Background_Scroll:
Damit eine Scrollable List in Unity möglich ist, benötigt man die Komponente „Scroll Rect“. Dazu erstellen wir ein GameObject und fügen die Komponente hinzu. Dann zieht man das Rechteck über die ganze Seite. Alle Child-Objekte, die unter dem Background_Scroll Objekt liegen, können dann hinauf oder hinunter geschoben werden.

Background:
Das Background Objekt stellt zum einen die Hintergrundfläche der Seite dar und zum anderen wird das GameObject genutzt, um die einzelnen Model Targets in einer schönen Liste darzustellen.
Folgende Komponenten wurden hinzugefügt:

Image:
mit dem Image kann man dem GameObject einen schwarzen Hintergrund vergeben

Vertical Layout Group:
mit dieser Komponente können untergeordnete Objekte in einer geordneten Weise angeordnet werden. Die Abstände für die Objekte werden für alle einzeln angegeben. Einstellungen wie das Padding (Links, Rechts Oben, Unten) oder der Abstand zwischen einzelnen Objekten kann angegeben werden. Auch die Ausrichtung kann eingestellt werden.

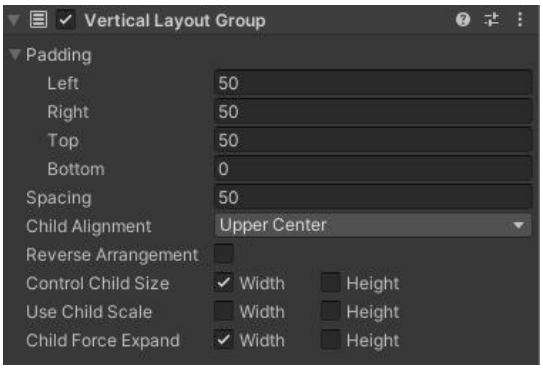


Abbildung 53: Komponente Vertical Layout Group in Unity

Der Background besitzt auch das selbst geschriebene C#-Datei „ModelTargetView“.

ModelPreview

Das ModelPreview GameObject repräsentiert eine Vorlage für alle Model Targets Elemente. Als einzige Komponente wird die Image Komponente verwendet.

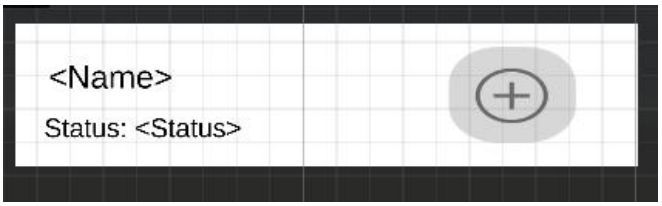


Abbildung 54: ModelPreview als Vorlage

Das Objekt besitzt drei weitere GameObjekte, dass zwei Textfelder und ein Button.

Button
Der Button verfügt über die Funktion, auf den nächsten Navigator weiterzuleiten und die entsprechenden Parameter zu übergeben, damit man die richtige Seite aufruft. Mit dem Button wird ein Model Target, dem Navigator übergeben. Der Button ist nur aktiv/aufrufbar, wenn beim Status DONE steht und wenn ein Dataset zum jeweiligen Model Target mitübergeben wurde. Ansonsten ist es nicht möglich ein Model erkennen zu lassen.

Textfeld-Titel
Dieses Textfeld ist dafür da, um den Titel oder Namen des Model Targets anzuzeigen.

Textfeld-Status
Hier wird der aktuelle Status von einem Model-Target angezeigt. Denn es werden auch Modelle angezeigt, die gerade von Vuforia trainiert werden. Zusätzlich können Fehler während der Generierung auftreten, die hier angegeben werden.

Title & Back

Diese zwei Objekte sind weiße Rechtecke, die den Titel der Seite und einen Button zur Verfügung stellen. Mit dem Button wird man zum Navigator weitergeleitet, mit den Parametern, dass man eine Seite zurück möchte

Loading_Text
Da alle Model Targets asynchron von der API abgerufen werden, dauert es ein paar Sekunden, bis alle Model Targets geladen wurden. Beim Loading_Text steht „wird geladen…“, der danach verschwindet.

11.6.9.3. Modell-Klassen

Die Klasse eines Model Targets besteht aus zwei Klassen. Die „Targets“-Klasse, dort werden allgemeine Informationen über alle Model Targets gespeichert, und in der „ModelTarget“-Klasse werden wichtige Daten über ein einzelnes Model Target gespeichert.

Targets-Klasse

```
public class Targets
{
    public string? total;
    public string? returned;
    public List<ModelTarget>? targets;

}
```

targets: Die Klasse besitzt eine Liste von allen Model Targets der Klasse „ModelTarget“> und zwei weitere Attribute.
Total: Anzahl der Model-Targets
returned:

Die Attribute können auch Null zurückbekommen. Das „?“ nach dem Typen kennzeichnet, dass es ein Nullable-Wert ist.

ModelTarget-Klasse

```
[Serializable]
public class ModelTarget
{
    public string? uuid;
    public string? name;
    public string? description;
    public string? modelGenerationStatus;
    public string? datasetZip;
    public string? datasetStatus;
}
```

In dieser Klasse werden alle relevanten Informationen gespeichert, die zur Erkennung der Objekte benötigt werden

UUID
Es ist der Schlüssel, mit dem das Model Target eindeutig identifiziert werden kann. Wenn man ein Model Target von der Vuforia Web API einzeln erhalten möchte, dann über die UUID.

Name
Der Name ist der Titel des Modells, soll ebenfalls eindeutig gespeichert werden, damit der Installateur das richtige Model finden kann.

Description
Beinhaltet eine Anleitung/Beschreibung/Information über das jeweilige Objekt. Wird neben dem Objekt angezeigt, wenn es erkannt wird.

modelGenerationStatus
Attribut für die Speicherung des aktuellen Status vom Generieren eines 3D-Modells.

datasetZip
Speicherung eines base64-Strings, der die zwei Dateien .dat und .xml speichert, die für die Erkennung der Modelle notwendig sind. Der base64-String muss dann zu einem byte-Array umgewandelt werden, damit man eine ZIP-Datei generieren kann.

datasetStatus
Attribut für die Speicherung des aktuellen Status vom Trainieren des 3D-Modells in Vuforia

AccessTokenResponse - Klasse

```
public class AccessTokenResponse
{
    public string accessToken;
    public string refreshToken;
}
```

Diese Klasse wird benötigt, um sich einen Bearer-Key abzuholen.
Der Bearer-Key wird im AccessToken gespeichert.

11.6.9.4. ModelTargetView-Datei

Diese Datei ist hauptsächlich für die Auflistung aller Model Targets verantwortlich.

Übergabeparameter der Datei

Text
Der Loading_Text wird dabei übergeben, damit der Text deaktiviert wird, wenn alle Model Targets vollständig geladen sind

Preview
Übergabe des ModelPreview. Es dient als Vorlage aller Model Targets, damit ein einheitliches Design entsteht.

Rect Transform

Die Komponente Rect Transform wird verwenden, um alle Target Objekte an den Background zu binden und damit sie in die Vertical Layout Group Komponente eingebunden werden.

Variablen

ControllerTarget: Klasse, die auf die API zugreift und alle Model Targets herunterladen kann. Außerdem dient sie zu dem Authentifizieren mit der API

Navigator: Klasse, die als Schnittstelle zwischen verschiedenen Scenes arbeitet. Weiterleitung zur nächsten/letzten Seite

Methoden:

void Start-Methode

Diese Methode wird automatisch aufgerufen, wenn die Seite ModelTargetView geladen wird.
In dieser Seite verweisen wir auf die Methode loadModelTargets(), um die Model Targets zu laden.

public async void loadModelTargets();

In dieser Methode wird sich über die Authentifizierungsschnittstelle in der Klasse ControllerTarget der Bearer-Key abgeholt. Der Bearer-Key wird für alle Transaktionen in der API benötigt, um zum Beispiel einen GET-Request absetzten zu können.
Danach wird die Methode ControllerTarget.getModelTargets(<Bearer-Key) aufgerufen, um alle Targets zu bekommen. Wenn das Target-Objekt nicht leer ist, dann wird die Methode buildView(<Targets>) aufgerufen, um nun die Scrollable List zu füllen.
Falls das Target-Objekt leer ist, wird der Loading_Text mit einer Fehlermeldung gesetzt.
Bevor Model Targets heruntergeladen werden, muss man sich anmelden, damit nur ein Installateur auf die den Endpoint zugreifen kann. Dazu gibt es den DatabaseController. Denn es wird sich der Benutzername und das Passwort von der internen SQLite Datenbank abgefragt und dann an die Rest-API mitgeschickt.

```
string bearer = await controller.authentication();
 Targets targets = await controller.getModelTargets(bearer);

//Targets targets = new Targets();
 if (targets != null)
 {
     text.enabled = false;
     buildView(targets);
     //buildView(null);
 }
 else
 {
     Debug.Log("not");
     text.SetText("ModelTargets konnte nicht geladen werden");
 }
```

public void buildView(Targets targets);

Diese Methode bewirkt, dass die Liste mit den Model Targets erstellt wird. Dafür wird ein Objekt der Klasse Targets übergeben.
Als ersten Schritt wird der Background auf seine Höhe angepasst, da der Hintergrund der Scrollable List größer sein muss, damit alle Model Targets auf die Scrollable List passen.
Als nächstes wird mit einer foreach-Schleife über alle Model-Targets iteriert. Dabei wird das Prefab dupliziert und Name und Beschreibung in die entsprechenden Textfelder eingefügt.
Die duplizierte Prefabs werden horizontal angeordnet, wie wir es vorher in der Komponente Vertical Layout Group angegeben haben.
Zum Schluss wird das Prefab gelöscht, damit es nicht mehr angezeigt wird.

```
foreach (var target in targets.targets)

{
   GameObject inst = Instantiate(preview, transform);
   TextMeshProUGUI[] texts = inst.GetComponentsInChildren<TextMeshProUGUI>();

   texts[0].text = target.name;
   texts[1].text = "Status: " + target.datasetStatus;
   Debug.Log("BuildView: "+target.datasetZip);

   if (target.datasetStatus.Equals("DONE"))
   {
       inst.GetComponentInChildren<Button>().onClick.AddListener(delegate {
            controller.loadZipFile(target.datasetZip);
           navi.LoadModel(target);    });
   }
   else
   {
       inst.GetComponentInChildren<Button>().interactable = false;
   }
   Debug.Log(target.name);
}
Destroy(preview);
```

public void setHigh(int length);

Methode, um die Höhe des Backgrounds zu setzen. Wenn es mehr als 6 Model Targets gibt, dann wird die Background Höhe jeweils um je 100 zu erhöhen.

11.6.9.5. Navigator-Klasse

Die Navigator-Klasse ist die Schnittstelle zwischen den verschiedenen Scenen. Möchte man eine andere Seite aufrufen, dann ruft man eine der implementierten Methoden der Navigator-Klasse auf.

Attribute

public static ModelTarget target;

Wenn man ein Model Target ausgewählt hat und den nun mittels Vuforia erkennen möchte.
Dann wird das ausgewählte Model Target hier gespeichert und nun hat der Observer auf das Target Zugriff.

Methoden

public void goToTargetModel() { SceneManger.LoadScene("ModelTargetScene") }

Eine Methode damit man zur ModelTargetScene wechseln kann.

public void goToLoadModels () { SceneManger.LoadScene("ModelView") }

Methode, um zur ModelTargetScene zu wechseln

public void LoadModel () { target = model; goToTargetModel() }

Methode, um das ausgewählte Target zu setzen und die „goToTargetModel" auswählen.

11.6.9.6. ControllerTarget-Klasse

Diese Klasse dient dazu, um auf die Api zugreifen zu können. Dabei werden Model-Targets heruntergeladen aber auch Authentifizierungszugriffe getätigt. Zusätzlich hat man Methoden zur Verfügung, um das Dataset von einem base64-String zu den Dateien .dat & .xml zu konvertieren.

public async Task<string> Authentication(string username, string password)

Bei dieser asynchronen Methode meldet sich der Installateur mit Username und Password an. Dabei bekommt man einen Bearer-Schlüssel zurück, den man für den Zugriff auf die Rest-API benötigt. Das Passwort und den Benutzernamen bekommt man über die interne SQLite Datenbank. Dort werden alle Informationen über den Installateur gespeichert.

Der Username und das Passwort werden über den RequestHeader mitgeschickt.
Damit wir auf die API zugreifen können, verwenden wir den HttpClient. Die richtigen Adressen, des jeweiligen Endpoints werden in der ConfigUrl-Klasse gespeichert.

Rückgabeparameter:
string: Rückgabe des Bearer-Key, um auf die API zugreifen zu können.

```
using (HttpClient client = new HttpClient())
{
    client.DefaultRequestHeaders.Accept.Clear();
    client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));

    StringContent content = new StringContent(jsonPayload, Encoding.UTF8,
"application/json");

    HttpResponseMessage response = await client.PostAsync(ConfigUrl.azureLogin,
content);


    if (response.IsSuccessStatusCode)
    {
        string jsonResponse = await response.Content.ReadAsStringAsync();

        AccessTokenResponse tokenResponse =
JsonUtility.FromJson<AccessTokenResponse>(jsonResponse);
```

public async Task<Targets> getModelTargets(string bearer)

In dieser Methode werden mittels HttpClient alle 3D-Objekte heruntergeladen. Wenn ein Fehler passiert, wird Null als Rückgabeparameter übergeben. Als Übergabeparameter wird der Bearer-Key übergeben. Die Adresse des Get-Request ist in der ConfigUrl gespeichert.

```
using (HttpClient client = new HttpClient())
{
    // HTTP Header erstellen, hier wird der Bearer-Key übergeben
    client.DefaultRequestHeaders.Accept.Clear();
    client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", bearer);

    // Post-Anfrage senden
    HttpResponseMessage response = await client.GetAsync(ConfigUrl.azureDataset);


    if (response.IsSuccessStatusCode)
    {
        //Antwort als Json-String einlesen
        string jsonResponse = await response.Content.ReadAsStringAsync();

        //Json zu Target
        target = JsonUtility.FromJson<Targets>(jsonResponse);
```

public void loadZipFiles (string base64files)

Diese Methode ist dafür da, den base64String zu einem Zip zu konvertieren, um auf die Files zugreifen zu können. Um ein Zip-File erstellen zu können, muss der base64String vorher zu einem byte Array umgewandelt werden. *„Convert.FromBase64String(<string>)“*
Mit File.WriteAllBytes(<Pfad>, <byte[]>) kann nun ein Zip-File erstellt werden.

Danach wird auf die Unzip(<Pfad>) weitergeleitet, damit dort das Zip-File extrahiert wird.

public void Unzip()

Die zuvor konvertierte Zip-Datei wird entpackt. Dafür wird System.IO.Compression importiert.

```
// Öffnen des ZipFiles
using (ZipArchive archive = ZipFile.OpenRead(ConfigPath.getPathAndroidZIP()))
{
    // Jeder Eintrag/File wird extrahiert
    foreach (ZipArchiveEntry entry in archive.Entries)
    {
        // genauer Pfad wird angelegt
        string entryPath = Path.Combine(extractPath, entry.FullName);

        Directory.CreateDirectory(Path.GetDirectoryName(entryPath));

        //Extrahieren des Eintrages/Files an den gewünschten Pfad
        entry.ExtractToFile(entryPath, true);
    }
}
```

11.6.9.7. ConfigUrl-Klasse

Die Klasse ConfigUrl speichert alle benötigten Adressen, um entweder auf die lokale API oder um auf die API in Azure zugreifen zu können.

localDataset
`https://localhost:7194/api/model-targets?includeDatasets=true`

localLogin
`https://localhost:7194/api/Authentication/installer-login`

azureLogin
`https://project-ar.azurewebsites.net/api/auth/installer-login`

azureDataset
`https://projectar.azurewebsites.net/api/modeltargets?includeDatasets=true`

11.6.9.8. ConfigPath-Klasse

In der ConfigPath-Klasse werden die Pfade für Android und Windows gespeichert. Es wird hauptsächlich mit Application.persistenDataPath oder Application.streamingAssetsPath gearbeitet, da man dort Berechtigungen zum Erstellen und Bearbeiten hat.

PersistenDataPath: (Android)
`/storage/emulated/<userid>/Android/data/<packagename>/files`

PersistenDataPath: (Windows/Unity)
`%userprofile%\AppData\LocalLow\<companyname>\<productname>`

StreamingAssetsPath: (Windows)
UnityProjekt: `project-ar\Assets\StreamingAssets\`

public static string getPathAndroidVuforia ()

Pfad, zum Speichern der .dat und .xml Datei im Android System

public static string getPathAndroidZIP()

Pfad, zum Speichern des Zip-Files im Android System
Name: targets.zip

public static string getPathWindowsVuforia()

Pfad, zum Speichern der .dat und .xml Datei in Windows

public static string getPathWindowsZIP()

Pfad, zum Speichern des Zip-Files in Windows

11.6.10. Erkennen eines Model Targets

11.6.10.1. Einleitung

In diesem Kapitel geht es um die automatische Erkennung eines Models mithilfe eines Model Targets. Hierbei kann dann der Installateur, dass Model erkennen lassen und wichtige Informationen wie Anleitungen oder Produktbeschreibungen in der AR Umgebung anzeigen lassen.

11.6.10.2. Aufbau ModelTargetScene

Die ModelTargetScene ist folgendermaßen aufgebaut.

Abbildung 55: Aufbau ModelTargetScene in Unity

ARCamera

Beinhaltet eine Kamera, die den Canvas/Seite anzeigt. Außerdem ist diese Kamera dafür zuständig Vuforia Komponenten anzeigen zu lassen.

Navigator:

Ein GameObject, dass als Navigator fungiert, indem es die Navigator-Datei beinhaltet. Solche GameObjekte werden mit einer Datei angelegt, damit man zum Beispiel mittels Buttons auf Methoden der Datei zugreifen kann.

ModelTarget:

Das ModelTarget-GameObject dient zum Erkennen eines Objektes. Als Komponente verwendet dieses Objekt, die Observer-Datei. Übergeben wird das Prefab, welches neben dem Erkannten angezeigt wird. Und ein Status-Textfeld, damit der aktuelle Status angezeigt wird.

Canvas:

Wie in der ModelView dient der Canvas zur visuellen Darstellung vom Return-Button und zum Anzeigen eines Status. Der Button verweist auf den Navigator, um zur ModelView-Scene zurückzukehren. Background ist eine schwarze Fläche am unteren Rand des Bildschirms Hier befindets sich ein Status-Feld, der den aktuellen Status des Erkennens während der Laufzeit angibt. Im Text steht „Status“ als Platzhalter.

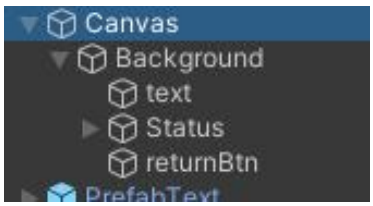


Abbildung 56: Aufbau Canvas-ModelTargets

PrefabText:

Ist eine Vorlage, welches neben dem Erkannten angezeigt wird. Dabei soll der Text, die Description sein, die wichtigen Informationen beinhalten.

11.6.10.3. Observer-Klasse

Die Klasse Observer, erstellt einen Model-Target und kann mittels dem Vuforia Behaviour, dass entsprechende Model erkennen. Zuvor werden die bestimmten Dateien in den entsprechenden Ordner geladen. Im Observer muss dann noch die Database und der Targetname eingestellt werden.

Attribute:

databaseName: String
Angeben des Database Namen, sollte eine .xml-Datei sein

targetName:String
Der TargetName wird verwendet, um das korrekte Modell zu erkennen. Wird in der .xml-Datei gespeichert, welcher TargetName das Modell besitzt.

statusUI & myModelPrefab
Übergabeparameter, Status von Vuforia wird angezeigt und das myModelPrefab wird als Vorlage für die Anleitung benutzt

Methoden:

void Start()
wird beim Anzeigen der Seite als erstes Aufgerufen. In unseren Fall setzen wir die VuforiaInstanz und wenn Vuforia aktiv wird dann soll die Methode „OnVuforiaInitialized“ aufgerufen werden.

```
VuforiaApplication.Instance.OnVuforiaInitialized += OnVuforiaInitialized;
```

void OnVuforiaInitialized(VuforiaInitError error)
In dieser Methode wird überprüft, ob ein Fehler bei der Vuforia Engine aufgetreten ist.
Wenn nicht, dann soll die OnVuforiaStarted() Methode aufgerufen

public String buildPath ()
Die buildPath Methode erstellt einen Pfad, wo die Dateien für die Erkennung des Objektes liegen.
Mittels ConfigPath-Klasse wird der Pfad erstellt

void OnVuforiaStarted()
Die Methode ist zuständig für die Aktivierung der Erkennung des Objektes. Dazu ruft man zuerst die buildPath() Methode auf und danach erstellt man ein ModelTargetBehaviour. Die ModelTargetBehaviour Klasse wird von Vuforia zur Verfügung gestellt und somit können Objekte erkannt werden. Bei der Methode CreateModelTarget() muss zuerst der Pfad und dann der targetName übergeben werden.
Zum Schluss wird die TrackingOptimization gesetzt. In unserem Fall verwenden wir „Default“.
Wenn sich der Status des erkannten ModelTargets ändert, so wird die OnTargetStatusChanged Methode aufgerufen.

```
modelTarget = VuforiaBehaviour.Instance.ObserverFactory.CreateModelTarget(
       //"storage/emulated/0/Android/data/com.project.projectar/files/
       MTDataset.xml", path, targetName);

modelTarget.SetTrackingOptimization(TrackingOptimization.LOW_FEATURE_OBJECTS);

modelTarget.OnTargetStatusChanged += OnTargetStatusChanged;
```

void OnTargetStatusChanged(ObserverBehaviour behaviour, TargetStatus status)

Diese Methode wird benützt, um den jeweiligen Status anzuzeigen und um die Beschreibung neben den erstellten ModelTarget anzuzeigen.

```
statusUI.text = statusUI.text + status.Status;
myModelPrefab.GetComponentInChildren<TextMeshProUGUI>().text =
Navigator.target.description;
mMyModelObject = Instantiate(myModelPrefab, modelTarget.transform);
mMyModelObject.transform.pos = new Vector3(0.1f, 0.1f, 0.1f);
mMyModelObject.SetActive(true);
```

XML & Dat – Dateien

Durch die beiden Dateien kann Vuforia ein Model erkennen. Hauptsächlich wird die. Dat -Datei verwendet, um ein Modell zu tracken. In der XML-Datei werden wichtige Informationen gespeichert. Informationen sind zum Beispiel, Tracking Mode, TargetName, UpVector.

Asynchrone Programmierung

In unserem Programm wird auf asynchrone Methoden zurückgegriffen, damit im Hintergrund die ModelTargets geladen werden und die App nicht angehalten wird. Durch asynchrone Methoden wird auch gewährleistet, dass keine Threads blockiert werden.

Als Rückgabewert wird die Task-Klasse verwendet, die man für asynchrone Methoden verwenden soll. Eine asynchrone Methode wird mit async deklariert.

```
public async Task<Targets> getModelTargets(string bearer)
```

Await wird beim Aufruf einer asynchronen Methode verwendet. Mit diesem Operator hält man an dieser Stelle an bis der asynchrone Vorgang abgeschlossen ist.

```
Targets targets = await controller.getModelTargets(bearer);
```

11.6.10.4.Probleme beim Implementieren der Erkennung

Wenig Dokumentation

Vuforia bietet wenig Informationen über die Model Target Technologie an. Es wird grundsätzlich alles beschrieben, wie es funktioniert, aber meist nur für statische Beispiele. Es wird wenig darüber berichtet, wie man mit Code ein Model Target oder eine Guide View erstellen kann. Da wir selbst noch nie mit Augmented Realtiy oder mit Vuforia gearbeitet haben, war es schwierig sich in das Thema einzuarbeiten.
Die meisten Informationen findet man über statische Beispiele. Das bedeutet, man lädt oder scannt ein 3D-Modell und fügt das Modell in die AR-Umgebung ein. Das Modell kann im Unity-Editor bearbeitet werden und genau auf die Anwendung angepasst werden.
Unsere Software soll verschiedene Model Targets zur Laufzeit verwenden können.

Pfadzugriff

Ein weiteres Hauptproblem waren die Dateizugriffe. Denn die XML- und DAT-Dateien müssen in den entsprechenden Pfad gespeichert werden. Als Fehlermeldung bekommt man „Access denied“ zurück. Deswegen versuchten wir zuerst die Schreib-und Leseberechtigungen zu setzen. Nach längerer Recherche sind haben wir bemerkt, dass es gar nicht an den Berechtigungen liegt, sondern dass der Pfad nicht richtig eingeben worden ist. Nachdem wir den Pfad richtig verwendet haben. Ist trotzdem der gleiche Fehler aufgetreten, da in selbst erstellten Verzeichnissen keine Dateien erstellt werden darf. Deswegen mussten wir auf die Application.persistenDataPath zugreifen.

Base64String zu ZIP-Datei

Eine weiter Hürde war das Konvertieren des base64String zu einer ZIP-Datei, die dann wieder entpackt werden musste, denn wir zuerst verwendeten wir die falsche Konvertierungsmethode. Dabei wurde ein ZIP-File erstellt, dass nicht extrahierbar war.

Größe des 3D-Modells

Eine sehr große Rolle spielt die Größe des Model Targets. Denn umso genauer die Länge, Breite und Höhe des 3D-Modells ist, desto besser kann das reale Objekt erkannt werden. Zudem wird die Beschreibung in Bezug zur Größe vom Model Target gesetzt. Das bedeutet, wenn das generierte Modell größer ist als das physische Objekt, dann wird die Beschreibung in größeren Abständen zum Objekt platziert. Bei unserem 3D-Modell Generator kann es passieren, dass die Größe des Model Targets nicht dem realen Objekt entspricht.

12. Ergebnis

12.1. Neue Objekte scannen

Im Endeffekt kann das Ergebnis des Objekt-Scans in 2 Teile eingeteilt werden. Zuerst erhalten wir den Abgegrenzten Teil in der Android Applikation. Hier wird der Benutzer oder die Benutzerin der App dazu aufgefordert Bilder eines Objektes zu machen. Grundsätzlich ist zu erwähnen, dass der Ablauf zur Generierung eines Objektes innerhalb der App angestoßen wird, und von der Rest API verwaltet und überwacht wird.

12.1.1. Android App

Wir erhalten insgesamt 3 Bildschirme auf welchen unterschiedlichen Aktionen getätigt werden können. Wie in Abbildung Abbildung 57 zu sehen, gibt es folgende 3 Szenen:

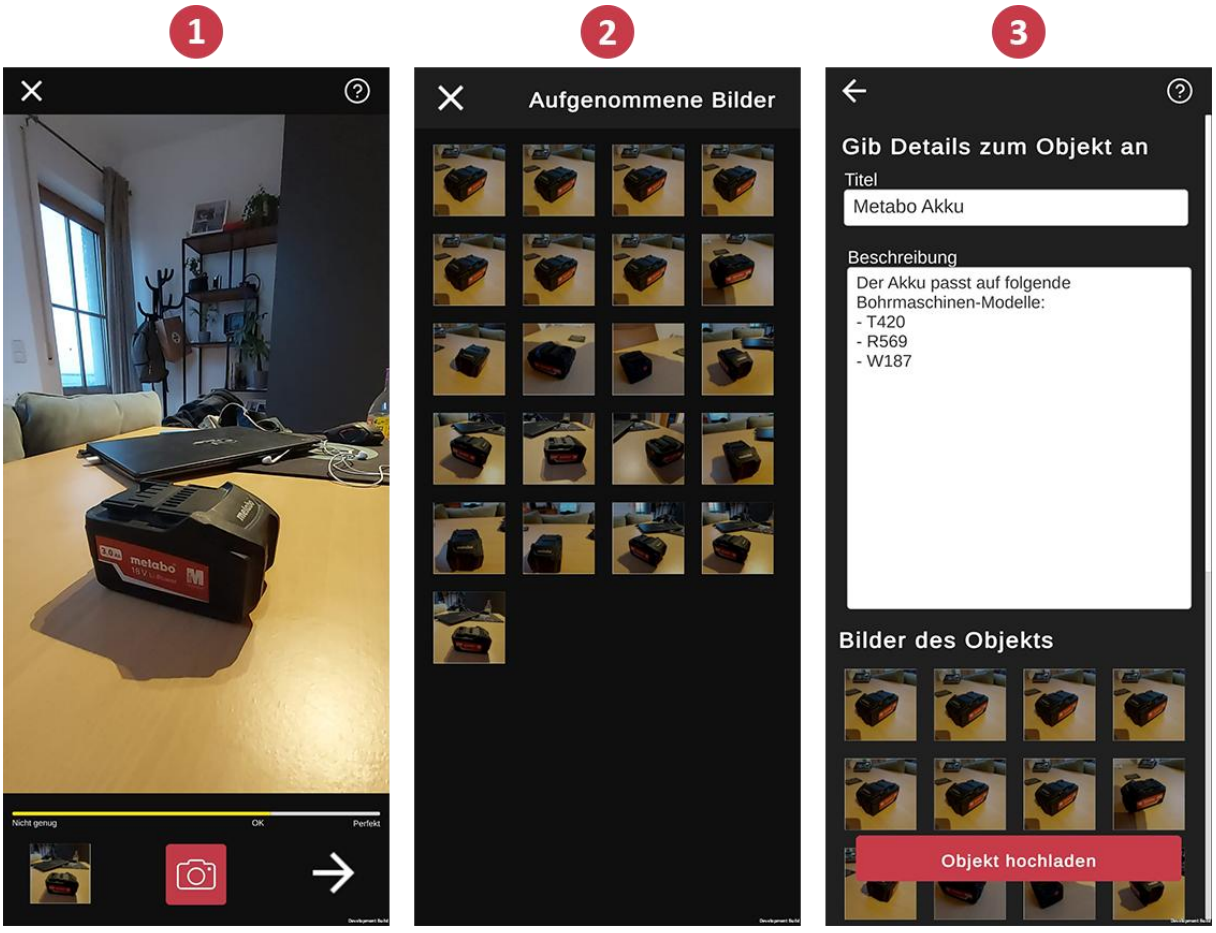


Abbildung 57: Bildschirme der Object-Scanning

(1) Objekt Scan

Um Bilder aufzunehmen, muss hierbei nur der untere Kamera Knopf betätigt werden. Grundsätzlich ist von uns eine Mindestanzahl der Fotos festgelegt. Diese liegt bei 20, an der unteren Fortschrittsanzeige, kann man den derzeitigen Stand ablesen. Des Weiteren, wird links unten das

zuletzt aufgenommene Bild angezeigt. Sobald die Untergrenze der gemachten Aufnahmen erreicht wurde, erscheint rechts vom Kamera-Knopf, ein Pfeil, um Prozess fortzusetzen.

(2) Aufgenommene Bilder

Um bei der Erfassung eines falschen oder unvorteilhaften Bildes Korrekturen vornehmen zu können, existiert diese Seite. Hierbei können diese durch Drücken noch einmal angesehen werden und, falls gewünscht, durch eine weitere Eingabe, diese entfernt werden. Eine Korrektur ist nach dem Fortsetzen in (1) Objekt Scan nicht mehr möglich, außer man navigiert wieder zurück zur ersten Szene.

(3) Absenden

Abschließend kann der Benutzer oder die Benutzerin noch den Titel und die Beschreibung des Objektes festlegen und abschließend die aufgenommenen Bilder zu bestätigen. Nach drücken des Absenden-Knopfes, werden alle Eingaben zur weiteren Verarbeitung an die Rest API gesendet. Der Bediener oder die Bedienerin der Applikation muss nach Absenden nichts mehr tun. Die Generierung, Training und Speicherung werden von der Rest API automatisch übernommen.

12.1.2. Generator Rest-API

Um aus den 2-dimensionalen Bildern ein 3D-Modell zu erhalten, existiert diese Generator Rest API, welche von der Haupt Rest API aufgerufen wird. Diese dient eher als Vermittler und sendet, die Ergebnisse immer weiter, bis am Ende ein Dataset erstellt wird, durch welches man die Objekte innerhalb der Applikation wieder erkannt werden können.

Grundsätzlich stehen 2 Endpoints zur Verfügung, welche von der eben genannten Haupt Rest API verwendet wird.

(1) Generierung

Hierbei handelt es sich um jene Schnittstelle, auf welche die aufgenommenen Bilder aus 12.1.1 (1) Objekt Scan übertragen werden. Nach erfolgreicher Übertragung wird die Generierung gestartet, und eine UUID wird zurückgegeben.

(2) Status

Durch die in (1) Generierung zurückgegebene UUID, kann mithilfe dieses Endpunktes jederzeit der Status abgefragt werden. Grundsätzlich wird man hierbei informiert, ob die Generierung fehlgeschlagen ist, noch den Prozess durchführt oder bereits fertig gestellt wurde. Ist der Ablauf fertig, kann jederzeit durch die festgelegte Identifikationsnummer das Objekt heruntergeladen werden.

12.2. ASP.NET Core web API

Die erhaltene Schnittstelle stellt die Business logic für die gesamte Anwendung zentralisiert zur Verfügung und ist mittels Azure gehosted. Die API ist unter https://project-ar.azurewebsites.net erreichbar.

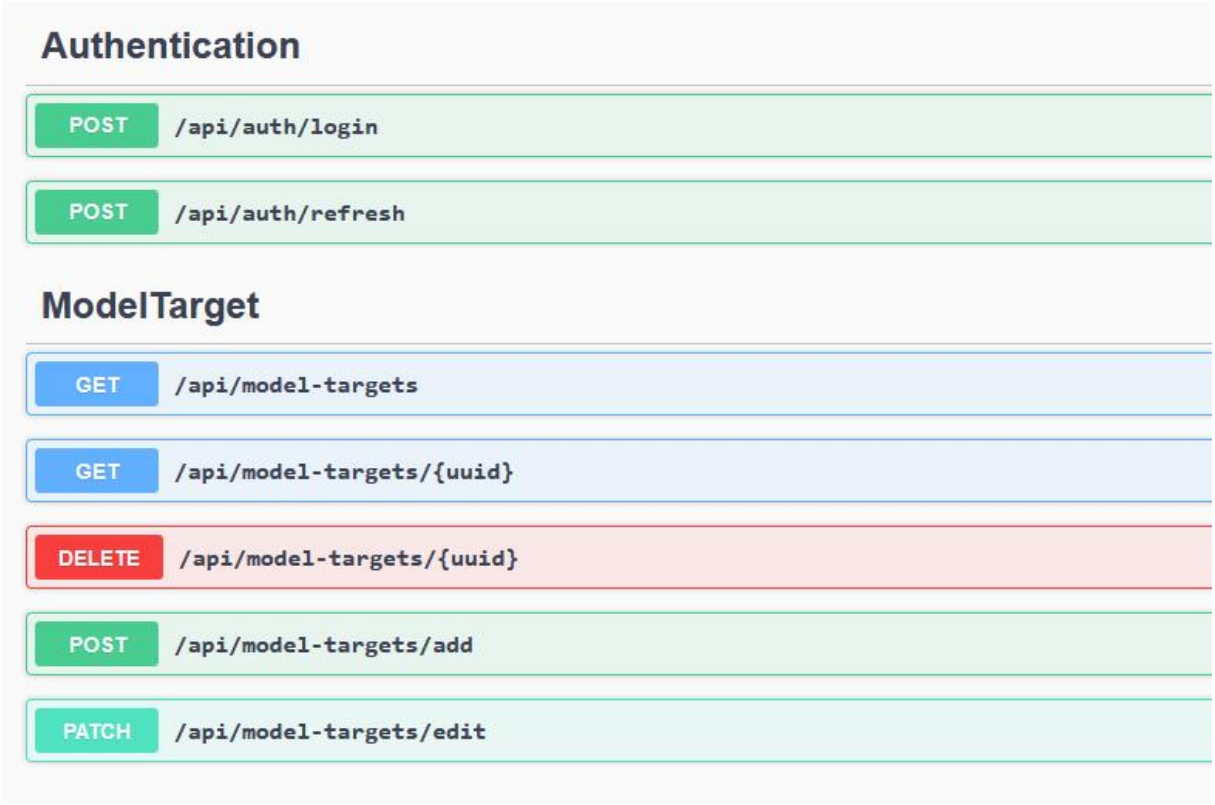


Abbildung 58: Ergebnis REST-API

Die zur Verfügung stehenden Endpoints sind auf obiger Abbildung zu erkennen.

(1) Authentifizierung

Um unbefugten Zugriff auf die API zu verhindern, wurde wie bereits erwähnt ein JWT-Prozess implementiert. Um sich nun anzumelden, und damit einen Access-Token für den Zugriff auf die Produktiv-Endpoints zu erhalten, kann der „POST /api/auth/login“ Endpoint aufgerufen werden.

Der „POST /api/auth/refresh” Endpoint ist dazu da, um mittels des erhaltenen und gespeicherten Refresh-Tokens ein neues Access-Token zu erhalten und damit den Silent-Login zu ermöglichen.

(2) Model-Targets

Diese Endpoints sind zum Hinzufügen, Erhalten, Bearbeiten und Löschen der Model-Targets zuständig.

„GET /api/model-targets“ – Gibt alle gespeicherten Model-Targets inklusive oder bei Bedarf auch exklusive aller Datasets als base64-Strings.

„GET /api/model-targets/{uuid}“ – Gibt das per „uuid” definierte Model-Target zurück.

„DELETE /api/model-targets/{uuid}" – Löscht das mittels „uuid" angegebene Model-Target.

„POST /api/model-targets/add" – Fügt das im Body des Requests angegebene Model-Target hinzu.

"PATCH /api/model-targets/edit" – Bearbeitet das im Request-Body spezfifizierte Model-Target.

12.3. Microsoft SQL Server Datenbank

Die Datenbank ist ebenfalls mit Azure gehosted und speichert wie in Kapitel 11.3 beschrieben, alle Benutzer, Model-Targets, Keys (wie beispielweise das Client-Secret für den Zugriff auf Vuforia) und momentan gültigen Refresh-Tokens.

Das Konzept der Datenbank beläuft sich auf eine relationale Darstellung und erfüllt daher auch die Konsistenzanforderung der Nutzdaten.

12.4. Angular Webapplikation für Administratoren

Die Angular Webapplikation dient dazu, die Model-Targets zu verwalten. Mithilfe der Applikation können Model-Targets angesehen, bearbeitet, und gelöscht werden.

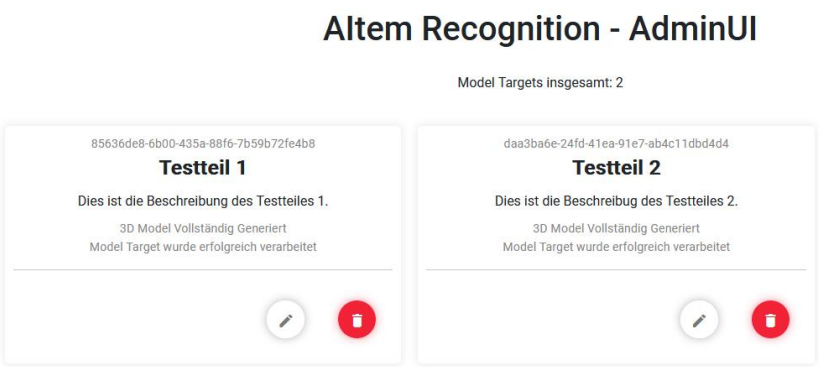


Abbildung 59: AdminUI Ergebnis Übersicht Model-Targets

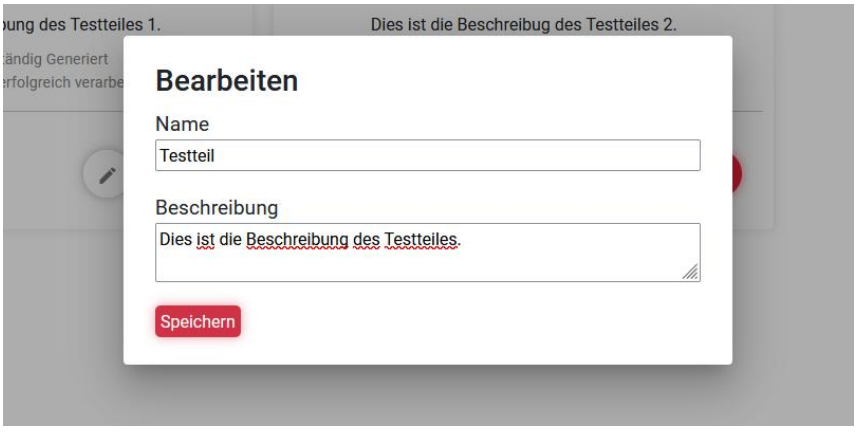


Abbildung 60: AdminUI Ergebnis Bearbeiten Model-Target

Außerdem ist hier der Status des Model-Targets einsehbar, also ob ein Fehler bei der Generierung oder beim Training entstanden ist, oder ob alles erfolgreich abgelaufen ist.

Durch dieses Tool können Administratoren beispielsweise veraltete Model-Targets entfernen, oder auch Beschreibungen aktualisieren und verbessern.

12.5. Objekterkennung und Informationen anzeigen

Um Objekte erkennen zu können, entwickelten wir eine Android-App, die Augmented Reality unterstützt. Die Android-App wurde in Unity entwickelt und konnte mittels dem Vuforia Package Augmented Reality Technologien verwenden.

Als erstes kann der Benutzer auswählen, ob er ein Objekt scannen oder erkennen will. Von dieser Seite kommt der Benutzer zur Auflistung aller gespeicherten Model Targets, wenn er auf den Button Objekt erkennen drückt.

Die Model Targets werden asynchron hereingeladen. Bei jedem Objekt steht der Status dabei, denn es kann sein, dass das Modell gerade nicht verfügbar ist. Zum Beispiel weil es gerade von Vuforia trainiert wird oder weil es Probleme bei der Generierung des 3D-Modells gibt.

Klickt man auf ein Model Target, dann wird man zur Erkennungs-Seite weitergeleitet. Unten sieht man den Status von der derzeitigen Erkennung. Der Status „Tracked“ bedeutet, dass das Objekt erkannt wurde. Wird das Modell erkannt, so wird die Beschreibung angezeigt. In diesem Beispiel sieht man ein Samsung S20 FE. Mit dem Zurück-Button kommt man wieder auf zur Auflistung aller Model Targets. Dort kann man sich ein neues Model Target aussuchen, wenn man Hilfe benötigt.

Abbildung 60: App-Auswahlseite

Abbildung 61: App-Auflistung aller Model Targets

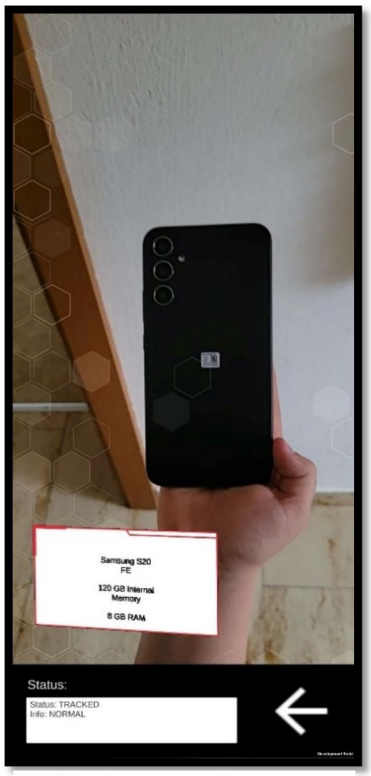


Abbildung 61: App-Erkennung von einem Handy

13. Resümee

In unserer Diplomarbeit AItem Recognition, entstand in der Zusammenarbeit dem Unternehmen ITPRO und wurde entwickelt, um die Effizienz der Montagearbeiten zu verbessern. Außerdem verfolgt unser Produkt das Ziel, eine innovative Augemted-Reality-Lösung zu bieten, welche es den Mitarbeitern unseres Kooperationspartners ermöglicht, Maschinenteile automatisch zu erkennen und relevante Informationen während der Arbeit zu erhalten.

Besonders hervorzuheben ist die positive Zusammenarbeit mit unserem Auftraggeber. Durch regelmäßigen Austausch und effektive Kommunikation konnten wir gemeinsam die Herausforderungen bewältigen und Lösungen erarbeiten. Außerdem erhielten wir so laufend Rückmeldung und konnten so unser Produkt entsprechend den Anforderungen entwickeln.

Je länger das Projekt dauerte, desto komplexer wurde die Struktur der Diplomarbeit. Unsere Diplomarbeit deckt einen großen Bereich an Technologien ab. Augmented Reality, zwei APIs, 3D-Modell Generator, Android-App und ein Dashboard in Angular kommen in unserem Projekt vor. Trotz den verschiedensten Technologien/Themen schafften wir es den Überblick zu behalten. Mit einer ausführlichen Planung und Dokumentation wusste jedes Teammitglied über den Inhalt des Projektes Bescheid.

Im Laufe der Implementierung stießen wir jedoch auf einige Herausforderungen. Zu erwähnen ist die knapp ausfallende Dokumentation der Vuforia Model Target Web API. Dadurch traten einige Probleme während der Entwicklung auf, welche durch eine saubere Beschreibung der Funktionen von Vuforia, hätten verhindert werden können. Für eine längere Dauer der Entwicklung ist auch unsere Entwicklungsumgebung, Unity, Schuld. Da das Austesten einer Funktion auf einem mobilen Endgerät einige Zeit dauert, wurden Fehler erst später entdeckt. Bis die Applikation auf ein Android Gerät übertragen wird, vergingen meist 5-10 Minuten. Aufgrund dessen konnten Funktionen erst später getestet werden und eine Fehlersuche gestaltete sich als sehr zeitaufwendig.

Zusammenfassend kann aber gesagt werde, dass wir trotz einiger Schwierigkeiten und der Großteils unbekannten Technologien, ein funktionsfähiges Produkt erstellen konnten. Weiters ist zu erwähnen, dass sowohl unser dreiköpfiges Projektteam als auch unser Ansprechpartner, stellvertretend für die Firma ITPRO, mehr als zufrieden ist mit dem Ergebnis. Laut unserem Auftraggeber wären auch bereits einige Kunden an dem Produkt interessiert. Insbesondere wir selbst sind stolz darauf, so ein umfangreiches Vorhaben erfolgreich in die Tat umgesetzt zu haben.

14. Aufgabenverteilung

Zur Übersicht unserer Aufgabenverteilung liegt folgendes Diagramm (Abbildung 62) vor. Allgemein ist dazu zu sagen, dass die Bereiche abgegrenzt und unabhängig voneinander entwickelt werden konnten. Am Ende war es das Ziel, alle Komponenten zu verbinden und ein voll funktionsfähiges Produkt zu erhalten.

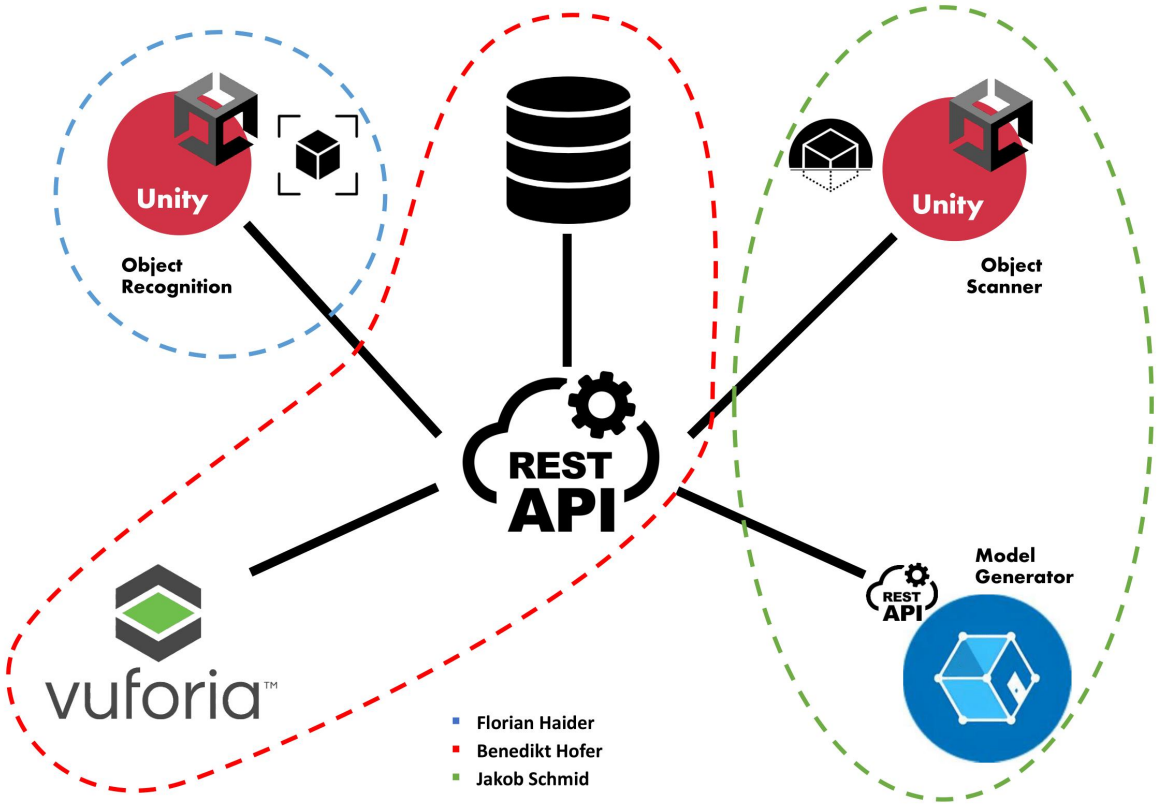


Abbildung 62: Aufgabenverteilung der Implementierung

14.1. Hofer Benedikt

Die technischen Aufgaben bestanden darin, die REST-API vollständig zu entwickeln und die dazugehörige Datenbank zu konzipieren und umzusetzen. Aufgrund des gesamtzeitlichen Zustandes des Projekts konnte auch noch eine zusätzliche Webapplikation implementiert werden, die das Verwalten der Model-Targets vereinfacht. Aus diesem Grund wurden folgende Kapitel der Implementierung und die dazugehörigen Unterkapitel von ihm verfasst:

- 11.2 REST-API
- 11.3 Datenbank
- 11.4 Webapplikation für Administratoren

Zusätzlich wurden auch die folgenden Kapitel verfasst:

- 5 Kurzfassung
- 8.3.2 Webapplikation für Administratoren
- 9.1.1 - 9.1.7
- 9.2 Verwendete Entwicklungssysteme
- 9.3 Verwendete Bibliotheken und Plug-Ins
- 11.1 Technischer Überblick

14.2. Schmid Jakob

In der Entwicklung bestand die Aufgabe darin den Prozess des Scannens eines Objektes umzusetzen. Hierbei handelt es sich sowohl um das Erfassen der Aufnahmen innerhalb der Android Applikation, zu finden in Kapitel 11.5.2, als auch um die Generierung und Verwaltung der daraus resultierenden dreidimensionalen Objekte, welche später für die Erkennung verwendet werden. Grundsätzlich bedeutet das, dass der gesamte Bereich der Fotogrammetrie durch ihn abgedeckt ist. Dies umfasst folgende Kapitel und deren Unterkapitel:

- 8.3.1 3D-Modelle einscannen
- 9.1.8 Fotogrammetrie Software
- 11.5 Object Scanning
- 12.1 Neue Objekte scannen

Weiters wurden folgende Kapitel verfasst:

- 6 Abstract
- 8.2 Zielsetzung
- 10.3.2 Unsere Boards
- 8.4 Projektumfeld

14.3. Haider Florian

Die Hauptaufgabe bestand darin, zuvor gescannte Objekte mittels 3D-Modelle in der Augmented Reality App zu erkennen. Dazu werden alle gespeicherten Model Targets aufgelistet. In der internen SQLite-Datenbank werden Informationen über den Installateur gespeichert. Damit dieser Zugriff auf die Rest-API erhält, muss sich der Installateur vorher anmelden. Außerdem sind Model Targets ein großer Bestandteil des Kapitels. Es musste sich über den Aufbau und über die Verwendung der Model Targets informiert werden. Zuständig war Florian Haider ebenfalls für die Protokollierung der Meetings mit dem Auftraggeber und dem Diplomarbeitsbetreuer.
Wie man zur Lösung des Erkennens gekommen ist, wird im Kapitel und in den 11.6 Object Recognition beschrieben. Weitere Kapiteln sind:

- 8.1 Motivation
- 8.4 Projektumfeld
- 9.1 Technologien

15. Quell-/Literaturverzeichnis

3DFLOW SRL. *3DF ZEPHYR*. [online]. [Accessed 24 Mar 2024]. Available from World Wide Web: <https://www.3dflow.net/3df-zephyr-photogrammetry-software/>

ALICEVISION DOCKERHUB. *Meshroom Image: Docker Hub*. [online]. [Accessed 24 Mar 2024]. Available from World Wide Web: <https://hub.docker.com/r/alicevision/meshroom/tags>

ALICEVISION. *Meshroom*. [online]. [Accessed 9 Mar 2024]. Available from World Wide Web: <https://alicevision.org/#meshroom>

ALICEVISION. *Meshroom Dokumentation*. [online]. [Accessed 10 Mar 2024]. Available from World Wide Web: <https://meshroom-manual.readthedocs.io/en/latest/index.html>

ASANA, INC. *Scrumban*. [online]. [Accessed 29 Mar 2024]. Available from World Wide Web: <https://asana.com/de/resources/scrumban>

BAUMBERG, Adam. 2002. *Blending images for texturing 3D*. Berkshire, England.

BLENDE ZEIT ISO. *Blende Zeit Iso*. [online]. [Accessed 9 Mar 2024]. Available from World Wide Web: <https://blende-zeit-iso.de/welche-brennweite-hat-meine-iphone-kamera-und-wieso-ist-das-wichtig/>

BLOC SHOP. *Gleek ER Symbols Notations*. [online]. [Accessed 30 Mar 2024]. Available from World Wide Web: <https://www.gleek.io/blog/er-symbols-notations>

DATATAB E.U. GRAZ. *k-Means Clusteranalyse: DATAtab*. [online]. [Accessed 24 Mar 2024]. Available from World Wide Web: <https://datatab.de/tutorial/k-means-clusteranalyse>

DAVID G. LOWE. 2004. *Distinctive Image Features*. Vancouver, B.C., Canada.

DAVID NISTÉR AND HENRIK STEWÉNIUS. 2006. *Scalable Recognition with a Vocabulary Tree*. Kentucky, UK.

DIE LEADAGENTEN. *Was ist TF-IDF: Die Leadagenten*. [online]. [Accessed 16 März 2024]. Available from World Wide Web: <https://www.die-leadagenten.de/online-marketing-glossar/tf-idf/>

ENGINE, PTC Vuforia. *Recommended Devices*. [online]. [Accessed 24 Aug 2023]. Available from World Wide Web: <https://developer.vuforia.com/library/platform-support/recommended-devices >

FLYING DONUT SOFTWARE P.C.. *Flying Donut*. [online]. [Accessed 29 Mar 2024]. Available from World Wide Web: <https://www.flyingdonut.io/>

FORMLABS GMBH. *Formlabs*. [online]. [Accessed 5 Feb 2024]. Available from World Wide Web: <https://formlabs.com/de/blog/photogrammetrie-leitfaden-und-software-vergleich/>

GISWIKI. *Delaunay-Triangulation: GisWiki*. [online]. [Accessed 22 Mar 2024]. Available from World Wide Web: <http://giswiki.org/wiki/Delaunay-Triangulation>

GOOGLE LLC. *Angular*. [online]. [Accessed 30 Mar 2024]. Available from World Wide Web: <https://angular.io/>

IANA. *jwt*. [online]. [Accessed 24 Mar 2024]. Available from World Wide Web: <https://www.iana.org/assignments/jwt/jwt.xhtml>

INCREDIBUILD SOFTWARE LTD. *CUDA Incredibuild*. [online]. [Accessed 23 Mar 2024]. Available from World Wide Web: <https://www.incredibuild.com/integrations/cuda>

INTERNET ENGINEERING TASK FORCE (IETF). *rfc-editor*. [online]. [Accessed 24 Mar 2024]. Available from World Wide Web: <https://www.rfc-editor.org/rfc/rfc7519.html>

ITPRO CONSULTING & SOFTWARE GMBH. *ITPro*. [online]. [Accessed 25 Mar 2024]. Available from World Wide Web: <https://www.itpro.at/>

JEFFREY S. BEIS AND DAVID G. LOWE. 1997. *Shape Indexing Using Approximate Nearest-Neighbour Search in*. Vancouver, B.C., Canada.

JÖREN CARSTENS UND HAUKE MARTENS. 2014. *Informatik Uni Hamburg*. [online]. [Accessed 10 Mar 2024]. Available from World Wide Web: <https://kogs-www.informatik.uni-hamburg.de/~seppke/content/teaching/sose16/bvproj/CM-SIFT15.pdf>

LOBO, Teresa. *Medium*. [online]. [Accessed 22 Mar 2024]. Available from World Wide Web: <https://medium.com/@loboateresa/understanding-structure-from-motion-algorithms-fc034875fd0c>

MICROSOFT. *Azure*. [online]. [Accessed 2 Apr 2024]. Available from World Wide Web: <https://azure.microsoft.com>

MICROSOFT. *Nuget Packages*. [online]. [Accessed 30 Mar 2024]. Available from World Wide Web: <https://www.nuget.org/packages/System.IdentityModel.Tokens.Jwt/>

MICROSOFT. *SQL Server*. [online]. [Accessed 2 Apr 2024]. Available from World Wide Web: <https://www.microsoft.com/de-at/sql-server>

NAYAR, Shree K. 2022. *SIFT Detector*. New York.

NVIDIA CORPORATION. *About CUDA Nvidia*. [online]. [Accessed 23 Mar 2024]. Available from World Wide Web: <https://developer.nvidia.com/about-cuda>

OKTA INC. *jwt.io*. [online]. [Accessed 24 Mar 2024]. Available from World Wide Web: <https://jwt.io/introduction/>

OPENEXR. *OpenEXR*. [online]. [Accessed 22 Mar 2024]. Available from World Wide Web: <https://openexr.com/en/latest/>

PINECONE SYSTEMS, INC. *Bag of Visual Words: Pinecone*. [online]. [Accessed 16 Mar 2024]. Available from World Wide Web: <https://www.pinecone.io/learn/series/image-search/bag-of-visual-words/>

PTC INC. *API References*. [online]. [Accessed 30 Mar 2024]. Available from World Wide Web: <https://developer.vuforia.com/sites/default/files/references/mt-web-api-openapi.html>

PTC INC. *Vuforia*. [online]. [Accessed 24 Mar 2024]. Available from World Wide Web: <https://www.ptc.com/de/products/vuforia>

PTC INC. *Vuforia Developer*. [online]. [Accessed 2 Apr 2024]. Available from World Wide Web: <https://developer.vuforia.com/>

PTC INC. *Vuforia Engine*. [online]. [Accessed 24 Mar 2024]. Available from World Wide Web: <https://www.ptc.com/en/products/vuforia/vuforia-engine/ar-app-development>

PTC INC. *Vuforia Model Target Web API*. [online]. [Accessed 24 Mar 2024]. Available from World Wide Web: <https://developer.vuforia.com/library/web-api/model-target-web-api>

PTC. *Vuforia*. [online]. [Accessed 9 Mar 2024]. Available from World Wide Web: <https://developer.vuforia.com/>

REYDAR. *How Does Augmented Reality Work?*. [online]. [Accessed 3 Aug 2023]. Available from World Wide Web: <https://www.reydar.com/how-does-augmented-reality-work/>

SCHOENBERGER, Johannes L. *SOLMAP*. [online]. [Accessed 25 Mar 2024]. Available from World Wide Web: <https://demuc.de/colmap/>

SKETCH.MEDIA GMBH. *Sketch Media Brennweite*. [online]. [Accessed 9 Mar 2024]. Available from World Wide Web: <https://sketch.media/projekte/blog/foto/203-brennweite.html>

TECHTARGET, INC. *Techtarget ERD*. [online]. [Accessed 30 Mar 2024]. Available from World Wide Web: <https://www.techtarget.com/searchdatamanagement/definition/entity-relationship-diagram-ERD>

THE MATHWORKS, INC. *What is Structure from Motion? MathWorks*. [online]. [Accessed 22 März 2024]. Available from World Wide Web: <https://de.mathworks.com/help/vision/ug/what-is-structure-from-motion.html;jsessionid=670167a4c211211ab5057408c514>

UXWING. *uxwing.com*. [online]. [Accessed 9 Mar 2024]. Available from World Wide Web: <https://uxwing.com/>

VERTABELO SA. *Vertabelo*. [online]. [Accessed 24 Mar 2024]. Available from World Wide Web: <https://vertabelo.com/blog/crow-s-foot-notation/>

VUFORIA, PTC. *MTG User Guide*. [online]. [Accessed 10 Aug 2023]. Available from World Wide Web: <https://developer.vuforia.com/library/objects/model-target-generator-user-guide>

VUFORIA, PTC. *Vuforia Engine Getting Started*. [online]. [Accessed 24 Jul 2023]. Available from World Wide Web: <https://developer.vuforia.com/library/>

WORLD WIDE WEB CONSORTIUM. *w3 org*. [online]. [Accessed 24 Mar 2024]. Available from World Wide Web: <https://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.2>

ZEBRA TECHNOLOGIES CORP. [online]. [Accessed 5 Feb 2024]. Available from World Wide Web: <https://www.zebra.com/de/de/products/spec-sheets/mobile-computers/handheld/tc22-tc27.html>

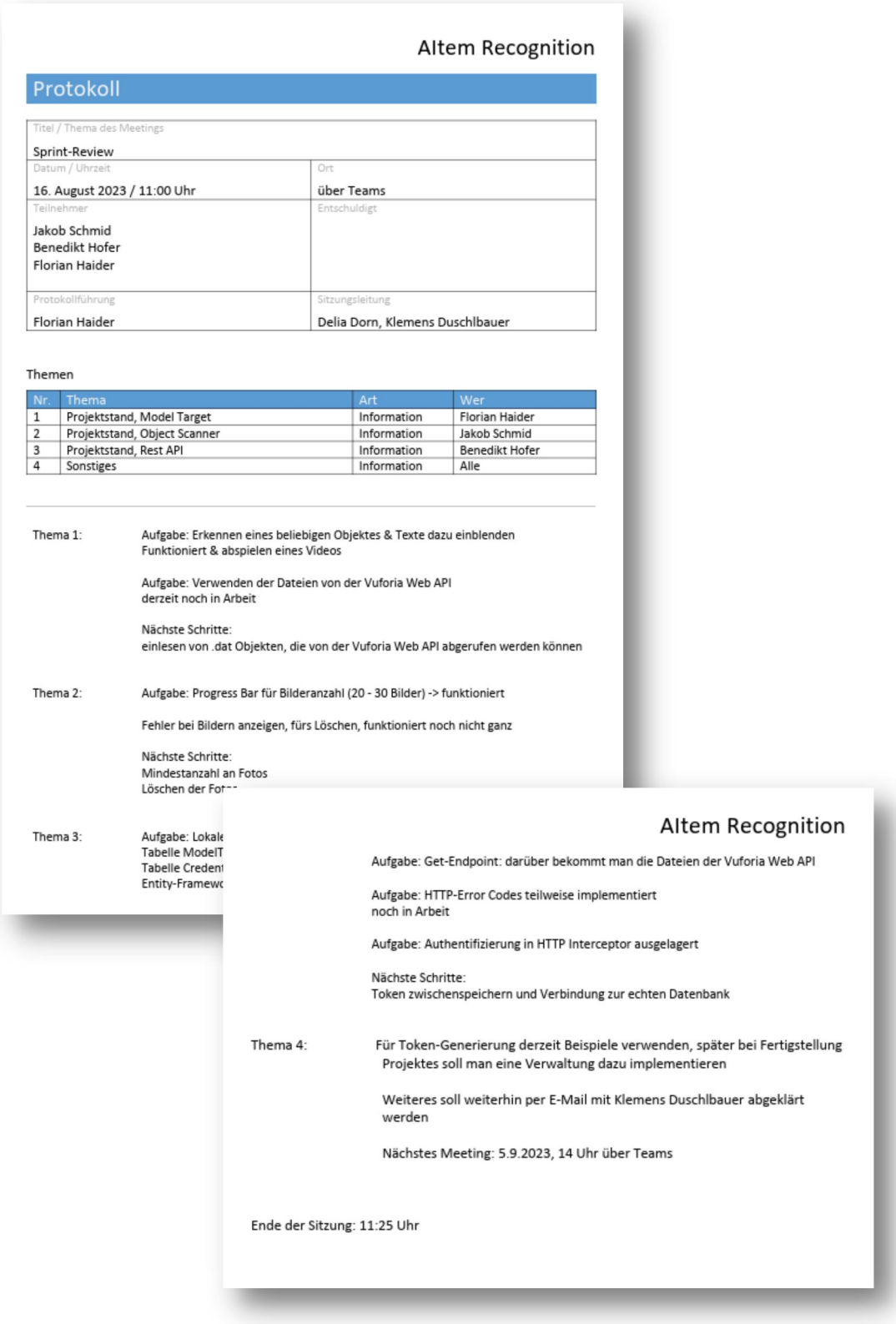
16. Abbildungsverzeichnis

17. Glossar

Begriff	Definition / Erklärung
Framework	Vorgefertigte Struktur, die Entwickler eine effiziente Grundlage für die Softwareentwicklung bietet
UUID	Universally Unique Identifier, ist eine Zeichenfolge, die verwendet wird, um Einträge eindeutig zu identifizieren
Thread	Ausführbare Einheit, die es ermöglicht mehrere Aufgaben parallel auszuführen
Request	Ein Request ist eine Anfrage. In unserem Kontext versteht man hier eine HTTP-Anfrage an eine REST-API.
Swagger	Tool zum automatischen Generieren von API-Dokumentation passend zu den definierten Endpoints. Swagger UI stellt eine Benutzeroberfläche zum Ansehen und Aufrufen der Endpoints zur Verfügung.
Model Target	Ein trainiertes 3D-Modell von Vuforia, dass zum Erkennen eines 3D-Modell verwendet, wird
OpenAPI Generator	Ein Code-Generator, der auf der OpenAPI Spezifikation basiert und Code für viele verschiedene Programmiersprachen zum Zugriff auf APIs generieren kann.
Frontend	Die Benutzeroberfläche, mit der Benutzer interagieren, um auf die Funktionen der Software zuzugreifen und sie zu nutzen.
Backend	Die nicht sichtbare, aber essenzielle Komponente der Software, die die Datenverarbeitung, Logik und Speicherung im Hintergrund steuert.
UI / User Interface	Benutzeroberfläche einer Software über die der Benutzer mit der Anwendung interagiert

18. Anhang

18.1. Protokoll



Altem Recognition

Protokoll

Titel / Thema des Meetings	
Sprint-Review	
Datum / Uhrzeit	Ort
16. August 2023 / 11:00 Uhr	über Teams
Teilnehmer	Entschuldigt
Jakob Schmid Benedikt Hofer Florian Haider	
Protokollführung	Sitzungsleitung
Florian Haider	Delia Dorn, Klemens Duschlbauer

Themen

Nr.	Thema	Art	Wer
1	Projektstand, Model Target	Information	Florian Haider
2	Projektstand, Object Scanner	Information	Jakob Schmid
3	Projektstand, Rest API	Information	Benedikt Hofer
4	Sonstiges	Information	Alle

Thema 1:

Aufgabe: Erkennen eines beliebigen Objektes & Texte dazu einblenden
Funktioniert & abspielen eines Videos

Aufgabe: Verwenden der Dateien von der Vuforia Web API
derzeit noch in Arbeit

Nächste Schritte:
einlesen von .dat Objekten, die von der Vuforia Web API abgerufen werden können

Thema 2:

Aufgabe: Progress Bar für Bilderanzahl (20 - 30 Bilder) -> funktioniert

Fehler bei Bildern anzeigen, fürs Löschen, funktioniert noch nicht ganz

Nächste Schritte:
Mindestanzahl an Fotos
Löschen der Fot

Thema 3:

Aufgabe: Lokale
Tabelle ModelT
Tabelle Credent
Entity-Framewo

Altem Recognition

Aufgabe: Get-Endpoint: darüber bekommt man die Dateien der Vuforia Web API

Aufgabe: HTTP-Error Codes teilweise implementiert
noch in Arbeit

Aufgabe: Authentifizierung in HTTP Interceptor ausgelagert

Nächste Schritte:
Token zwischenspeichern und Verbindung zur echten Datenbank

Thema 4:

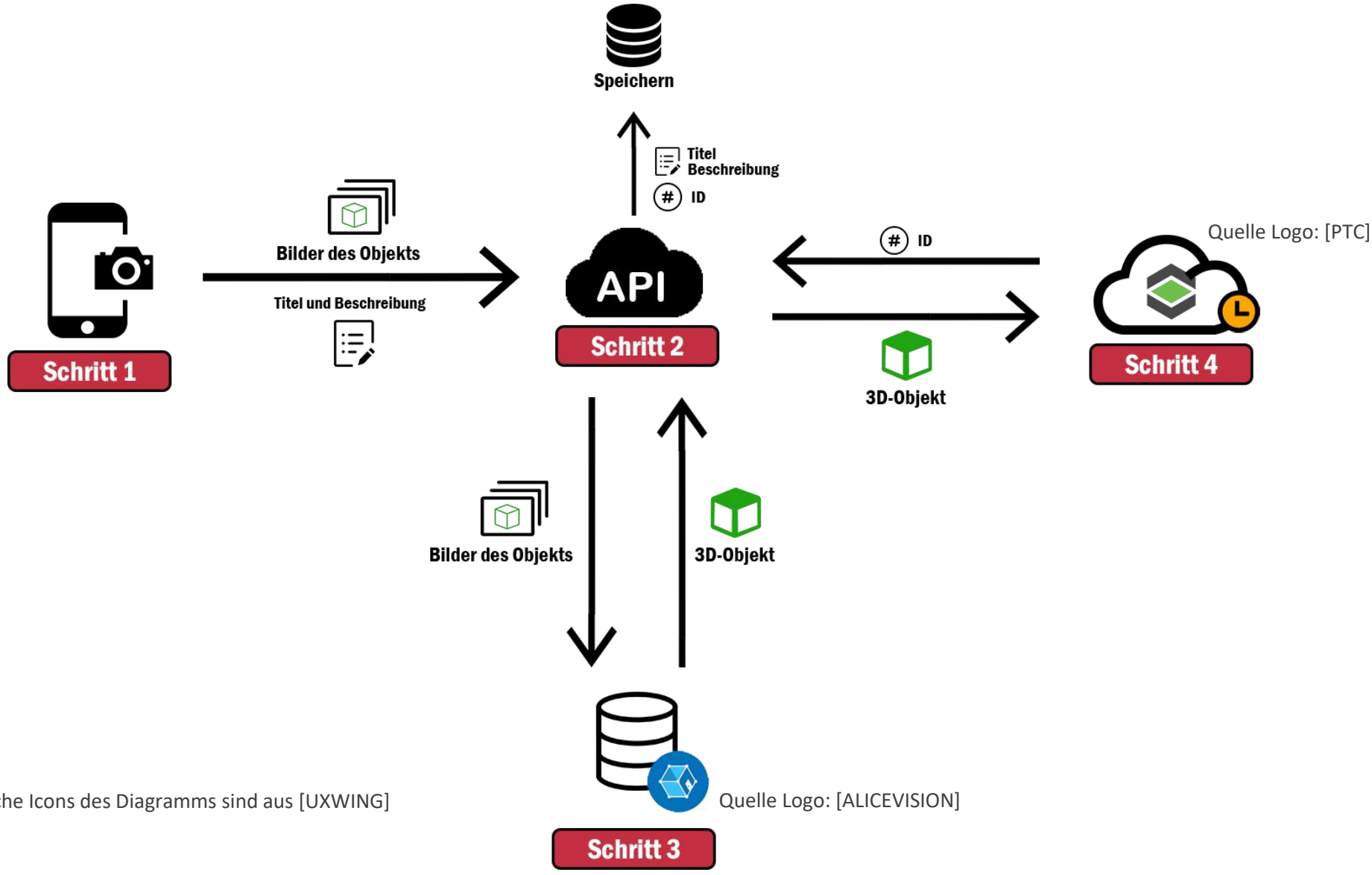
Für Token-Generierung derzeit Beispiele verwenden, später bei Fertigstellung Projektes soll man eine Verwaltung dazu implementieren

Weiteres soll weiterhin per E-Mail mit Klemens Duschlbauer abgeklärt werden

Nächstes Meeting: 5.9.2023, 14 Uhr über Teams

Ende der Sitzung: 11:25 Uhr

18.2. Objekt Scannen Diagramm

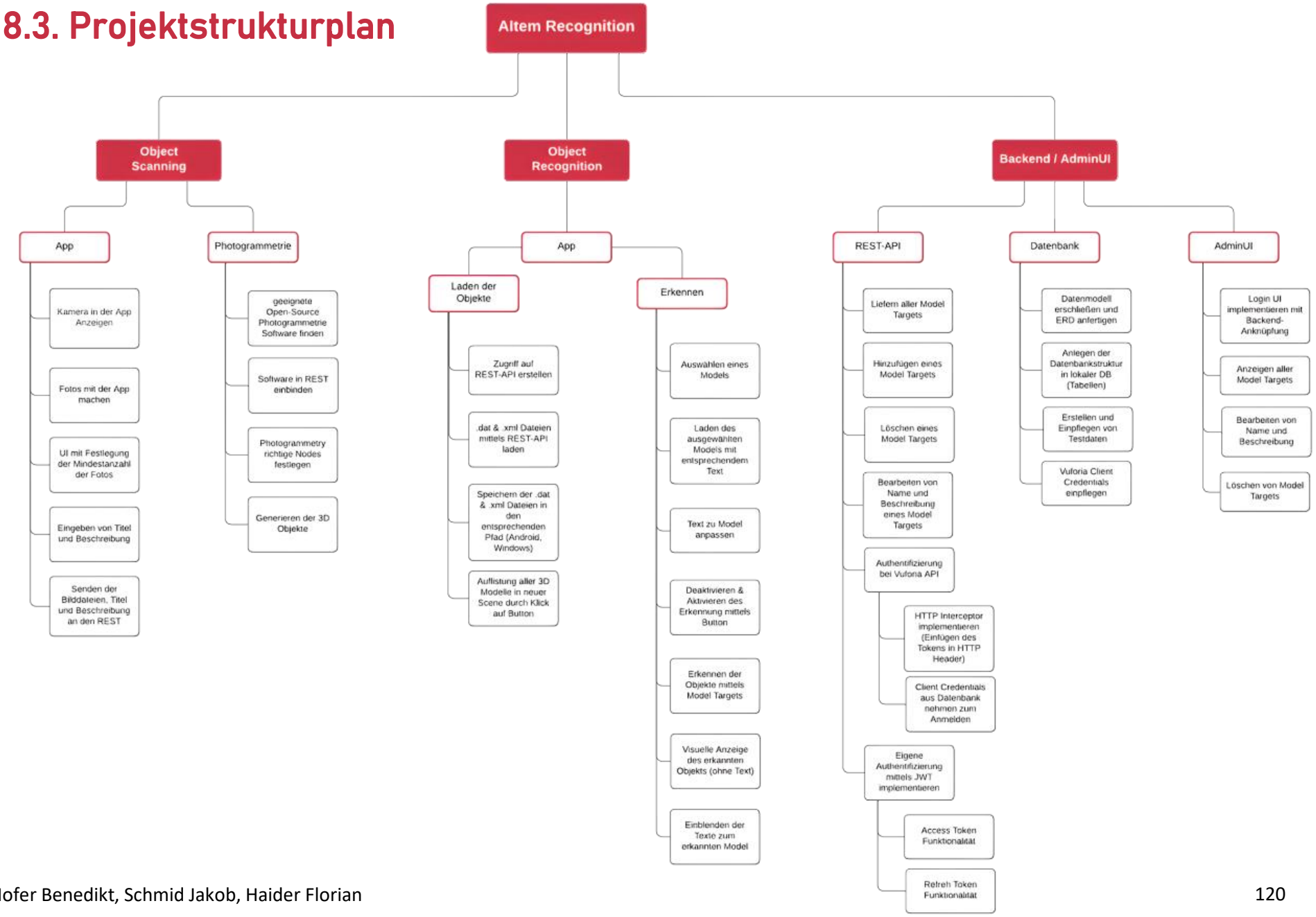


Quelle Logo: [PTC]

sämtliche Icons des Diagramms sind aus [UXWING]

Quelle Logo: [ALICEVISION]

18.3. Projektstrukturplan



18.4. SWOT-Analyse

INTERNE FAKTOREN	
STÄRKEN (+)	**SCHWÄCHEN (-)**
+ **Innovative Technologie** + **Flexibilität und Anpassungsfähigkeit** + **Informationen Management** relevante Informationen zu Maschinen o.ä. sind dadurch zentralisiert und leicht zugänglich	+ **Technische Herausforderungen** + **Benutzerakzeptanz** Akzeptanz der App durch Techniker und Monteur:innen könnte Herausforderung sein. Benutzeroberfläche muss daher intuitiv gestaltet werden.
EXTERNE FAKTOREN	
CHANCEN (+)	**BEDROHUNGEN (-)**
+ **Marktnachfrage** + **Wachsende Augmented Reality Branche** + **Internationale Firmen müssen immer mehr sprachliche Barrieren überbrücken**	+ **Datenschutz und Sicherheitsbedenken** + **Wettbewerb** mögliche Konkurrenz in AR- / Technologiebranche

18.5. Altem Recognition | Anmerkungen

Pfade für das Erkennen eines Model Targets

Speicherung in der „ConfigUrl.cs“.

```
localDataset = "https://localhost:7194/api/model-targets?includeDatasets=true";

localLogin = "https://localhost:7194/api/Authentication/login";

azureLogin = "https://project-ar.azurewebsites.net/api/auth/login";

azureDataset = "https://project-ar.azurewebsites.net/api/model
targets?includeDatasets=true";
```

Pfade für das Erkennen eines Model Targets

Pfade, für die Speicherung der ZIP-, Dat- und XML- Dateien werden in der „ConfigPath.cs“ statisch gespeichert.

public static string getPathAndroidVuforia()
`/storage/emulated/<userid>/Android/data/<packagename>/files`

public static string getPathAndroidZIP()
`\storage\emulated\<userid>\Android\data\<packagename>\files\targets.zip`

public static string getPathWindowsZIP()
`%userprofile%\AppData\LocalLow\<companyname>\<productname>\targets.zip`

public static string getPathWindowsVuforia()
`%userprofile%\AppData\LocalLow\<companyname>\<productname>`

Diese Methoden werden in der ControllerTarget.cs und in der ModelView.cs verwendet, um den base64String zu den Dat & XML-Dateien zu konvertieren.

ModelTarget erstellen und anzeigen

Alles befindet sich dazu im „Observer.cs“.

Ein ModelTarget besteht aus einer „Database“ und dem TargetName. Als Attribut wird der Database Name gesetzt. Der Default-Wert ist „MTDataset.xml“, weil aus jedem ModelTarget ein MTDataset.xml generiert wird.
Die Navigator.cs speichert den TargetName und die Beschreibung des 3D-Modelles. Mittels folgendem Code kann ein Modeltarget generiert werden.

```
modelTarget = VuforiaBehaviour.Instance.ObserverFactory.CreateModelTarget(
     //"storage/emulated/0/Android/data/com.project.projectar/pram-shadow-
files/assets/Vuforia/MTDataset.xml",
     path,
     targetName);
```

Möchte man ein selbst erstelltes Model Target statisch ausprobieren, so kann der Pfad und der targetName in dieser Datei angegeben werden.

Die Beschreibung wird in einem Textfeld angezeigt, dass in einem Prefab gespeichert wird. Dies kann folgendermaßen an das ModelTarget generiert werden.

```
if(status.Status != Status.NO_POSE)
 {
     mMyModelObject = Instantiate(myModelPrefab, modelTarget.transform);
     mMyModelObject.transform.localScale = new Vector3(0.025f, 0.025f, 0.025f);
     mMyModelObject.transform.localPosition = new Vector3(0, 2f, 0);
     /*
     mMyModelObject.transform.parent = modelTarget.transform;
     mMyModelObject.transform.localPosition = new Vector3(0, 0, 0);
     mMyModelObject.transform.localRotation = Quaternion.identity;
     mMyModelObject.transform.localScale = new Vector3(0.05f, 0.05f, 0.05f);
     mMyModelObject.transform.gameObject.SetActive(true);*/

     Debug.Log("[TS] scale " + modelTarget.transform.localScale.ToString());
     //mMyModelObject.transform.localScale = new Vector3(0.1f, 0.1f, 0.1f);
     Debug.Log($"[TS] instantiated");

     //mMyModelObject.transform.pos = new Vector3(0.1f, 0.1f, 0.1f);
     mMyModelObject.SetActive(true);
     Debug.Log($"[TS] set active");

 }
```

18.6. Kooperationsangebot

Höhere Technische Bundeslehranstalt Perg
Machlandstraße 48, 4320 Perg
E-Mail office@htl-perg.ac.at | Web www.htl-perg.ac.at
Tel 07262 53926 | Fax 07262 53926-6

Kooperationsangebot an die HTL-Perg

☒ Höhere Abteilung für Informatik ☐ Fachschule für Informationstechnik

Unternehmen:	ITPRO - Consulting & Software GmbH
Straße:	Buchnerplatz 1
PLZ, Ort, Staat:	4020 Linz
Ansprechperson:	Delia Dorn
Position:	Human Resources & Marketing
Telefon:	+43 732 615141 23
FAX:	
E-Mail:	d.dorn@itpro.at
Web-Adresse:	https://www.itpro.at

Gewünschte Kooperation (Mehrfachnennungen möglich):

☐ Ferialpraktikum
☐ Schulprojekt
☒ Diplomarbeit
☐ Berufspraktikum (Fachschule)
☐ Abschlussarbeit (Fachschule)

Betroffene SchülerInnen (falls bekannt) Benedikt Hofer, Jakob Schmid, Florian Haider

Titel/Thema der Kooperationsidee:

Altem Recognition
for mounting services

Kurze Beschreibung der Kooperationsidee:

Die Diplomarbeit soll das Schulprojekt „Project AR" dahingehend erweitern, dass der Monteur unter Zuhilfenahme der Technologie „Vuforia Object Recognition" für Unity auch ohne Hilfe eines Technikers Informationen zu den durchzuführenden Arbeitsschritten erlangen kann. Diese Funktion soll ebenfalls über das Hauptmenü ansteuerbar sein und wird getrennt vom anderen Modus (Live-Kommunikation zwischen Monteur und Techniker) behandelt. In diesem sieht man vorerst wiederum die Kameraaufnahme des Gerätes am Bildschirm. Sollten nun Maschinenteile via „Vuforia Object Recognition" erkannt werden, so werden die entsprechenden Objekte optisch hervorgehoben und durch einen Tippen auf diese wird der dazugehörige Informationstext eingeblendet. Dabei kann es sich um eine Beschreibung des Teiles oder aber auch um eine direkte Arbeitsanweisung handeln. Diese Funktionalität soll dem Monteur Hilfe bieten, falls momentan kein Techniker erreichbar ist.

Frühest möglicher Beginn:	Juli 2023
Spätest mögliches Ende:	April 2024

16.03.2023
Datum

IT PRO Consulting & Software GmbH Buchnerplatz 1 | A-4020 Linz T: +43 (732) 615141-0 | Fax: DW 51 E-Mail: office@itpro.at www.itpro.at
Unterschrift

Bitte senden Sie dieses Formular an die HTL-Perg, Machlandstraße 48, A-4320 Perg, oder via E-Mail an office@htl-perg.ac.at.

18.7. Abnahmeprotokoll

htl perg	**HTBLA Perg** **Höhere Lehranstalt für Informatik**	**Reife- und Diplomprüfung**

Abnahmeprotokoll	**HTBLA Perg**
Projektname *Altem Recognition*	**Projektnummer :**
Auftraggeber ITPRO - Consulting & Software GmbH	**Empfänger :** Klemens Duschlbauer

Das Werk „Altem Recognition“ Version 1.0

basierend auf dem Kooperationsangebot vom 16.März 2023

wurden am 27. März 2024 zur Abnahme angemeldet.

Zu dem Werk gehören folgende **Komponenten**:

- ☑ Dieses Protokoll
- ☑ Die ProjectAR Dokumentation/Anleitung
- ☑ Dockerfile und Code zur Erstellung des Generators
- ☑ Source Code des RestAPI
- ☑ Source Code des AdminUI
- ☑ APK der Application inkl. Source Code

Mit seiner Unterschrift bestätigt der Auftraggeber die Abnahme des Werkes.

Datum: 27.03.2024

IT|PRO Software die's einfach macht
Buchnerplatz 1 | A-4020 Linz
T: +43 (732) 615141-0
E-Mail: office@itpro.at
www.itpro.at

Unterschrift
Auftraggeber

Unterschrift
Auftragnehmer

Altem-Recognition_Abnahmeprotokoll.doc Seite 1 von 1