

Master Thesis

in der Fachrichtung
Informatik

Thema:

**Indoor Navigation mit Machine Learning und
Augmented Reality
am Beispiel der FH Wedel**

Eingereicht von: Mara Pape

Erarbeitet im: 5. Semester

Abgegeben am: 31. August 2022

Referent:
Prof. Dr. Ulrich Hoffmann
Fachhochschule Wedel
Feldstraße 143
22880 Wedel
Tel.: +49 (0)4103 /8048-41
E-Mail: ulrich.hoffmann@fh-wedel.de

Zweitkorrektor:
Prof. Dr. Dennis Säring
Fachhochschule Wedel
Feldstraße 143
22880 Wedel
Tel.: +49 (0)4103 /8048-43
E-Mail: dennis.saering@fh-wedel.de

Indoor Navigation mit Machine Learning und Augmented Reality- am Beispiel der FH Wedel

2022, Master Thesis von Mara Pape, Fachhochschule Wedel

Zusammenfassung:

In dieser Arbeit wurde ein Indoor Navigationssystem für die Fachhochschule Wedel entwickelt, welches maschinelles Lernen für die Standortermittlung und Augmented Reality für die Nutzerführung verwendet. Die Augmented Reality-Komponente selbst verwendet ebenfalls maschinelles Lernen. Die Anwendung bietet eine Benutzeroberfläche, die intuitiv zu bedienen ist und Anfragen an einen Server initiiert, der die entsprechenden Berechnungen durchführt.

Inhaltsverzeichnis

Abbildungsverzeichnis	VII
Listings	VIII
Abkürzungsverzeichnis	VIII
1 Einleitung	1
1.1 Motivation	1
1.2 Ziel	3
1.3 Überblick über die Arbeit	3
2 Grundlagen	4
2.1 Indoor Standortermittlung	4
2.2 Machine Learning	8
2.2.1 Neuronale Netze	10
2.2.2 Neuronale Netze für Bildverarbeitung	11
2.2.3 Transfer Learning	13
2.3 Wegfindung	14
2.4 Augmented Reality	15
2.5 Bildverarbeitung	18
2.6 Zusammenfassung	20
3 Indoor Navigation	21
3.1 Grundsätzliche Funktionsweise	21
3.2 Standortermittlung	22
3.3 Wegfindung	23
3.4 Anforderungen an die Indoor Navigation	24
3.4.1 Funktionale Anforderungen	24
3.4.2 Nicht funktionale Anforderungen	25
3.5 Verwandte Arbeiten	27
3.5.1 Nutzung von Computer Vision	27
3.5.2 Nutzung von ML ohne Computer Vision	28
3.5.3 Hybride Ansätze	28
3.6 Zusammenfassung	29

Inhaltsverzeichnis

4 Augmented Reality	30
4.1 Grundsätzliche Funktionsweise	30
4.2 Anforderungen an die Augmented Reality	33
4.2.1 Funktionale Anforderungen	33
4.2.2 Nicht funktionale Anforderungen	33
4.3 Verwandte Arbeiten	34
4.3.1 Markerlose AR	34
4.3.2 Markerbasierte AR	35
4.4 Zusammenfassung	35
5 Realisierung	36
5.1 Verwendete Technologien	36
5.2 Architektur	38
5.3 Interne Darstellung der FH	39
5.4 Kommunikationsfluss	42
5.5 Indoor Navigation	48
5.5.1 Softwaredesign	48
5.5.2 Daten und Datenerhebung	53
5.5.3 Training	57
5.5.4 Architektur der Modelle	65
5.5.5 Wegfindung	68
5.6 Augmented Reality	70
5.6.1 SoftwareDesign	70
5.6.2 AR-Implementationen	73
5.6.3 Vergleich der Implementationen	85
5.7 Zusammenfassung und Fazit	85
6 Ergebnis	87
6.1 Starten und Nutzung der Anwendung	87
6.2 Anforderungen Indoor Navigation	88
6.3 Anforderungen Augmented Reality	94
6.4 Demo-Video mit Standorterkennung und AR	97
6.5 Erkenntnisse	98

Inhaltsverzeichnis

7 Zusammenfassung und Ausblick	99
7.1 Zusammenfassung	99
7.2 Verbesserungsmöglichkeiten	100
7.3 Weitere Funktionalitäten	101
Literaturverzeichnis	103
Anhang	109
Eidesstattliche Erklärung	113

Abbildungsverzeichnis

2.1	TOA-basierte Positionsbestimmung	6
2.2	Ablauf von überwachtem Lernen	8
2.3	Funktionsweise Convolution	11
2.4	Funktionsweise Pooling	12
2.5	Verschiedene AR-Applikationen	16
2.6	Markerbasierte AR	17
2.7	Kernel für die Bildverarbeitung	18
3.1	Ablauf Indoor Navigation-Teilanwendung	22
3.2	Rundgang im Erdgeschoss der FH Wedel	23
4.1	Indoor Navigation mit Augmented Reality	31
4.2	Ablauf der AR-Teilanwendung	32
5.1	Grobe Architektur der Anwendung	38
5.2	Lageplan der FH Wedel	40
5.3	Ausschnitt aus der internen Darstellung der FH Wedel	41
5.4	Kommunikationsfluss	45
5.5	Klassendiagramm Indoor Navigation	49
5.6	Positionsvalidierung	50
5.7	Bildausschnitte aus dem Gang A1, Position 5, Blickrichtung 4	56
5.8	Gang D0 mit verschiedenen Wetterkonditionen	57
5.9	Architektur der Modelle	66
5.10	Diagramm der Zusammenhänge der Python-Files	71
5.11	Objekterkennung mit Bildverarbeitungsmaßnahmen	74
5.12	Berechnung der Markerpositionen auf dem Boden	82
5.13	Ermittlung des richtigen Intervalls	83
5.14	Berechnung der Markerposition innerhalb des Intervalls	84
6.1	Aufnahmen der laufenden Applikation	88
6.2	Demo-Video der Segmentierung	95
7.1	Gerasterter vollständiger Lageplan	112

Listings

5.1	Adjazenzliste des Gebäudeteils E im 1. Stock	42
5.2	Route der initialen Navigation	43
5.3	Route der regulären Navigation	43
5.4	Route der Augmented Reality-Anfrage	44
5.5	Anfrage der Initialen Navigation vom Client	46
5.6	Antwort der Initialen Navigation vom Server	46
5.7	Antwort der AR-Anfrage vom Server	47
5.8	Einzelbilderzeugung	54
5.9	Automatisches Labeling der Trainings- und Testdaten	58
5.10	Transfer Learning	64
5.11	Form der Layer	67
5.12	Implementation der Breitensuche	69
6.1	Befehle zum Starten des Backends	87
7.1	package.json	109
7.2	environment.yml	110

Abkürzungen

AR Augmented Reality.

CNN Convolutional Neural Network.

CPU Central Processing Unit.

FPS Frames per Second.

GPU Graphics Processing Unit.

JSON Javascript Object Notation.

ML Machine Learning.

PDR Pedestrian Dead Reckoning.

RSS Received Signal Strength.

TOA Time of Arrival.

1

Einleitung

1.1 Motivation

Der Wunsch nach Mobilität der Menschen in Deutschland ist spätestens mit der Diskussion um das 9 Euro-Ticket wieder präsent [Gmb]. Doch dieser Wunsch ist nicht nur auf Menschen in Deutschland beschränkt. Die meisten Menschen besuchen im Laufe ihres Lebens verschiedene Orte. Der Weg dorthin kann beispielsweise zu Fuß oder mit Verkehrsmitteln geschehen, wie etwa mit dem Fahrrad oder mit einem gemieteten E-Roller. Für weitere Strecken kommen zum Beispiel der eigene PKW, ein PKW eines Car-Sharing-Betriebs oder andere öffentliche Verkehrsmittel wie Bus oder Bahn in Frage. Doch nicht nur Personen müssen an verschiedene Orte gelangen können, sondern auch Güter. Eine Frage, die sich in allen Szenarien stellt, ist die Frage nach der Route zum Ziel. Das Feld der automatischen Navigation kann diese Frage beantworten.

Navigation besteht aus der Standortermittlung, Routenplanung und dem Führen des Users durch ebendiese Route. Sie hilft also Menschen, sich in einer für sie fremden Umgebung zurechtzufinden. In der Vergangenheit war ein dediziertes Navigationsgerät dafür notwendig, etwa fest integriert in ein Auto. Inzwischen ist sie aber auch auf dem Handy verfügbar, zum Beispiel durch Anbieter wie Google Maps. Das Navigationsgerät kann also auch mobil und mit Akku verwendet werden. Daher können auch etwa FußgängerInnen oder andere VerkehrsteilnehmerInnen die Navigation per App benutzen. Die verwendete Technik zur Standortbestimmung basiert meist auf Satellitensignalen (GPS - Global Positioning System). Innerhalb eines Gebäudes werden diese Signale allerdings durch das Gebäude selbst abgeschirmt [HK10], sodass darüber in Innenräumen keine Standortbestimmung möglich ist.

1 Einleitung

Als GästIn ist jedoch auch das Finden eines bestimmten Raumes in einem fremden Gebäudekomplex nicht immer einfach. Lagepläne können ungenau und die Beschilderung unzureichend sein. Zudem ist räumliches Denken erforderlich, um einen Lageplan korrekt zu lesen und der direkte Weg ist möglicherweise nicht immer barrierefrei beziehungsweise das Ziel möglicherweise gar nicht barrierefrei erreichbar. Auch Menschen mit eingeschränktem Sehvermögen können von Navigationssystemen profitieren.

Durch die Abschirmung der Satellitensignale müssen jedoch andere Verfahren herangezogen werden, denn auch eine Verbesserung des GPS-Systems, welches sich High-Sensitivity GPS nennt, ist innerhalb eines Gebäudes nicht genau genug [HK10]. Es gibt dennoch verschiedenste Möglichkeiten, präzise die Position in Innenräumen zu bestimmen und nachzuverfolgen, wie beispielsweise mit einer Standortermittlung und -Verfolgung basierend auf WLAN- oder Bluetooth-Signalen. Daneben existieren Verfahren, die die Bewegung von NutzerInnen anhand von Sensoren schätzen oder Computer Vision nutzen [NAHF20].

Durch Angabe eines Zielorts können NutzerInnen diesen auch ohne Kenntnis des Aufbaus des Gebäudes einfach erreichen, indem zuerst der Standort ermittelt und anschließend eine Route zum Ziel berechnet wird. Auch in der Fachhochschule Wedel ist das sinnvoll, da sich jedes Semester neue Studierende zurechtfinden müssen und sie beispielsweise für die Firmenkontaktmesse regelmäßig GästInnen empfängt, die ebenfalls nicht mit dem Aufbau des Gebäudes vertraut sind.

Nachdem eine Route berechnet wurde, müssen die berechneten Informationen dem/-der NutzerIn aufbereitet werden. Dies kann über verschiedene Wege geschehen, zum Beispiel über eine Audiodeskription für Personen mit eingeschränktem Sehvermögen, textuell oder mittels Symbolen. Texte oder Symbole können mit Augmented Reality die Realität überlagern. Somit wird AnwenderInnen die Handhabung erleichtert, denn die Routeninformation muss nicht erst von einer zweidimensionalen oder textuellen Darstellung in die Realität übertragen werden, sondern überlagert diese. Vorausgesetzt ist ein uneingeschränktes Sehvermögen des Users.

1.2 Ziel

Ziel dieser Arbeit ist ein Indoor-Navigationssystem über eine Web-Applikation, die intuitiv mit dem Smartphone bedient werden kann. Diese soll eine einfache Navigation innerhalb des Hauptgebäudes der Fachhochschule Wedel ermöglichen. Die Anwendung besteht aus Frontend und Backend. Das Frontend übernimmt die Aufnahme eines oder mehrerer Bilder und sendet sie an das Backend zur Standortermittlung und zur Berechnung des Pfades. Außerdem rendert das Frontend Symbole, insbesondere Pfeile, auf das Kamerabild. Die Positionsinformation des jeweiligen Symbols stammt aus dem Backend.

Der Standort, der für die Navigation notwendig ist, wird mit Hilfe von maschinellem Lernen bestimmt. Navigationsinformationen sollen zum einen über Augmented Reality als auch in Schriftform an den/die NutzerIn gegeben werden. Dabei soll es eine zusätzliche Validierung der ermittelten Position anhand der vorher ermittelten Position geben. Die initiale Position wird über den Median mehrerer Bilder gebildet.

1.3 Überblick über die Arbeit

Im Kapitel 2 werden Grundlagen behandelt, die für das Verständnis der folgenden Kapitel gegebenenfalls benötigt werden, vor allem im Bereich Machine Learning, speziell Computer Vision, und Augmented Reality. In Kapitel 3 und 4, welche die beiden Bestandteile des Indoornavigationssystems beschreiben, werden Anforderungen postuliert und diese Arbeit mit anderen verwandten Arbeiten verglichen. Anschließend wird in Kapitel 5 die Implementation beleuchtet und in Kapitel 6 das Ergebnis behandelt sowie die Erfüllung der Anforderungen geprüft. Das letzte Kapitel 7 beinhaltet eine Zusammenfassung und einen Ausblick, wie der bestehende Prototyp weiter verbessert und mit welchen Funktionalitäten erweitert werden kann.

2

Grundlagen

In diesem Kapitel werden zum besseren Verständnis einige grundlegende Sachverhalte erläutert. Dabei werden Themenbereiche wie Indoor Navigation, Machine Learning, Wegfindungsalgorithmen, Augmented Reality und Bildverarbeitung aufgegriffen.

2.1 Indoor Standortermittlung

Das Global Positioning System (GPS) ist ein Outdoor System und kann in Innenräumen nicht gut verwendet werden, da die Signale vom Gebäude selbst abgeschirmt werden [HK10]. Daher gibt es verschiedene Varianten, mit denen eine Navigation innerhalb eines Gebäudes trotzdem funktionieren kann. In diesem Abschnitt werden einige Varianten der Indoor-Standortermittlung aufgezeigt, die kein GPS nutzen. Die technischen Ansätze können unterteilt werden in:

1. Computer Vision
2. Kommunikationstechnologie
3. Pedestrian Dead Reckoning (PDR)

Computer Vision

Computer Vision benötigt ein Input Bild, welches beispielsweise mit einer omnidirektionalen Kamera (360° Sichtfeld in der Horizontalen), 3D-Kamera oder einer Handykamera aufgenommen wurde. Aus diesem Input kann über Bildverarbeitungsalgorithmen durch die Feature-Extraktion eine Position ermittelt werden [KKAmAA20, S. 2,3]. Eine Feature-Extraktion ermittelt die wichtigsten Merkmale eines Bildes.

Eine Art Computer Vision zu realisieren, ist mit maschinellem Lernen. Bei diesem Verfahren kann ein sogenanntes Model erkennen, welche Objekte auf einem Bild zu sehen sind und wo diese sich befinden. Dafür muss das Model mit Hilfe von großen Datenmengen trainiert werden. Auf eine spezielle Technologie mittels Machine Learning und deren Funktionsweise wird in Abschnitt 2.2 genauer eingegangen, die in dieser Arbeit verwendet wurde.

Computer Vision erfordert aber nicht zwangsläufig maschinelles Lernen. Andere oft verwendete Methoden sind SURF (Speeded Up Robust Features [BVG06]) oder SIFT (Scale Invariant Feature Transform [Lin12]). Diese können auch für Markererkennung, wie etwa für markerbasierte AR in Kapitel 2.4 notwendig, verwendet werden. Diese Methoden bestehen aus zwei Aufgaben. Zum einen müssen sogenannte Interest Points ermittelt werden, zum anderen werden anhand dieser sogenannte Feature-Deskriptoren gebildet. Die Interest Points sind Wiedererkennungsmerkmale in einem Bild und finden sich zum Beispiel an Kanten oder anderen markanten Stellen. Ein Interest Point wird anschließend angereichert mit Informationen, welche die Nachbarregionen des Bildes beschreiben. Diese Beschreibung der den Interest Point umgebenden Pixeln erfolgt durch einen Deskriptor als Feature-Vektor, der gleiche Objekte unabhängig der Größe und perspektivischen Verzerrungen erkennt, das heißt, der Feature-Vektor eines Objekts ist ähnlich, unabhängig der Kameraposition und des Aufnahmewinkels. Die Ähnlichkeit zweier Vektoren wird oft anhand der euklidischen Distanz ermittelt. Somit kann mit diesem Algorithmen Objekterkennung durchgeführt werden.

Kommunikationstechnologie

Bei den kommunikationstechnologischen Ansätzen können verschiedene Signale verwendet werden, zum Beispiel WLAN, Bluetooth, Ultraschall oder Infrarot. Aus diesen können Eigenschaften extrahiert werden, wie etwa aus welcher Richtung das Signal kommt (AoA - Angle of Arrival), wie stark das Signal ist (RSS - Received Signal Strength) oder an welcher Position das Signal aufgenommen wurde (CoO - Cell of Origin)[HG10].

Die technologiespezifischen Signale werden von den Emittoren ausgesendet, welche der Receiver empfängt. Der Receiver kann ein Mobiltelefon sein, aber auch ein eigens dafür konzipiertes Gerät [HK10].

Je nach Reichweite und Anzahl der Emitter, werden verschieden viele Signale empfangen. Diese werden von Positionierungsalgorithmen verwendet, um die Position des/der NutzerIn zu bestimmen [KKAmAA20]¹. Die verbreitetsten Methoden sind zeitbasiert, winkelbasiert oder RSS-basiert.

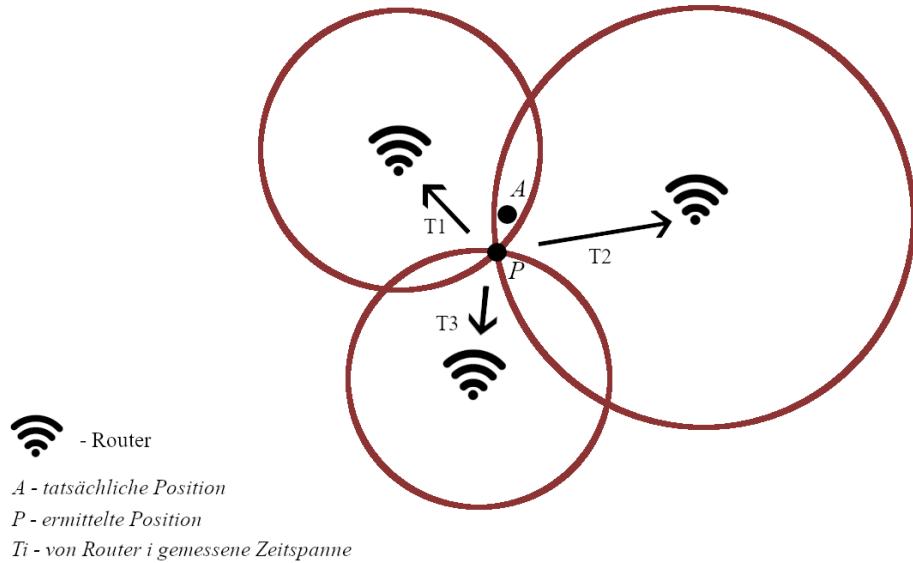


Abbildung 2.1: TOA-basierte Positionsbestimmung

Zeitbasierte Methoden nutzen beispielsweise die Eigenschaft Time of Arrival (TOA). TOA wertet die Zeitspanne aus, die ein Signal braucht, um von Emitter zu Receiver zu gelangen. Wie in Abbildung 2.1 zu sehen, werden verschiedene Zeitspannen mit dem Receiver ermittelt, in diesem Beispiel durch drei WLAN-Emitter. Mit anderen Emittoren funktioniert dieser Ansatz genauso. Durch die Zeitspanne kann die Entfernung zum Emitter ermittelt werden. Dafür muss die Geschwindigkeit des Signals bekannt sein. Die roten Kreise um die Emittoren stellen die mögliche Position des Receivers bezogen auf einen Emittoren dar. Die Position kann durch einen Schnittpunkt eindeutig bestimmt werden, wenn mehrere Signale empfangen werden, wie der Punkt P in Abbildung 2.1 zeigt. Dieser nähert den Punkt A gut an. Wird nur ein Signal empfangen, kommen mehrere Positionen in Frage. Einige davon können jedoch ausgeschlossen werden, da dort durch andere Emittoren ebenfalls Signale empfangen werden müssten. Diese Ansätze sind allerdings so konzipiert, dass die Signale über den direkten Weg von Emittoren zu Receiver kommen. Sind Wände oder andere Hindernisse auf dem Weg, braucht das Signal länger, womit eine Ungenauigkeit der Standortlokalisierung einhergeht [Kee17].

¹Der weitere Abschnitt beruht, wenn nicht anders angegeben, auf [KKAmAA20].

RSS-basierte Methoden können auf einem ähnlichen Ansatz beruhen wie TOA. Anhand der empfangenen Signalstärke kann die Entfernung zu den Emittoren ermittelt und die Berührungs punkte der Kreise um die Emitter berechnet werden. Das Verfahren, mit der die Position bestimmt wird, nennt sich Trilateration.

Daneben gibt es sogenannte Fingerprinting Methoden, welche auf verschiedenen Signaleigenschaften wie zum Beispiel RSS beruhen. Dieser Ansatz nutzt eine Datenbank, welche den einzelnen Positionen empfangene Signale beziehungsweise Signalstärken zuordnet. In der sogenannten Offline-Phase werden an verschiedenen Positionen die Signale gemessen und mit der Position zusammen in die Datenbank eingepflegt. Die Nutzung der Standortermittlung durch dieses Verfahren wird als Online-Phase bezeichnet. Die durch den Receiver empfangenen Signale werden per Pattern Matching mit den Daten in der Datenbank verglichen. Das ähnlichste Datenelement, also zum Beispiel eine Zeile in einer relationalen Datenbank, ist die geschätzte Position des Receivers. Pattern Matching in der Datenbank kann mit verschiedenen Algorithmen gelöst werden, wie zum Beispiel der euklidischen Distanz oder Algorithmen aus dem Feld von maschinellem Lernen, wie etwa Support Vector Machines oder K-Nearest Neighbor.

Pedestrian Dead Reckoning

Der Pedestrian Dead Reckoning-Ansatz (PDR)² nutzt Sensoren innerhalb eines Gerätes, etwa eines Smartphones, um anhand der letzten bekannten Position und den Sensordaten eine neue Position zu ermitteln. Dabei werden keine weiteren Emitter und Receiver benötigt, wie etwa bei den kommunikationstechnologischen Methoden zur Positionsbestimmung.

Für PDR wird zum Beispiel ein Beschleunigungssensor benötigt, der die Beschleunigung in allen drei Achsen misst. Die Schrittlänge, die von NutzerIn zu NutzerIn variiert, kann mit Hilfe der maximalen und minimalen Beschleunigung sowie einem nutzerspezifischen Parameter berechnet werden. Durch die Beschleunigungssensoren kann auch gemessen werden, wann ein Schritt durchgeführt wurde und somit die zurückgelegte Strecke berechnet werden. Ein Gyroskop ist ebenfalls in Smartphones integriert und kann die Ausrichtung der Geräts messen. Mit diesen Informationen kann die Bewegung von NutzerInnen nachverfolgt werden. Dafür wird ein vorher bekannter Standort benötigt, von dem aus die Bewegung aufgezeichnet wird. Mit diesem Ansatz akkumulieren sich Rundungsfehler, das heißt, es sollte eine Möglichkeit der Rekalibrierung geben, zum

²Dieser Abschnitt beruht, wenn nicht anders angegeben, auf [KKAmAA20].

Beispiel durch bestimmte Marker innerhalb des Gebäudes, von denen die jeweiligen Positionen bekannt sind. Außerdem können NutzerInnen das entsprechende Navigationsgerät nicht etwa in einer Tasche verstauen, da die Sensoren sonst nicht die korrekten Daten aufnehmen, wenn das Gerät zum Beispiel mit der Vorderseite entgegen der Laufrichtung zeigt.

2.2 Machine Learning

Machine Learning (ML) oder maschinelles Lernen ist ein Teil des Gebiets der künstlichen Intelligenz (KI, engl. AI). Dabei lernt der Computer, genauer ein Machine Learning Model, eine Aufgabe anhand von Daten zu lösen, ohne dass das explizite Programmieren von Funktionen benötigt wird, welche die Daten als Parameter verwenden. Dabei gibt es verschiedene Arten von Algorithmen, solche Modelle zu trainieren, zum Beispiel mit überwachtem (engl. supervised) und unüberwachtem (engl. unsupervised) Lernen [ZLLS21, S. 22ff].

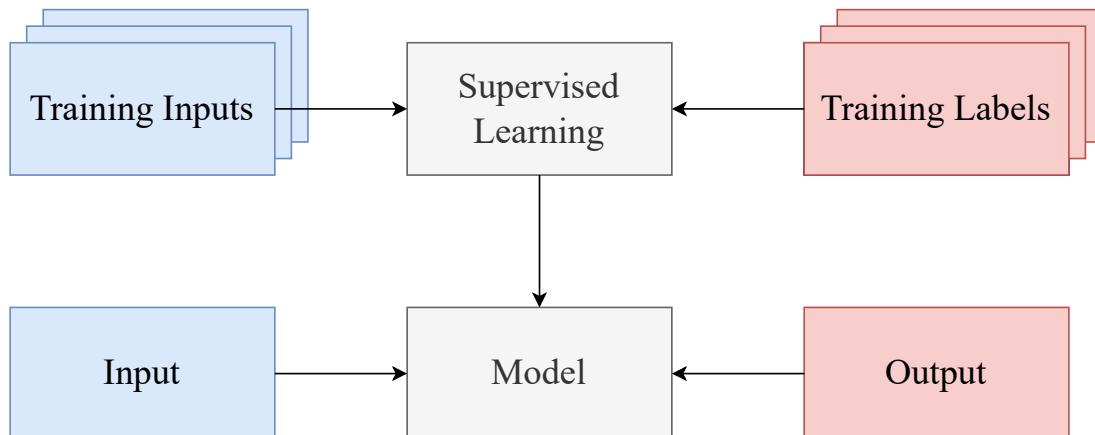


Abbildung 2.2: Ablauf von überwachtem Lernen, Bildquelle: [ZLLS21, S. 23]

Überwachtes Lernen erfordert zum einen die Trainingsdaten selbst, die in Abbildung 2.2 mit *Training Inputs* bezeichnet werden, aber auch die *Training Labels*, die das entsprechend richtige Ergebnis enthalten. Ein Model, das zum Beispiel eine Frucht auf einem Bild erkennen soll, erhält als Beispiel ein Bild mit einem Apfel und das zugehörige Label *Apfel* als Input für das Training. Das wird mit allen Beispielen, die den gesamten Trainingsdatensatz formen, wiederholt. Wenn der Trainingsprozess des überwachten Lernens beendet wurde, kann das Model auch andere Bilder, auf denen zum Beispiel ein Apfel zu sehen ist, richtig als Apfel klassifizieren, wie in der unteren Hälfte der Abbildung

2.2 zu sehen ist. Bei unüberwachten Lernmethoden gibt es dieses Label nicht und das Model muss die Daten eigenständig strukturieren.

Typische Aufgaben für ein Machine Learning-Model sind nach [ANK18]³:

1. **Klassifikation:** Bei Klassifikationsproblemen soll ein ML-Model erkennen, zu welcher Kategorie ein bestimmter Input gehört. Die Kategorien müssen im Voraus bekannt sein. Solche Kategorien können boolesche Werte oder andere problemspezifische Kategorien sein. Diese können zwei oder mehr Klassen enthalten. Ein Model kann zum Beispiel die Aufgabe haben, zu ermitteln, ob ein Bild als Input einen Hund, eine Katze oder eine Maus zeigt. Das Ergebnis einer Klassifikation ist in der Regel eine Menge an Wahrscheinlichkeiten für jeden möglichen Output. Es ist auch möglich, Models zu trainieren, die mehrere Kategorien zu einem Input bestimmen. Wenn auf einem Bild alle drei Tiere zu sehen sind, soll das Model die Kategorien *Hund*, *Katze* und *Maus* als Output zurückliefern [ZLLS21, S. 25].
2. **Anomalie-Erkennung:** Bei der Anomalie-Erkennung sollen Anomalien in bestimmten Mustern entdeckt werden. Das können zum Beispiel Bank-Transaktionen sein. Wird eine Anomalie bei einem Konto festgestellt, können KontoinhaberInnen alarmiert werden. Falls eine Person unrechtmäßigen Zugang bekommen konnte, kann das Konto gesperrt werden.
3. **Regression:** Diese Art Aufgabe behandelt kontinuierliche, also nicht diskrete Werte. Anhand der gelernten Daten wird mit neuen Daten eine Vorhersage getroffen, wie zum Beispiel bei einer Preisvorhersage für Immobilien.
4. **Clustern:** Diese Art Aufgabe fällt unter das unbeaufsichtigte Lernen, denn durch das Model werden Datenpunkte eigenständig einem Cluster zugeordnet. Die Daten werden automatisch strukturiert. Dabei ist im Vorhinein nicht bekannt, welche Punkte in welches Cluster fallen.

³Sofern nicht anders angegeben, ist die Quelle für diesen Absatz [ANK18]

5. **Reinforcement Learning**⁴: Hierbei werden zukünftige Entscheidungen eines Models durch die vergangenen Erfahrungen beeinflusst. Das zugrundeliegende Prinzip ist Trial and Error, das heißt, zunächst wird das Model zufällige Ergebnisse liefern. Je besser ein Ergebnis, desto besser ist die Bewertung des Ergebnisses. So passt sich das Model entsprechend dieser Bewertung an, um eine möglichst optimale Bewertung zu erreichen. Dabei muss die Aufgabe für das Model nicht spezifiziert werden, sondern nur eine Bewertungsfunktion, die einer Prediction des Models eine Bewertung zuordnet. Die Prediction ist das Ergebnis, welches dieses Model berechnet hat.

2.2.1 Neuronale Netze

Neuronale Netze sind Teil des maschinellen Lernens. Ein klassisches Neuronales Netz besteht aus vielen Neuronen, welche einen Input und einen Output haben. Die Neuronen sind in Layern über gewichtete Verbindungen miteinander verknüpft. Neuronen können ein anderes Neuron in dem nachfolgenden Layer aktivieren, wenn sie mit den entsprechenden Gewichtungen insgesamt einen Schwellwert überschreiten. Neuronen im Input Layer, dem ersten Layer der Netzes, haben keine Vorgänger und werden durch den Input direkt aktiviert. Neben dem Input Layer und dem Output Layer gibt es noch weitere Layer, welche sich zwischen diesen befinden, die Hidden Layer [Sch15, S. 4].

Das Training eines neuronalen Netzes besteht darin, die Gewichtungen so anzupassen, dass im letzten Layer, dem Output Layer, das korrekte Neuron oder die korrekten Neuronen aktiviert werden. Das Training sollte mit circa 80% des Datensatzes erfolgen, 20% sollten verwendet werden, um zu validieren, dass das Model nicht nur Daten erkennt, mit denen es trainiert wurde (Overfitting), sondern auch dem Model unbekannte Daten richtig ausgewertet werden[G17, S. 29].

⁴Reinforcement Learning ist in dem Sinne keine Problemstellung, die maschinelles Lernen lösen kann, sondern eine Art Training, welches einem Model das Lösen eines Problems beibringt. In der Quelle wird Reinforcement Learning in einer Aufzählung von Aufgaben für ML-Modelle aufgeführt und daher hier ebenfalls erwähnt.

2.2.2 Neuronale Netze für Bildverarbeitung

Neuronale Netze, speziell Convolutional Neural Networks (CNNs) [ZLLS21, S. 225], eignen sich besonders gut für Bildverarbeitung und sind somit prädestiniert für Computer Vision. CNNs enthalten spezielle Layer, die Convolutional Layer, welche den Input aus dem vorigen Layer durch Filteroperationen komprimieren. Diese Operationen können sehr gut parallelisiert und somit auf mehrere Kerne auf einer GPU (Graphics Processing Unit) aufgeteilt werden, was zu einer signifikant schnelleren Verarbeitung des Inputs führt. Es müssen allerdings nicht zwangsläufig Bilddaten mit CNNs ausgewertet werden. Auch die Verarbeitung eindimensionaler Inputs wie etwa von Audiosignalen profitiert von der Architektur und wird häufig verwendet. Da in dieser Arbeit jedoch nur Bildinformationen verarbeitet werden, beschränkt sich dieser Abschnitt ebenfalls nur auf Bildverarbeitung.

Convolutional Layer

Die Abbildung 2.3 zeigt eine Beispielberechnung, wie ein Bild, als eine 3×3 -Matrix dargestellt, mit einem 2×2 -Kernel gefiltert wird. Die blau hinterlegten Felder werden zu einem Feld komprimiert, und zwar derart, dass der Wert des Inputs mit dem Wert des Kernels an der entsprechenden Stelle multipliziert wird [ZLLS21, S. 231]. Das wird für jede Position wiederholt und das Ergebnis addiert, also $0 \cdot 0 + 1 \cdot 1 + 3 \cdot 2 + 4 \cdot 3 = 19$. Anschließend wird der Kernel verschoben und die Prozedur mit dem neuen Matrixausschnitt wiederholt. Auf diese Weise wird die Input-Matrix verkleinert, indem mehrere Pixel zusammengefasst werden. Sowohl der Bild als auch der Kernel können aber verschiedene Größen haben. Typische Kernelgrößen sind 3×3 , 5×5 , et cetera. Der Prozess in einem solchen Layer nennt sich Convolution.

$$\begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 0 & 1 \\ \hline 2 & 3 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 19 & 25 \\ \hline 37 & 43 \\ \hline \end{array}$$

Abbildung 2.3: Funktionsweise Convolution, Quelle [ZLLS21, S. 231]

An dem gegebenen Beispiel der Convolution wird auch ersichtlich, dass die Filteroperation kommutativ ist und somit parallelisiert werden kann. Der Kernel wird dabei meist zufällig initialisiert, im Zuge des Trainings werden die Werte im Kernel angepasst. [ZLLS21, S. 233].

Pooling Layer

Pooling funktioniert ähnlich wie Convolution, jedoch ohne Kernel. Es gibt stattdessen ein sogenanntes Pooling-Window, welches genau wie der Kernel sukzessive über die Input-Matrix geschoben wird [ZLLS21, S. 246]. Die durch das Pooling Window markierten Felder werden ebenfalls zu einem Wert zusammengefasst, was meistens der Durchschnitt (Average Pooling) oder das Maximum (Max Pooling) ist, wie in Abbildung 2.4 zu sehen.

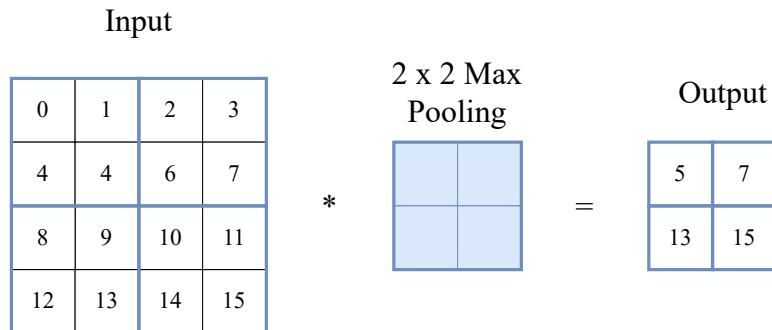


Abbildung 2.4: Funktionsweise Pooling, Quelle [ZLLS21, S. 246]

Durch insbesondere Pooling- und teilweise Convolution-Layer wird der Input größer, je weiter sich ein Layer im CNN befindet, während die relevanten Features erhalten bleiben. Je größer das Pooling Window ist, desto stärker wird die Input Matrix verkleinert [G17, S. 364].

Im Output Layer finden sich entsprechend wenige Neuronen, um eine Lösung der Aufgabe zu kommunizieren, zum Beispiel ob sich auf dem Input-Bild ein Apfel befindet (binärer Output, daher zwei Neuronen) oder welches Obst auf dem Bild zu sehen ist (mehrere Neuronen, zum Beispiel jeweils eins für Banane, Apfel, Orange, Kiwi und Birne). In jedem Fall wird die Input-Matrix in so eine Information transformiert, sodass Bildinformationen zwangsläufig stark aggregiert werden müssen.

Dense Layer

Ein Dense Layer, auch Fully Connected Layer genannt, ist allgegenwärtig im Bereich Neuronaler Netze. Er verbindet alle Input-Neuronen mit allen im Layer vorhandenen Neuronen [ZLLS21, S. 95] und enthält damit potentiell viele zu trainierende Gewichtungen. Daher wird bei CNNs dieser Layer hauptsächlich als Output Layer verwendet und nicht als Hidden Layer [ZLLS21, S. 226]. Bei anderen Neuronalen Netzen, die zum Beispiel Tabellendaten auswerten, ist das nicht zwangsläufig der Fall.

2.2.3 Transfer Learning

Transfer Learning [ZQD⁺21] ist eine spezielle Art, ein Model zu trainieren. Die Idee dabei ist, Wissen von einer Domäne in eine andere, jedoch ähnliche, zu transferieren. Das ist vor allem dann sinnvoll, wenn es nur wenige Trainingsdaten oder nur wenige gelabelte Trainingsdaten gibt. Außerdem kann die Lern-Performance erhöht werden. Ein Analog-Beispiel könnte folgendes Szenario sein: Es gibt zwei Personen. Die erste Person spielt exzellent Geige und besitzt viel musikalisches Hintergrundwissen und eine zweite Person, die kein musikalisches Hintergrundwissen hat, und kein Instrument spielen kann. Beide Personen sollen das Klavierspielen erlernen. Durch die Erfahrung und das Wissen der ersten Person wird diese das Spielen des neuen Instruments deutlich schneller erlernen können als die Person ohne musikalisches Wissen und Fertigkeit. So verhält es sich auch mit Transfer Learning. Haben zwei Domänen allerdings wenig gemein, wird Transfer Learning womöglich keinen positiven Effekt bringen. Das Wissen der ersten Person wird nicht dabei helfen, zu lernen, wie man Fahrrad fährt. Des Weiteren gibt es einen Effekt, der sogenannte negative transfer, der sich durch eine Verschlechterung der Performance auswirkt. Ähnlichkeiten der Domänen sind keine inhaltlichen Ähnlichkeiten und stören somit das Lernen und Anpassen an die neue Domäne.

Transfer Learning benötigt ein vortrainiertes Model und einen Ziel-Datensatz für die gewünschte Aufgabe des zu trainierenden Models. Das vortrainierte Model wird mit einem verwandten Datensatz trainiert. Anschließend wird es mit allen Gewichten und der Struktur der Layer in das zu trainierende Model kopiert. Eine Ausnahme dabei ist der Output Layer, denn dieser wird durch einen Output Layer speziell für die durchzuführende Aufgabe ersetzt. Mit einem erneuten Training wird der Output Layer von Grund auf neu trainiert, die übrigen Gewichte werden fein abgestimmt [ZLLS21, S. 578].

2.3 Wegfindung

Wegfindung dient dazu, in einem Suchbaum, dem ein Graph zugrunde liegt, einen Pfad zu einem Ziel zu finden. In diesem Kapitel werden kurz verschiedene Algorithmen und ihre Eigenschaften erklärt. Grundsätzlich kann man zwischen informierter und uninformerter Suche unterscheiden. Die informierte Suche hat Wissen über die Umgebung in Form einer Heuristik, die die Suche in die geschätzte richtige Richtung lenkt. Die uninformerste Suche arbeitet ohne eine solche Heuristik. Beispiele für uninformerste Suchen sind Breiten- und Tiefensuche, A* ist ein informierter Suchalgorithmus [NM15].

Tiefensuche

Bei der Tiefensuche [Cor07, S. 535ff] wird ein Pfad so lange abgelaufen, bis ein Blatt erreicht ist. Anschließend wird dieser bis zu einem Knoten zurückgelaufen, der noch unbesuchte Nachbarn enthält. Einer dieser Nachbarn wird anschließend besucht und der Pfad bis zum Blatt weiterverfolgt. Die Suche ist beendet, wenn das Ziel gefunden wurde oder alle Knoten durchsucht wurden. Dieser Algorithmus ist allerdings anfällig für Zyklen im zugrundeliegenden Graphen.

Breitensuche

Bei der Breitensuche [Cor07, S. 544ff] werden alle Knoten innerhalb einer Ebene des Baums durchsucht. Dieser Algorithmus findet garantiert den kürzesten Pfad in einem ungewichteten Graphen und hat somit keine Probleme mit Zyklen im Graphen.

Dijkstra

Dijkstra [Cor07, S. 598f] ist ein Algorithmus zur Routenfindung in einem gewichteten⁵ Graphen. Er basiert auf der Breitensuche, funktioniert aber mit einer Priority Queue, bei der die kürzeste Wegstrecke zu einem Knoten zuerst exploriert wird. Das heißt, wenn ein Knoten zwar auf einer tieferen Ebene im Baum ist als ein anderer Knoten, aber gemessen an den Kantengewichtungen schneller erreicht werden kann, wird dieser zuerst weiter verfolgt. Auch dieser Algorithmus ist nicht anfällig für Zyklen und findet ebenfalls den kürzesten Pfad.

⁵Durch die Gewichtung lassen sich verschiedene Kriterien an eine Route stellen, denn ein Kantengewicht kann verschiedene Bedeutungen haben. Es kann nicht nur aussagen, wie kurz oder lang eine Strecke ist, sondern zum Beispiel auch, wie sicher oder effizient sie ist oder wie viel Zeit benötigt wird, um das Ziel zu erreichen. Je nach Kriterium ändern sich die Kantengewichte und eine andere Route ist die bestmögliche [WFW20].

A*

Der A* Algorithmus [PM17, S. 89] ist ein optimierter Breitensuche-Algorithmus, der eine Heuristik nutzt, um den vielversprechendsten Pfad weiter zu explorieren. Die Heuristik liefert zu jedem Knoten einen Schätzwert, wie weit das Ziel entfernt ist. Während der Suche wird derjenige Pfad weiterexploriert, dessen Summe aus bisheriger Wegstrecke und der geschätzten verbleibenden Wegstrecke am geringsten ist.

2.4 Augmented Reality

Augmented Reality (AR) bezeichnet das Erweitern der Realität um eine virtuelle Realität. Sie unterscheidet sich von Virtual Reality (VR) insofern, als bei VR die Realität komplett ausgeblendet und durch eine virtuelle Welt ersetzt wird, während bei AR die Realität durch Informationen, Grafiken oder ähnliches erweitert (augmented) wird.

AR hat einen vielfältigen Einsatzbereich, etwa im Bereich Design, beispielsweise in der Inneneinrichtung mit der Ikea Place-App, wie in Abbildung 2.5a zu sehen ist oder im Bereich Montage oder Medizin, bei denen die Ausführenden mit Hilfe von AR durch den entsprechenden Prozess geleitet oder unterstützt werden, wie in Abbildung 2.5b zu sehen ist.

Im Bereich Bildung kann AR ebenfalls verwendet werden, zum Beispiel wenn durch AR chemische Experimente durchgeführt werden können, wie das mit der App Elements 4D [YMY18] möglich ist (Abbildung 2.5c).

Google Maps [LLC] nutzt zur Navigation ebenfalls Augmented Reality (Abbildung 2.5d) und im Freizeit-Bereich kommt AR auch zum Einsatz, etwa in Spielen wie Pokémon Go [Nia], in 2.5e zu sehen oder zum „Ausprobieren“ von Tattoos, wie das mit der App InkHunter [Ink] (Abbildung 2.5f) möglich ist.

Die Quellen der Bilder von Abbildung 2.5 finden sich in der Fußnote⁶.

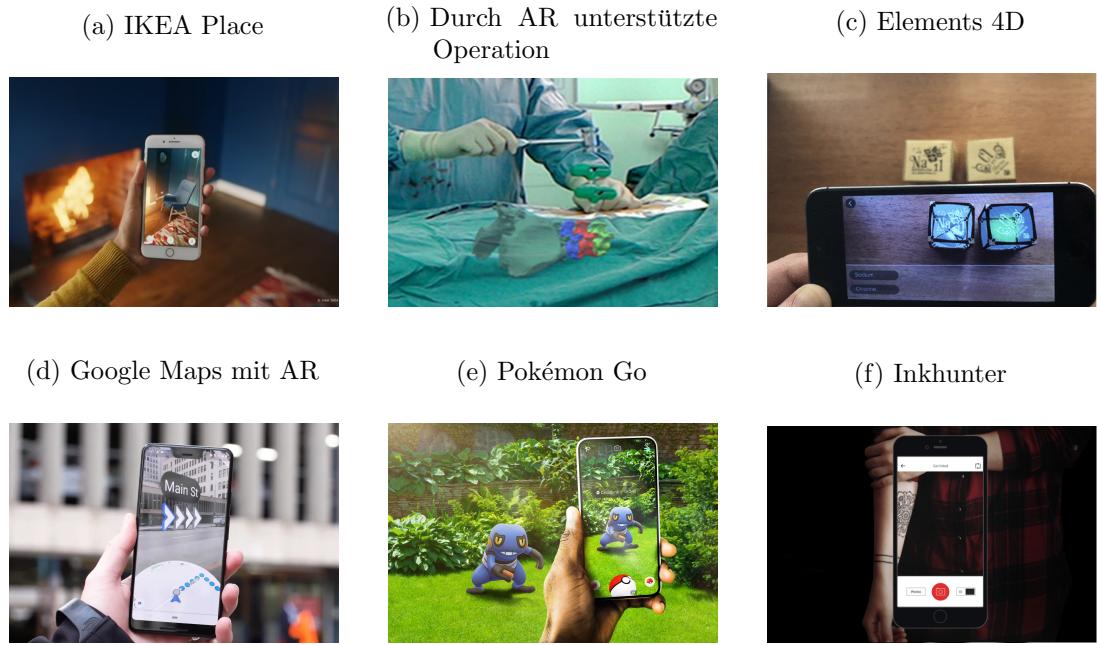


Abbildung 2.5: Verschiedene AR-Applikationen

Arten von AR

Die Art der AR-Anwendung kann anhand der verwendeten Geräte unterschieden werden. Zum einen gibt es sogenannte See Through-Geräte, durch die NutzerInnen direkt die Realität mit virtuellen Elementen überlagert sehen können, zum anderen gibt es Geräte, die mit Video-Komposition arbeiten. Hierbei wird eine Abbildung der Realität mit virtuellen Informationen überlagert. Ein Beispiel für ein See Through-Gerät ist eine Frontscheibe im Auto, auf der Navigationsinformationen oder die aktuelle Geschwindigkeit angezeigt werden, oder eine Brille. Geräte, die Video-Komposition nutzen, können unter Anderem Head Mounted Displays (HMD's) sein, auf denen die Realität durch eine Kamera aufgenommen und über das Display angezeigt wird, oder auch ein Mobiltelefon mit einer Kamera. [PSPS11]

Die Art einer AR-Anwendung lässt sich aber auch anhand ihrer Funktionsweise unterteilen. Hier können vier Arten unterschieden werden [EFSd19].

⁶Bildquellen der Abbildung 2.5

Die Angaben sind der Reihe nach von links nach rechts von oben nach unten.

IKEA Place: [Ste]

AR unterstützte Operation: [BPEET20]

Elements 4D: [YMY18]

Google Maps: <https://medialist.info/en/2019/05/12/google-maps-ar-navigation-with-augmented-reality/>, Zugriff 08.08.2022

Pokéémon Go: <https://pokemongolive.com/>, Zugriff 08.08.2022

Inkhunter: <http://www.inkhunter.tattoo/>, Zugriff 08.08.2022

Markerbasierte AR

Bei der markerbasierten AR [EFSd19] werden in der Realität platzierte Marker mit der AR-Anwendung erkannt. Solche Marker können zum Beispiel QR-Codes oder Symbole, wie auf der Abbildung 2.6 zu sehen, sein. Das Verfahren muss skalierungs- und rotationsinvariant sein, das heißt, der Marker darf beliebig groß und beliebig rotiert sein und muss trotzdem erkannt werden. Anhand der Position und Ausrichtung des ermittelten Markers können Informationen, 3D-Modelle oder Anderes an der gewünschten Stelle, zum Beispiel direkt auf oder neben dem erkannten Marker in der gewünschten Skalierung und Ausrichtung eingeblendet werden.



Abbildung 2.6: Markerbasierte AR, Bildquelle [EFSd19]

Markerlose AR

In einigen Anwendungen wird keine aufwendige Bildanalyse realisiert, sondern Informationen anhand von anderen Sensoren gewonnen, wie zum Beispiel der Position und der Ausrichtung [EFSd19]. Damit sind die Informationen allerdings nicht mit Objekten in der Realität verknüpft, sondern nur mit der Standortposition- und Ausrichtung. Das Objekt in der Realität wird also nicht direkt erkannt, stattdessen wird lediglich angenommen, dass es an der entsprechenden Stelle existiert. Die Überlagerung mit AR wird anhand der vermuteten Position ausgerichtet. Das macht diesen Ansatz leichtgewichtig und lokal auf weniger rechenstarken Geräten lauffähig.

Projektionsbasierte AR

Eine weitere Möglichkeit, AR zu nutzen, ist über Projektionen auf reale Objekte [EFSd19]. So können zum Beispiel verschiedene Designs eines Kleidungsstücks betrachtet

werden, ohne diese vorher aufwendig anfertigen zu müssen. Teilweise kann mit den Objekten interagiert werden.

Superimposition-basierte AR

Bei diesem Ansatz wird ein detektiertes Objekt aus der Realität durch ein virtuelles Objekt teilweise oder ganz ersetzt [EFSd19]. Dafür wird Objekterkennung benötigt. Diese kann entweder über klassische Bildverarbeitung (s. Abschnitt 2.5 und 2.1) oder Machine Learning (s. Abschnitt 2.2) realisiert werden.

2.5 Bildverarbeitung

In diesem Abschnitt werden einige Methoden der Bildverarbeitung, die auch in diesem Projekt verwendet sind, erklärt. Eine grundlegende Technik dabei ist die Kernel Convolution, die in Kapitel 2.2 bei der Funktionsweise der Convolutional Layer bereits beschrieben wurde. Ein Kernel wird Stück für Stück über das Bild geschoben. Ausgehend von dem durch den Kernel ausgewählten Bildausschnitt und dem Kernel selbst werden mathematische Operationen durchgeführt, die zu einer neuen Matrix führen.

	(a) Boxfilter	(b) Sobelfilter
$\frac{1}{9}$	$\begin{array}{ c c c } \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$	$\begin{array}{ c c c } \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$

Abbildung 2.7: Kernel für die Bildverarbeitung

Ein praktisches Werkzeug in der Bildverarbeitung ist die Kantenerkennung, die ebenfalls auf einer solchen Convolution des Bildes beruht. Durch Rauschen im Bild kann eine Kantenerkennung aber beeinträchtigt werden. Ein Glätten des Bildes kann zu einer Verbesserung dieses Phänomens führen, wie es auch in diesem Projekt der Fall war. Glättungen können ebenfalls mit einer Kernel Convolution durchgeführt werden. Der einfachste Kernel ist ein Box Kernel [RCG18, S. 165], der ein Bild weichzeichnet und dessen Koeffizienten alle den selben Wert, typischerweise 1, haben. Daneben gibt es einen Normalisierungsfaktor, der die Summe aller Koeffizienten enthält, damit das Bild nicht heller oder dunkler wird. Ein Beispiel ist in Abbildung 2.7a zu sehen.

Nachdem das Bild weichgezeichnet wurde, kann ein Kanten-Kernel [RCG18, S. 709ff] eingesetzt werden, um eine Kantenerkennung zu realisieren. Ein Sobel-Kernel ist ein solcher Kanten-Kernel und in 2.7b zu sehen. Das Ergebnis einer Filteroperation mit diesem Kernel ergibt einen hohen Wert, wenn eine vertikale Linie durch die Mitte des Kernels führt, da hier die Gewichtungen mit 2 am höchsten sind. Ist das nicht der Fall, gleichen sich die Gewichtungen im Kernel untereinander aus, da die Summe dieser insgesamt 0 ist, und resultieren in einem kleinen Wert und damit einer geringen Intensität nahe 0. Je höher der Wert der Filteroperation, desto höher ist auch die Intensität im Ergebnisbild nach der Kantenerkennung.

Der Filter kann auch rotiert werden, sodass horizontale oder diagonale Linien erkannt werden.

Hough Lines

Der Hough Lines Algorithmus dient ebenfalls der Erkennung von Linien in einem Bild, diese können aber eine beliebige Steigung besitzen [IK88].

Der Algorithmus benötigt ein binäres Inputbild, welches aus einer Kantenerkennung hervorgegangen ist. Jedes Pixel, welches in dem Binärbild als Kante eingestuft wird, wird in den Hough Raum übertragen. Das geschieht, indem für einen solchen Punkt alle Linien betrachtet werden, die diesen enthalten. Ein Punkt im Hough Raum entspricht einer Linie im Inputbild, umgekehrt entspricht ein Punkt im Binärbild einer sinusartigen Linie im Hough Raum. Wurden alle erkannten Kantenpunkte übertragen, ergeben sich bestimmte Ballungspunkte im Hough Raum, aus denen sich die im Ursprungsbild erkannten Linien ableiten lassen.

2.6 Zusammenfassung

Das Grundlagenkapitel behandelt die Bereiche Indoor Navigation, Machine Learning, Wegfindung, Augmented Reality und Bildverarbeitung.

In der Entwicklung der Applikation kommt Computer Vision mit Hilfe von Machine Learning sowohl in der Navigation selbst, als auch in der Augmented Reality-Komponente vor. Der AR-Teil ist markerlos, das heißt, es werden keine QR-Codes oder Ähnliches verwendet, sondern Objekterkennung. Beide Modelle basieren auf einem Neuronalen Netz, welches Convolution für ihren jeweiligen Zweck verwendet. Insbesondere für AR ist eine Vorverarbeitung des Bildes mit Bildverarbeitungsmethoden notwendig, damit die Objekterkennung besonders gut funktioniert.

In den folgenden beiden Kapiteln werden Anforderung an die Anwendung formuliert und anschließend die Realisierung, bei der das hier formulierte Wissen zum Einsatz kommt.

3

Indoor Navigation

Dieses Kapitel handelt von der Indoor Navigation, die aus einer Standortermittlung und dem Finden einer Route besteht. Diese Information muss an den User übermittelt werden, der AR-Aspekt der Anwendung wird in diesem Kapitel allerdings außenvorgelassen und in Kapitel 4 besprochen. In den folgenden Abschnitten wird erläutert, was die grundsätzliche Funktionsweise der Indoor Navigation ist, welche Aufgabe sie löst und welche funktionalen und nicht funktionalen Anforderungen es an sie gibt. Anschließend wird die gewählte Herangehensweise erörtert.

3.1 Grundsätzliche Funktionsweise

Die Funktionalität der Indoor Navigation in diesem Projekt setzt sich aus der Standortermittlung und der Wegfindung zusammen. Durch die Lokalisierung wird anhand eines oder mehrerer aufgenommener Bilder ein Standort innerhalb der Fachhochschule Wedel ermittelt. Diese Bilder werden per Handykamera aufgenommen und an das Backend gesendet. Zur Verdeutlichung wird ein Use Case herangezogen. Die Applikation ist schematisch in Abbildung 3.1 dargestellt.

Use Case: Finden eines Hörsaals

Frau Muster ist neu eingeschriebene Studentin an der Fachhochschule Wedel. An ihrem ersten Vorlesungstag findet die erste Vorlesung in Hörsaal 5 statt. Da die Räumlichkeiten noch unvertraut für sie sind, weiß sie nicht, wo sich dieser befindet. Sie sieht sich nach einem Lageplan für die FH um.

Frau Muster findet einen Lageplan sowie einen QR-Code, der die URL codiert, unter der die Indoor Navigation erreichbar ist. Sie scannt diesen mit ihrem Smartphone und die Web-Applikation wird geladen. Über die Handykamera wird die Umgebung gefilmt und in einem Fenster innerhalb der Webapplikation wiedergegeben. Über

3 Indoor Navigation

ein Drop Down Menü kann sie ihr Ziel, also Hörsaal 5, auswählen, wie schematisch in Abbildung 3.1a dargestellt. Anschließend wird sie aufgefordert, sich ungefähr in die Mitte des Gangs zu stellen und ihr Handy für eine genauere Standortermittlung etwas zu schwenken (Abbildung 3.1b). Dann startet die Navigation und gibt ihr Informationen, wann und in welche Richtung sie abbiegen oder ob sie Treppen nehmen muss, wie in Teilabbildung 3.1c zu sehen ist, bis sie das Ziel erreicht.

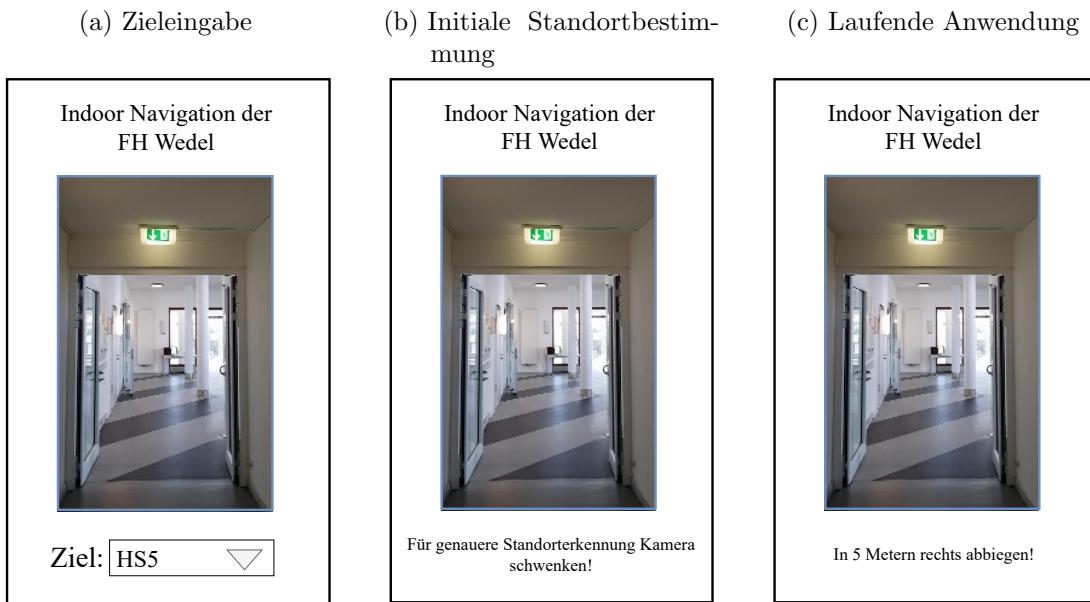


Abbildung 3.1: Ablauf Indoor Navigation-Teilanwendung

3.2 Standortermittlung

Die Standortermittlung ist die Basis der gesamten Navigationsanwendung. Diese wurde mit Machine Learning realisiert. Dieser Ansatz hat den Vorteil, dass keine zusätzliche Infrastruktur benötigt wird, etwa Emitter oder Receiver, wie bei kommunikationstechnologischen Ansätzen oder eine Möglichkeit zur Rekalibrierung wie mit Pedestrian Dead Reckoning, wie in Kapitel 2.1 erwähnt. Es werden ebenfalls keine eigens dafür konzipierten Geräte benötigt, sondern nur ein Handy mit Kamera und einem Internetzugang. Ein Internetzugang ist nötig, um per Kamera aufgenommene Bilder an das Backend zu schicken, denn dort findet die rechenintensive Auswertung des Bildes mit dem Machine Learning Model statt. Während der Navigation muss die Kamera permanent angeschaltet sein, was sich negativ auf den Akkustand auswirken kann. Allerdings ist lediglich mit einer kurzen Nutzungsdauer bedingt durch die Gebäudegröße

der Fachhochschule zu rechnen. Zudem ist das Modell auf ein bestimmtes Aussehen der Umgebung trainiert. Wenn sich zum Beispiel durch Renovierungsarbeiten das Aussehen von Wände, Fußböden oder Mobiliar ändert, kann das negative Auswirkungen auf die Standortermittlung haben.

3.3 Wegfindung

Sobald die Position eines/r NutzerIn bekannt ist, kann der Wegfindungsalgorithmus gestartet werden. Anschließend wird die berechneten Route zurück an das Handy des/der NutzerIn übermittelt und dort die entsprechenden Informationen aufbereitet angezeigt. In Kapitel 2 wurden im Abschnitt Wegfindung verschiedene Routenfindungsalgorithmen beschrieben.

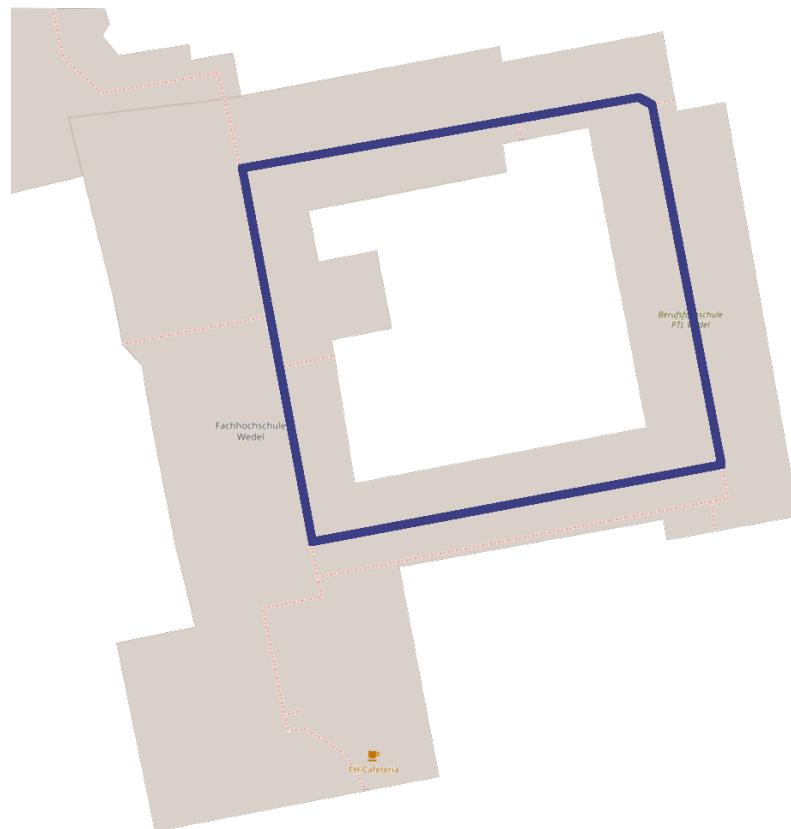


Abbildung 3.2: Rundgang im Erdgeschoss der FH Wedel

In diesem Szenario ist der Graph ungewichtet und ungerichtet, da Gänge immer in alle möglichen Richtungen abgelaufen werden können. Alle Knoten sind gleichweit zu ihren Nachbarn entfernt. Durch den Rundgang im Erdgeschoss der FH Wedel, wie in Abbildung 3.2 eingezeichnet, darf der gewählte Routenfindungsalgorithmus nicht anfällig

für zyklische Strukturen sein. Daher wurde hier Breitensuche implementiert. Da der Suchraum begrenzt ist, wurde der Algorithmus nicht weiter optimiert.

3.4 Anforderungen an die Indoor Navigation

In diesem Abschnitt werden nacheinander die funktionalen und nicht funktionalen Anforderungen für diesen Teil des Projekts erläutert. Dabei werden Standortermittlung und Wegfindung zusammen betrachtet,

3.4.1 Funktionale Anforderungen

1) Zieleingabe

NutzerInnen muss die Möglichkeit geboten werden, ein Navigationsziel eingeben zu können.

2) Abweichung vom berechneten Pfad

Sollte die ermittelte Position vom Pfad abweichen, zum Beispiel, weil ein/e NutzerIn aus Versehen oder mit Absicht nicht den übermittelten Weginformationen gefolgt ist, muss die Navigation einen neuen Pfad vom aktuellen Standpunkt aus berechnen.

3) Machine Learning

Der Standort und die Blickrichtung von NutzerInnen müssen zudem hauptsächlich mit maschinellem Lernen und Computer Vision bestimmt werden. Das heißt, ein ML-Model nimmt ein Bild entgegen und liefert eine Position inklusive der Blickrichtung als Ergebnis. Für eine präzisere Bestimmung der Position darf aber auch eine Anreicherung der Information aus anderen Quellen verwendet werden, s. Kapitel 3.2.

4) Falsch ermittelte Standorte verwerfen

Von der Standortermittlung falsch ermittelte oder zu ungenaue Standorte müssen verworfen werden, damit User nicht unnötig irritiert werden.

5) Cross-Plattform

Des Weiteren muss die Applikation Cross-Plattform sein, damit die Navigation unabhängig des Betriebssystems auf jedem Handy mit Internetzugang und Kamera genutzt werden kann.

6) Anzeige der nächsten Abbiegung

NutzerInnen muss mitgeteilt werden, in welcher Entfernung und in welcher Richtung die nächste Abbiegung ist. Dafür muss die berechnete Route oder relevante Teile davon ans Frontend überliefert werden.

7) Ziel erreicht

Usern muss mitgeteilt werden, wenn sie ihr Ziel erreicht haben.

8) JSON-Format

Die Kommunikation zwischen Frontend und Backend muss im JSON-Format (Javascript Object Notation) stattfinden. JSON ist ein menschenlesbares und leichtgewichtiges Format zur Datenübertragung, welches sowohl von dem Frontend mit Javascript als auch dem Backend mit Python verarbeitet werden kann.

3.4.2 Nicht funktionale Anforderungen

1) Echtzeit

Eine der wichtigsten nicht-funktionalen Anforderungen ist die Antwortzeit. Die berechneten Routeninformationen müssen ausreichend schnell an den/die NutzerIn zurückübermittelt werden können. Die angenommene und erwartete Geschwindigkeit der Person ist in Innenräumen Schrittgeschwindigkeit, was ungefähr zwischen 5 und 10 km/h liegt. Mit einer Geschwindigkeit von 5 km/h bewegt sich ein Mensch innerhalb einer Sekunde circa $5000m/3600s = 1,39m/s$. Mit 10 km/h sind es entsprechend $2,78m/s$. Die Gänge der Fachhochschule wurden in Abschnitte von etwa 2,5 Metern Länge unterteilt (s. Kapitel 5.5.2). Daher bringt eine schnellere Antwortzeit keinen Vorteil, da die Standortbestimmung nur diskret einzelne Abschnitte bestimmen und keine Zwischenräume abbilden kann. Mit einer Antwortzeit $< 0,5$ Sekunden wird bei langsamem Gehen spätestens alle 0,7 Meter der Standort aktualisiert. Geht eine Person also langsam mit 5 km/h durch einen Gang, wird beim Durchlaufen eines Abschnitts drei bis vier mal der Standort ermittelt, bei sehr schnellem Gehen alle 1,39 Meter, also ein bis zwei Mal pro Abschnitt. Daher muss eine Antwort in weniger als 0,5 Sekunden erfolgen.

2) Kürzeste Route

Die Routenfindung muss so implementiert werden, dass immer die kürzeste Route gefunden wird, damit NutzerInnen schnell ihr Ziel finden.

3) Genauigkeit der Standortermittlung mit dem Trainingsdatensatz

Das Machine Learning-Modell muss den Standort sehr präzise ermitteln können. Damit auch im Betrieb eine ausreichend genaue Standortermittlung sichergestellt werden kann, soll das verwendete Modell mindestens eine Genauigkeit von 95% besitzen.

4) Genauigkeit der Standortermittlung im Betrieb

Während des Betriebs der Applikation befinden NutzerInnen sich nicht immer an den Positionen, an denen die Videoaufnahmen gemacht wurden. Trotzdem muss die Standortermittlung das korrekte Ergebnis liefern. Als richtig gilt eine der beiden Positionen, zwischen denen sie sich befinden.

5) Sinnvolle Informationen

Eine nicht-funktionale Anforderung ist, dass die Informationen sinnvoll, richtig und ausreichend genau sind. Das heißt, die gegebenen Informationen sollen ausreichend genau sein, dass der/die NutzerIn das Ziel findet, ohne selbst nennenswert suchen zu müssen. Hierzu zählt nicht die Einblendung von Symbolen auf die Kamera durch den Augmented Reality-Teil, sondern nur die textuelle Information.

6) Minimale Datenübertragung

Außerdem sollte die Menge der übertragenen Daten möglichst gering sein, da GästInnen sich nicht immer mit dem WLAN verbinden und somit unnötig mobiles Datenvolumen verbrauchen, insbesondere weil durch diese Anwendung kontinuierlich Bilder verschickt werden. Hierbei muss aber gewährleistet sein, dass die Berechnung der Position mit dem entsprechenden Bild ausreichend genau ist und gleichzeitig der Anforderung an eine schnelle Antwort genügt.

7) Benutzerfreundlichkeit

Das System sollte benutzerfreundlich sein. Da es auch von GästInnen genutzt werden soll, die die Fachhochschule nur wenige Male besuchen, darf keine zusätzliche Erklärung oder Einführung benötigt werden. Das System muss intuitiv zu bedienen sein.

3.5 Verwandte Arbeiten

Wie in den Grundlagen bereits angedeutet, gibt es verschiedenste Herangehensweisen, um eine Indoor Navigation zu ermöglichen. Die Nutzung von maschinellem Lernen ist keine Seltenheit, aber auch keine Notwendigkeit. In diesem Kapitel wird sich auf die Nutzung von Machine Learning beschränkt. Hierbei kann unterschieden werden, ob Computer Vision verwendet wird oder nicht. Bei der Betrachtung der verwandten Arbeiten wird der Aspekt, wie Navigationsinformationen zu NutzerInnen gelangen, ausgeklammert und im Kapitel 4 besprochen. Da Machine Learning zurzeit ein schnelllebiger Themenbereich ist, wurden hier nur aktuelle Arbeiten aus den letzten drei Jahren betrachtet.

3.5.1 Nutzung von Computer Vision

In der Arbeit von Khan, Ullah und Nabi [KUN19] wurde 2019 ein markerbasierter Ansatz zur Positionsbestimmung implementiert. Die Marker wurden auf Papier gedruckt und an Decken befestigt. Diese können anschließend per Kamera erkannt werden. Anhand dieser werden Navigationsinformationen zur Verfügung gestellt. Für die Marker-Erkennung gibt es verschiedene Bibliotheken, wie ARTag [Fia05] oder ARToolKit [KB99]. Diese funktioniert mit einer korrekten Klassifizierung von 96,9%. Die Repräsentation des Gebäudes wird automatisch generiert, indem die Marker zu einem Graphen verbunden werden.

Die Arbeit von Kunhoth et al. [KKAmAA19] aus dem Jahr 2019 beinhaltet die Entwicklung drei verschiedener Varianten für die Standortermittlung. Eine davon funktioniert mit Hilfe eines CNNs, eine anderes über QR-Codes und die letzte über BLE (Bluetooth Low Energy), welche hier nicht weiter betrachtet wird. Die ersten beiden Ansätze nutzen eine Kamera, welche Bilder an einen Server sendet, der eine Bildauswertung durchführt. Das CNN verwendet Pooling Layer und Dense Layer und wird dazu genutzt, anhand eines Bildes die Position zu bestimmen, an der es aufgenommen wurde, wie das auch in diesem Projekt realisiert wurde. Die zweite Variante nutzt gedruckte QR-Codes an Wänden, auf Böden und an Türen, welche ebenfalls durch einen Server auf einem Input-Bild detektiert und klassifiziert werden, um die Position zu ermitteln. Die Güte der Standortgenauigkeit wurde mit *Medium* bewertet und ist damit im Vergleich zu der BLE-Variante, welche mit *Low* bewertet wurde, genauer, es wurden aber keine

weiteren Zahlen, wie prozentuale richtige Klassifizierung oder Ähnliches erwähnt. Für die Repräsentation des Gebäudes wurden Indoor Maps und Lagepläne verwendet.

Die Arbeit von Fusco und Coughlan [FC20] aus 2020 nutzt für das Tracken des Standorts eines Users den Kamerainput eines Smartphones sowohl für PDR, denn aus diesem werden Bewegungen des Users abgeleitet, als auch für die Erkennung von Schildern, in diesem Fall speziell Notausgangsschilder, da durch die feste Größe des Schildes die Entfernung zu diesem ermittelt und für die Positionsermittlung verwendet werden kann. Ein Schild ist nicht immer eindeutig einer Position zuzuordnen, sodass mehrere mögliche Szenarien betrachtet werden müssen. Mit der Zeit konvergiert mit dem dort verwendeten Algorithmus die getrackte Position und konnte eine ungefähre Genauigkeit von einem Meter erreichen.

3.5.2 Nutzung von ML ohne Computer Vision

Machine Learning muss nicht im Computer Vision-Kontext eingesetzt werden, sondern kann auch zur Präzisierung von zum Beispiel kommunikationstechnologischen Methoden wie etwa der Nutzung von WLAN-Signalen verwendet werden. In der Arbeit von Cui et al. [CLZ⁺20] aus 2020 wird genau das realisiert. Hier wird die Position eines/r NutzerIn über WiFi-Fingerprinting bestimmt. Dabei wird der Extreme Learning Machine-Algorithmus verwendet, der ein Feed Forward Neural Network¹, welches nur einen Hidden Layer besitzt, sehr schnell trainieren kann, kombiniert mit einer Principal Component Analysis (PCA), mit der ein Datensatz vereinfacht werden kann. Das Model wird während der Online-Phase, also während der Nutzung des Systems, kontinuierlich weitertrainiert. Im Vergleich zu anderen Algorithmen, wie etwa K-Nearest Neighbor oder Extreme Learning Machine ohne PCA, konnte dieser Ansatz den Positionierungsfehler reduzieren.

3.5.3 Hybride Ansätze

Es gibt auch hybride Positionierungsalgorithmen, die sowohl Computer Vision als auch andere Technologien kombiniert nutzen. Ein Beispiel dafür ist die Arbeit von Qian et al. [QCYL20] aus 2020. Zur grundlegenden Positionsbestimmung wird PDR (Pedestrian Dead Reckoning) verwendet. Um die Nachteile wie etwa das Kumulieren von Abweichungen in der Bewegungsmessung auszugleichen und um festzustellen, ob

¹Ein Feed Forward Neural Network besitzt nur Verbindungen in eine Richtung.

das Stockwerk gewechselt wurde, wird anhand von RGB-Bildern und einem CNN-Model zusätzlich die Position bestimmt. Die ermittelte Position sowohl aus dem CNN als auch die mit PDR werden kombiniert und führen mit 0,81 Metern zu deutlich weniger Abweichungen der realen Position im Vergleich zu einer Standortermittlung nur mit PDR (1,74 Meter).

Im Vergleich zur Arbeit von Khan et al. oder einem Teil der Arbeit von Kunhoth et al. werden in dieser Arbeit keine Marker oder QR-Codes zur Positionsbestimmung verwendet. Stattdessen wurde eine markerlose Positionserkennung mit Hilfe eines CNNs implementiert, wie das in einem anderen Teil der Arbeit von Kunhoth et al. oder der Arbeit von Qian et al. entwickelt wurde. Ein Fingerprinting Ansatz mittels WLAN, wie von Cui et al. vorgeschlagen oder die Nutzung von PDR wurde nicht in Betracht gezogen.

3.6 Zusammenfassung

In diesem Kapitel wurde die grundsätzliche Funktionsweise einer möglichen Indoornavigation für die FH Wedel anhand eines Use Case vorgestellt. Zunächst wird das Ziel ausgewählt, die initiale Standortermittlung und anschließend die Navigation durchgeführt. Ebenso wurden die verschiedenen Anforderungen wie Echtzeit oder das Finden der kürzesten Strecke formuliert. Im folgenden Kapitel wird zunächst die Augmented Reality-Komponente beleuchtet und ebenfalls Anforderungen formuliert. Die jeweiligen Anforderungen werden in Kapitel 6 wieder aufgegriffen und deren Erfüllung geprüft.

4

Augmented Reality

Augmented Reality erweitert die Realität um beispielsweise Text, Grafiken oder Animationen. Dabei sollte eine AR-Anwendung laut Microsoft¹, die selbst AR-Geräte wie die Microsoft HoloLens herstellen, folgende Eigenschaften integrieren können.

1. Kombination der Realität mit virtuellen Elementen, die virtuellen Elemente überlagern also die Realität
2. Echtzeit-Interaktion, damit sich die Anwendung durch eine schnelle Antwortzeit möglichst natürlich und intuitiv anfühlt
3. akkurate 3D-Objekterkennung, damit die virtuellen Elemente korrekt in der Szene platziert werden können

Diese drei Gesichtspunkte werden auch in dieser Arbeit fokussiert betrachtet.

4.1 Grundsätzliche Funktionsweise

Diese AR-Anwendung arbeitet mit Video-Komposition. Abbildung 4.1 zeigt den schematischen Aufbau der Anwendung.

Pfeile oder andere Symbole werden als ein Overlay auf die Kameraaufnahme gelegt (Eigenschaft 1) und geben so die berechneten Weginformationen an. Das beschränkt sich dabei auf die nächste Bewegungsrichtung. Die Berechnung der Art, Größe und Ausrichtung der Pfeile und Symbole soll in Echtzeit möglich sein, damit keine signifikante Verzögerung sichtbar ist (Eigenschaft 2). Um die Positionen und Ausrichtung der Pfeile und Symbole zu berechnen, ist vor allem die Erkennung des Bodens wichtig. Anhand des Bodens kann erkannt werden, wie weit sich der Flur in Blickrichtung erstreckt und

¹<https://www.microsoft.com/en-us/hololens>, Zugriff 18.07.2022

Indoor Navigation der FH Wedel



In 5 Metern rechts abbiegen!

Abbildung 4.1: Indoor Navigation mit Augmented Reality

somit, wann ein/e NutzerIn abbiegen muss. Mit perspektivischen Berechnungen kann ermittelt werden, auf welcher Höhe des Gangs das Navigationsziel liegt.

Der grobe Ablauf der Anwendung ist in Abbildung 4.2 beschrieben.

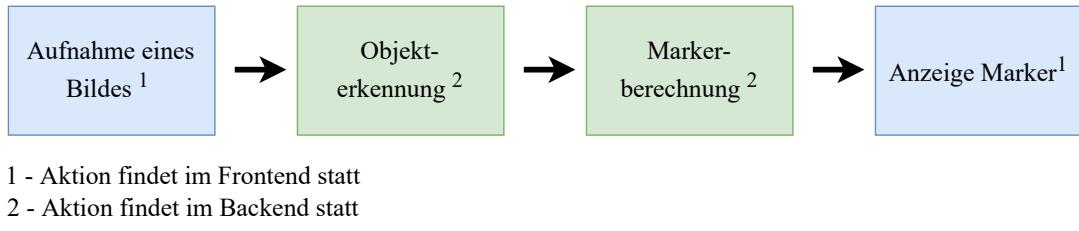


Abbildung 4.2: Ablauf der AR-Teilanwendung

Teile der Anwendung finden im Frontend, andere im Backend statt. Wie bei der Indoor-Navigation-Teilanwendung auch, wird im Frontend ein Bild aufgenommen, welches an das Backend übermittelt wird. Dort wird die Objekterkennung durchgeführt. Diese wird in diesem Fall nur dazu verwendet, den Fußboden zu detektieren und wurde mit einem Segmentation-Model [ZZP⁺18] [ZZP⁺17]² implementiert, welches 150 Kategorien erkennt. Eine Bildsegmentierung ordnet jedem Pixel eines Eingabe-Bildes eine Kategorie beziehungsweise eine Wahrscheinlichkeit pro Kategorie zu. Das bedeutet, zu einem Pixel gehören 150 Wahrscheinlichkeiten. Diese Kategorien sind sehr allgemein gehalten. Das Model kann zum Beispiel Türen, Böden, Fenster, Bäume, Schilder, Straßen, Duschen oder den Himmel klassifizieren. Sie beschränkt sich also nicht nur auf Innenräume. Ist der Boden erkannt, wird eine konvexe Hülle um alle erkannten Bodenflächen gezeichnet. Das dient zum Ausschließen von Artefakten, wenn der Boden an einigen Stellen falsch klassifiziert wurde oder andere Objekte die Sicht auf den Boden versperren. Ist diese Berechnung ebenfalls abgeschlossen, werden per perspektivischer Berechnung die Art, Ausrichtung und Position der anzuzeigenden Symbole ermittelt. Diese werden zukünftig auch als Marker bezeichnet, sind aber nicht zu verwechseln mit den AR-Markern beschrieben in Abschnitt 2.4, die zum Beispiel durch QR-Codes realisiert werden. Diese Marker markieren die Position und beinhalten zusätzlich die Art und Ausrichtung eines Symbols. Die berechneten Informationen werden an das Frontend zurückgesendet, welches diese aufbereitet auf das Kamerabild rendert.

²Repository: [Vis]

4.2 Anforderungen an die Augmented Reality

Auch dieser Teil der Anwendung muss funktionale wie nicht-funktionale Anforderungen erfüllen, die im Folgenden aufgelistet und gegebenenfalls erörtert werden.

4.2.1 Funktionale Anforderungen

1) Objekterkennung

Um die richtigen Positionen der entsprechenden Marker für die Navigation zu berechnen, ist eine gute Objekterkennung nötig. Dafür müssen der Boden, Abzweigungen und Türen korrekt erkannt werden.

2) Korrekte Symbole

Die angezeigten Symbole müssen entsprechend der berechneten Route korrekt sein, das heißt, sie müssen in ihrer Ausrichtung korrekt sein, dürfen also nicht in die falsche Richtung zeigen, an geeigneter Stelle platziert werden und die korrekte Art des Symbols muss ausgewählt werden.

3) Berechnung der Positionen der anzuzeigenden Symbole

Die Symbole, die im Frontend angezeigt werden sollen, müssen perspektivisch korrekt berechnet worden sein, damit der Eindruck von Symbolen auf dem Boden entsteht.

4) JSON-Format

Die Kommunikation zwischen Frontend und Backend muss im JSON-Format stattfinden.

4.2.2 Nicht funktionale Anforderungen

1) Echtzeit

Auch dieser Teil der Anwendung soll in Echtzeit möglich sein. Echtzeit ist hier aber enger gefasst, das heißt, die Berechnung muss schneller erfolgen als im Kapitel 3 beschrieben, denn der Kamerainput kann sich von Frame zu Frame ändern. Darauf muss in der Anwendung reagiert werden. Das ist vor allem an Abbiegungen relevant, denn dort ändert sich der Inhalt des Bildes signifikant. Läuft der/die NutzerIn einen Gang geradeaus entlang, wird sich der Bildausschnitt tendenziell nur marginal von Frame zu Frame ändern, zum Beispiel eine erhöhte oder verringerte Position der Kamera, während die Position der Marker im Wesentlichen ähnlich bleibt. Trotzdem sollte die Position der

entsprechenden Symbole schnellstmöglich angepasst werden, um NutzerInnen ein flüssiges AR-Erlebnis zu ermöglichen. Daher sollte die Framerate, welche in FPS angegeben wird, ungefähr bei 30FPS liegen. Dies ist die Framerate, die im Allgemeinen für Videos verwendet wird und somit für das menschliche Auge flüssige Bewegungen darstellen kann [ARF13].

2) Ausreichende Genauigkeit

Die angezeigten Informationen sollen so genau sein, dass ein/e NutzerIn zum einen die schriftlichen Informationen nicht benötigt und zum anderen möglichst wenig nach dem Navigationsziel suchen muss.

4.3 Verwandte Arbeiten

In diesem Abschnitt werden verschiedene verwandte Arbeiten betrachtet hinsichtlich der Nutzung einer Implementation mit Augmented Reality für eine Navigationsanwendung. Dabei wird in diesem Kapitel besonderer Fokus auf die Durchführung der AR-Komponente und weniger auf der Positionierungs- und Routenfindungs-Komponente gelegt.

4.3.1 Markerlose AR

Die Arbeit von Huang et al. [HHCW20] beschäftigt sich mit der Implementation eines Indoor Navigationssystems ARBIN, welches die Technologie BLE (Bluetooth Low Energy) zur Positionierung nutzt, zusammen mit der Nutzung von ARCore, einer AR-Bibliothek von Google. Die Platzierung der AR-Elemente wird nicht durch Objekterkennung bestimmt, wie das in diesem Projekt der Fall ist, sondern durch die Ermittlung der Ausrichtung des Smartphones, und ist daher eine markerlose Anwendung.

Eine weitere Anwendung wurde 2021 von Karan et al. [KMSF21] entwickelt. Hier wird eine Outdoor- und eine Indoor-Navigationsanwendung entwickelt, die AR zur Nutzerführung verwendet. Wie in der Arbeit zuvor, wird ARCore, die Google Bibliothek, für das Erfassen einer 3D Szene aus einem 2D Bild verwendet. In dieser Applikation werden bestimmte Marker virtuell auf den Boden gezeichnet und verbunden. Diese sollen NutzerInnen den Weg visualisieren und in der Position fest sein, also sich nicht mit dem/der NutzerIn mitbewegen. Daher wird die Surface Detection von ARCore

verwendet. So erwecken die Marker den Anschein, sie seien in der Realität auf dem Boden platziert.

4.3.2 Markerbasierte AR

Das Indoor Navigationssystem einer Bibliothek entwickelt von Romli et al. [RRG⁺20] nutzt ebenfalls Marker für AR. Diese sind allerdings keine QR-Codes oder Barcodes, sondern bestimmte Bildausschnitte oder kleinere Objekte, die von der Anwendung erkannt werden. Dafür wurde mittels Vuforia, einer AR Engine, ein Modell trainiert, welches diese Bilder oder Objekte erkennen und anhand dieser die Positionierung und Ausrichtung von anzuzeigenden Elementen übernehmen kann.

Die vorgestellten Arbeiten nutzen verschiedene Ansätze, AR zu realisieren, nutzen aber alle eine Bibliothek im Vergleich zu dieser Arbeit, die eine eigene Objekterkennung³ implementiert und nutzt. Hierbei wird vor allem der Boden zur Platzierung von AR-Elementen genutzt, während bei den hier beleuchteten Arbeiten verschiedene Ansätze, wie etwa die Nutzung der Ausrichtung des Mobiltelefons, eine Surface Detection oder die Etablierung eigener spezieller Marker über Vuforia, Verwendung finden.

4.4 Zusammenfassung

In diesem Kapitel wurde die grundsätzliche Funktionsweise der AR-Komponente erläutert. Zuerst muss ein Bild aus dem Frontend im Backend ankommen, anschließend wird eine Objekterkennung durchgeführt und die Position, Ausrichtung und Art der AR-Elemente, die im Frontend angezeigt werden sollen, berechnet. In Abschnitt 4.2 wurden verschiedene Anforderungen postuliert, um Benutzerfreundlichkeit zu garantieren, beispielsweise durch Echtzeit und die Anzeige adäquater Informationen. Auch diese Anforderungen werden in Kapitel 6 hinsichtlich ihrer Erfüllung wieder aufgegriffen und bewertet.

Da alle Anforderungen und die grobe Funktionsweise sowohl von der Indoor Navigation als auch Augmented Reality beleuchtet wurde, wird im folgenden Kapitel die Realisierung des gesamten Projektes beschrieben.

³mit Hilfe des Segmentierungsmodels [Vis]

5

Realisierung

Dieses Kapitel beschäftigt sich mit der Realisierung von IndoorNav, dem Indoor-Navigationssystem. Zunächst werden die verwendeten Technologien vorgestellt, die Architektur und ein Klassendiagramm beleuchtet und der Kommunikationsfluss der einzelnen Komponenten der Anwendung beschrieben. Anschließend wird aufgezeigt, wie das Gebäude der FH Wedel intern dargestellt und wie die Indoor Navigation (Standortermittlung und Routenfindung) implementiert wurde. Dabei wird auf die Datengrundlage eingegangen, die für das anschließende Training des ML-Modells verwendet wurde. Im nachfolgenden Abschnitt wird die Architektur der Modelle erläutert. Ein Abschnitt über die Wegfindung schließt das Unterkapitel der Indoor Navigation ab. Anschließend geht es um die AR-Komponente. Hierbei wurden zwei verschiedene Ansätze ausgetestet, die im Detail vorgestellt werden.

5.1 Verwendete Technologien

In diesem Abschnitt werden die wichtigsten Technologien vorgestellt. Die vollständige Liste aller Bibliotheken befindet sich im Anhang (Kapitel 7.3) oder kann im Code selbst eingesehen werden.

Frontend

Das Frontend wurde mit React[MP] realisiert. Mit React können komponentenbasierte, interaktive Benutzeroberflächen gebaut werden. Jede Komponente verwaltet dabei ihre eigenen Status selbst, was ein effizientes Rerendering mit sich bringt. Nur die Komponenten, die ein Update benötigen, weil sich zum Beispiel der Status der Komponente geändert hat, werden neu gerendert, alle anderen nicht. Da React das Programmieren mit Javascript (und Typescript) erlaubt, kann Frontendlogik direkt innerhalb der Komponenten geschrieben werden. Da das Frontend sonst sehr einfach gehalten wurde, gibt es keine weiteren verwendeten Technologien.

Backend

Das Backend ist deutlich komplexer und verwendet daher mehr Technologien als das Frontend. Die Programmiersprache, in der der Programmcode geschrieben wurde, ist Python[Fou].

Flask und Flask API

Für die Kommunikation zwischen Backend und Frontend wird Flask API[Pal] verwendet. Flask ist ein leichtgewichtiges Framework für Webbapplikationen in Python. Flask ermöglicht WSGI (Web Server Gateway Interface) und somit die Verarbeitung und Beantwortung von Anfragen des Clients.

Tensorflow und Keras

Keras ist eine High Level API, geschrieben in und für Python, basierend auf TensorFlow 2 [Tena]. Tensorflow ist eine Plattform, mit der sich Projekte im Bereich maschinelles Lernen realisieren lassen. Keras [Ker] ist eine Schnittstelle zu Tensorflow und vereinfacht durch einen gewissen Abstraktionsgrad die Nutzung von Tensorflow. Mit Keras können verschiedenste Machine Learning Modelle einfach trainiert werden.

Jupyter Notebooks

Jupyter Notebooks sind Dateien, die aus einzelnen Text- und Codezellen bestehen können. Die Codezellen können einzeln ausgeführt werden, die Textzellen können zur Dokumentation verwendet werden.

Google Colab

Google Colab [Col] erlaubt das Ausführen von Python Code in Form von sogenannten Colab Notebooks. Colab Notebooks sind Jupyter Notebooks, die von Google gehostet werden. Google Colab erlaubt das Ausführen von Code über die Cloud, sodass keine besondere Hardware für das Ausführen, wie zum Beispiel das Trainieren eines Machine Learning Models, benötigt wird. Mit einem Upgrade auf Colab Pro wird die zur Verfügung gestellte Rechenleistung erhöht und hat in diesem Projekt insbesondere durch die Nutzung einer GPU zu einer signifikanten Verkürzung der Trainingszeit geführt.

5.2 Architektur

In Abbildung 5.1 wird die grobe Architektur in Form von Packages sowie das Frontend dargestellt. In den folgenden Abschnitten wird detaillierter auf die Einzelteile der Architektur eingegangen.

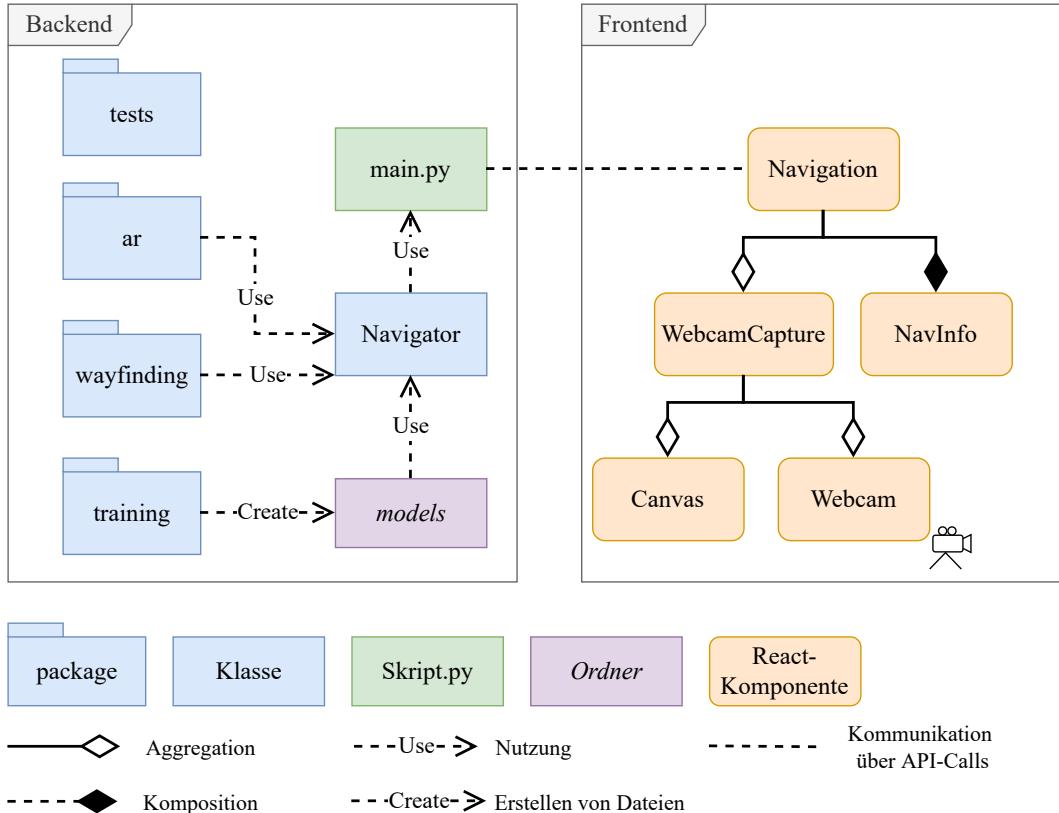


Abbildung 5.1: Grobe Architektur der Anwendung

Die Anwendung besteht aus Frontend und Backend. Das Navigation-Modul stößt die Navigation an und kommuniziert über das `main`-Skript mit dem Backend-Server, welcher die Anfragen des Clients (Frontend) annimmt. Das Frontend schickt Anfragen für die Navigation und AR an den Server. Antworten auf die Navigationsanfrage werden mit der Komponente `NavInfo` angezeigt, welche die berechneten Navigationsinformationen in Schriftform und mit Pfeilen anzeigt. Mit der Komponente `WebcamCapture` werden Bilder aufgenommen und die AR-Informationen in der `Canvas`-Komponente gezeichnet, die direkt über der `Webcam`-Komponente liegt und somit ein deckungsgleiches Overlay bildet.

Im Backend werden je nach Anfrage des Clients verschiedene Methoden der `Navigator`-Klasse aufgerufen. Der `Navigator` ist zuständig, sich die letzte Position und den zuletzt

berechneten Pfad zu merken, damit die Validierung der Position sowie die Positionen der Marker in dem AR-Modul berechnet werden können. Zudem enthält er die bereits initialisierten Modelle, die für diese Berechnungen nötig sind.

Die Berechnung der AR-Marker und der Indoor-Navigation erfolgen getrennt voneinander, da die AR-Berechnung aufgrund der verschiedenen Echtzeit-Anforderungen häufiger durchgeführt werden muss als die Wegfindung. Die Wegfindung kann nur diskrete Positionen ermitteln. Bei AR ist eine schnellere Aktualisierung der Marker erforderlich. Die Berechnung der AR Marker findet im AR-Package anhand des vom Frontend übermittelten Bildes sowie des zuletzt ermittelten Pfades statt. Die Wegfindung findet ebenfalls über ein Bild aus dem Frontend im Wayfinding-Package statt.

Daneben existiert ein Test-Package zum Testen der Komponenten.

Im Training-Package findet sich der Code zum Trainieren mehrerer Modelle. Ebenfalls befinden sich dort alle Trainings- und Testdaten für das Training, genauso wie verschiedene Skripte, die Daten verschieden vorzubereiten für verschiedene Trainingsprozesse. Die trainierten Modelle werden in dem Ordner *models* abgespeichert und können von dort aus aufgerufen werden.

5.3 Interne Darstellung der FH

Im Folgenden wird gezeigt, wie die FH Wedel für dieses Projekt innerhalb des Programms repräsentiert wird. Als Grundlage wurde der offizielle Lageplan der FH Wedel verwendet, dieser wies allerdings einige Ungenauigkeiten auf. So waren zum Beispiel Türen in Gängen versetzt oder einige Türen nicht mehr benutzbar, weswegen die interne Darstellung marginal von dem Lageplan in Abbildung 5.2 abweicht.

Der kleine Gebäudeteil (Gebäude G), in dem sich auch das Audimax befindet, wurde nicht in diesem Prototypen der Indoor Navigation miteinbezogen. Eine größere Darstellung der Abbildung ist in Anhang in Abbildung 7.1 zu sehen.

Rasterung der Fachhochschule Wedel

Die interne Darstellung der FH wird durch ein Raster erzeugt, wie in Grafik 5.3 zu sehen. Dies ist ein Ausschnitt aus dem Lageplan, in dem die Rasterung des Gebäudeteils E im 1. Stock (E1) eingezeichnet ist. Die gesamte Rasterung ist in Anhang in Abbildung 7.1 zu sehen. Die Zahlen von 0 bis 9 bezeichnen die Positionen innerhalb der Flure des

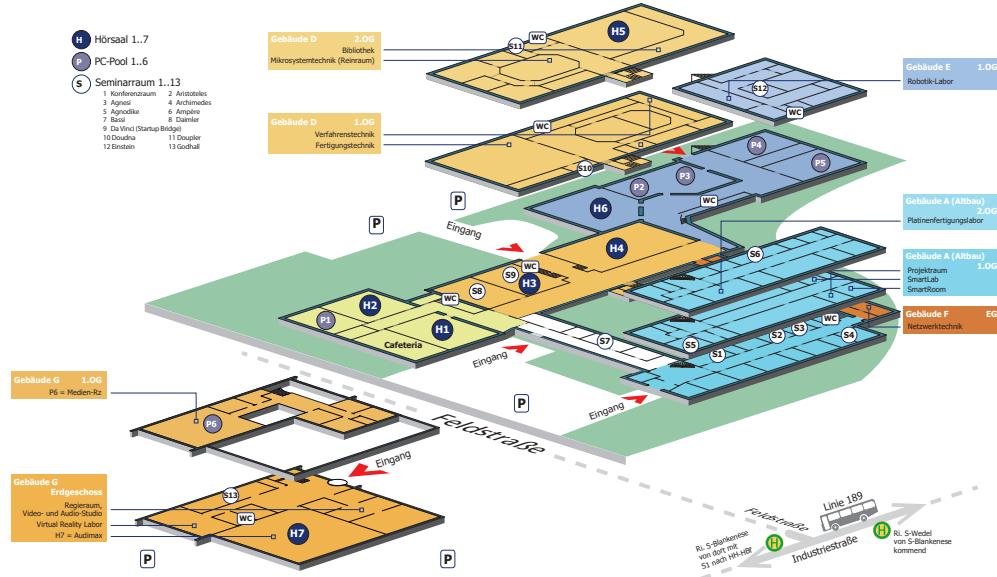


Abbildung 5.2: Lageplan der FH Wedel

Stockwerks. Die Räume haben die Bezeichnung E1.<Raumnummer>. In der oberen rechten Ecke stehen die Bezeichnungen für die möglichen Richtungen, in die ein Gang und somit ein User ausgerichtet sein kann. Diese werden nachfolgend als Blickrichtung bezeichnet und durch eine Zahl von 1 bis 4 kodiert. Sie gelten global für das gesamte Gebäude der FH.

Alle Räume können Ziel der Navigation sein. Der Wegfindungsalgorithmus funktioniert auch, wenn das Navigationsziel eine Position innerhalb eines Gangs ist. Diese Option ist für NutzerInnen aber nicht vorgesehen und deswegen über das Frontend nicht verfügbar. Die Positionen sind daher nur Knoten in einem Suchbaum, aber nie Blätter. Räume können umgekehrt nie Knoten sein.

Anhand des Ausschnitts des Lageplans in Abbildung 5.3 lassen sich einige Ungenauigkeiten erkennen: Die Türen der Büroräume E1.10 bis E1.13, E1.06 und E1.08 sind nicht eingezeichnet, nichtsdestotrotz existieren sie und sind per Navigation erreichbar.

Darstellung als Adjazenzliste

Ein gängiger Ansatz zur Repräsentation von solchen Topologien sind Adjazenzlisten. Hierbei ist jede Position und jeder Raum ein Eintrag einer Liste. Ein Listenelement steht für eine Position und enthält alle von dort aus erreichbaren Knoten. Die Adjazenzliste für diesen Bildausschnitt ist in 5.1 zu sehen. Über die Methode `set_neighbors` werden der

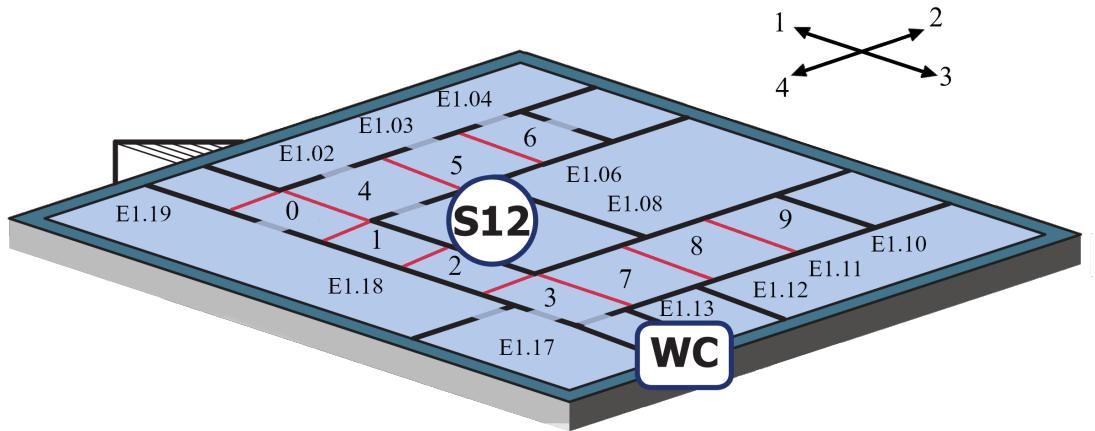


Abbildung 5.3: Ausschnitt aus der internen Darstellung der FH Wedel

Reihe nach die erreichbaren Positionen oder Räume angegeben. Die Reihenfolge ergibt sich aus den Blickrichtungen. Der erste Parameter gibt immer an, welche Nachbarn sich in Blickrichtung 1 befinden, der zweite Parameter, welche Nachbarn sich in Blickrichtung 2 befinden und so weiter.

Eine Position, die ein S enthält, steht für eine Treppe. Die Position E1S03 steht für den dritten Abschnitt einer Treppe, die in die erste Etage des Gebäudeteils E führt. '–' steht dafür, dass in eine Richtung keine Nachbarn existieren. Gibt es mehrere Nachbarn, werden diese als Array übergeben. Intern werden alle Parameter in ein Array umgewandelt. Positionen im Flur werden mit <Gebäudeteil><Stockwerk><Positionsnummer> bezeichnet. Die Positionsnummer ist in Grafik 5.3 ersichtlich, verzichtet jedoch der Übersichtlichkeit halber auf das Präfix E1.

`e1_nodes` ist Teil der Adjazenzliste für die gesamte Repräsentation der Fachhochschule. Für jedes Gebäudeteil und Stockwerk gibt es eine solche Liste, die zu einer einzigen Liste konkateniert wird. Knoten in dieser Liste sind tatsächliche Knotenobjekte, die Nachbarn werden aber zunächst in der String-Repräsentation gespeichert und über das Attribut `name` referenziert.

```

1 e1_nodes[0].set_neighbors('E1S03', 'E104', 'E101', ['E1.18', 'E1
   .19'])
2 e1_nodes[1].set_neighbors('E100', '--', 'E102', '--')
3 e1_nodes[2].set_neighbors('E101', '--', 'E103', '--')
4 e1_nodes[3].set_neighbors('E102', 'E007', ['E1.14', 'E1.15', 'E1
   .16'], 'E1.17')
5 e1_nodes[4].set_neighbors('E1.02', 'E105', 'E1.07', 'E100')
6 e1_nodes[5].set_neighbors('E1.03', 'E106', '--', 'E104')
7 e1_nodes[6].set_neighbors('E1.04', 'E1.05', 'E1.06', 'E105')
8 e1_nodes[7].set_neighbors('--', 'E108', 'E1.13', 'E103')
9 e1_nodes[8].set_neighbors('--', 'E109', 'E1.12', 'E107')
10 e1_nodes[9].set_neighbors('E1.08', 'E1.09', ['E1.10', 'E1.11'], '
    E108')
11 e1_nodes[10].set_neighbors('E1S01', '--', '--', 'E018')
12 e1_nodes[11].set_neighbors('--', '--', 'E1S00', 'E1S02')
13 e1_nodes[12].set_neighbors('--', 'E1S01', 'E1S03', '--')
14 e1_nodes[13].set_neighbors('E1S02', '--', 'E100', 'E108')

```

Listing 5.1: Adjazenzliste des Gebäudeteils E im 1. Stock

5.4 Kommunikationsfluss

Es gibt in der Kommunikation von IndoorNav drei verschiedene Anfragen, die der Client stellen kann: die initiale Navigation (Listing 5.2), die reguläre Navigation (Listing 5.3) und die AR-Anfrage (Listing 5.4). Frontend und Backend kommunizieren über das JSON-Format. Die Anfragen inklusive der JSON-Struktur werden im Folgenden betrachtet.

Initiale Navigation

Die initiale Navigation nimmt mehrere Bilder und das Ziel der Navigation entgegen (Zeile 3 und 4). Anschließend wird die initiale Standortermittlung durchgeführt. Das geschieht mit fünf vom Client übermittelten Bildern (Zeile 5). Bei jedem Bild wird einzeln der Standort inklusive der Blickrichtung ermittelt und anschließend der Median gebildet, um eine höhere Genauigkeit zu erlangen. Dafür wird das Machine Learning Model der Standortermittlung `navigator.localizer` benötigt.

```

1 @app.route('/initial_navigation', methods=['POST'])
2 def initial_navigation():
3     uploaded_files = request.files.getlist('images')
4     navigator.destination = request.form.get('destination')
5     location = initial_localization(navigator.localizer,
6         uploaded_files)
7     navigator.current_pos = location
8
9     return navigator.do_navigation(is_init_loc=True, destination=
10        navigator.destination, uploaded_files=uploaded_files), 201

```

Listing 5.2: Route der initialen Navigation

Anschließend wird die ermittelte Position in dem Navigator-Objekt gespeichert und somit für den Zeitraum der Navigation persistiert (Zeile 6). In Zeile 8 wird die Navigationsfunktion über den Navigator aufgerufen. Dabei dient der erste Parameter `is_init_loc` als Angabe, ob die Funktion ohne Validierung aufgerufen werden kann oder nicht.

Reguläre Navigation

Die reguläre Navigation findet nach der initialen Navigation statt. Hierbei wird aus nur einem Bild ein Standort und eine Blickrichtung ermittelt. Anschließend wird der Pfad berechnet (Zeile 4) und die Position validiert, das heißt, es wird geprüft, ob die neue Position realistisch zu Fuß erreicht werden kann. Ist das nicht der Fall, kann die Berechnung verworfen werden. Somit wird die Genauigkeit der Standortermittlung weiter erhöht.

```

1 @app.route('/navigation', methods=['POST'])
2 def navigation():
3     uploaded_files = request.files.getlist('images')
4     return navigator.do_navigation(False, navigator.destination,
5         uploaded_files), 201

```

Listing 5.3: Route der regulären Navigation

AR

Nachdem der Pfad mindestens ein Mal ermittelt wurde, kann das Augmented Reality Modul verwendet werden. Dafür schickt der Client eine Anfrage an die Route `ar`. In Zeile 5 wird die Berechnung der Markierung aufgerufen. Dafür wird das Segmentierungsmodell

benötigt, welches in dem Navigator-Objekt bereits initialisiert wurde und über `navigator.segmenter` als Parameter in die Methode `calc_markers` gegeben wird. Außerdem benötigt die Marker-Berechnung den zuletzt berechneten Pfad sowie die Position und eine Referenz zu dem Bild, für das die AR-Informationen berechnet werden sollen.

```

1 @app.route('/ar', methods=['POST'])
2 def ar():
3     uploaded_file = request.files.getlist('image')
4     position = int(navigator.current_pos[-1:])
5     return calc_markers(navigator.segmenter, navigator.
6         current_path, position, navigator.destination, uploaded_file),
7         201

```

Listing 5.4: Route der Augmented Reality-Anfrage

Kommunikationsfluss

Abbildung 5.4 zeigt den Kommunikationsfluss zwischen den Teilanwendungen. Das Frontend startet die Navigation mit dem Aufruf der Route `initial_navigation`. Dafür werden fünf Bilder aufgenommen. Diese Bilder enthalten, sofern NutzerInnen die angezeigte Anweisung befolgen, die Kamera leicht zu schwenken, leicht unterschiedliche Bildausschnitte der selben Position und Blickrichtung. Diese Bilder werden mit der Anfrage an das Backend übermittelt. Dort wird daraufhin die initiale Navigation durchgeführt, bestehend aus fünf Standortermittlungen mit allen gelieferten Bildern, der Berechnung des Medians, sowie der Wegfindung, und die Antwort an den Client gesendet. Im Frontend wird die Komponente `NavInfo` neu gerendert, da sich ihr interner Zustand ändert.

Da es nun einen zuletzt berechneten Pfad gibt, kann der Client auch eine Anfrage an die Route `ar` schicken beziehungsweise das Backend entgegennehmen. Dafür wird ebenfalls ein einzelnes Bild aufgenommen und an das Backend geschickt. Dort werden Position und entsprechende Richtung berechnet, die in Form von Pfeilen und anderen Icons im Frontend auf die Komponente `Canvas` gezeichnet werden. Das Frontend schickt kontinuierlich `ar`-Anfragen an das Backend. Sobald eine neue Antwort für diesen Request kommt, werden die im Frontend gehaltenen Daten aktualisiert, wodurch die Komponente neu gerendert wird.

5 Realisierung

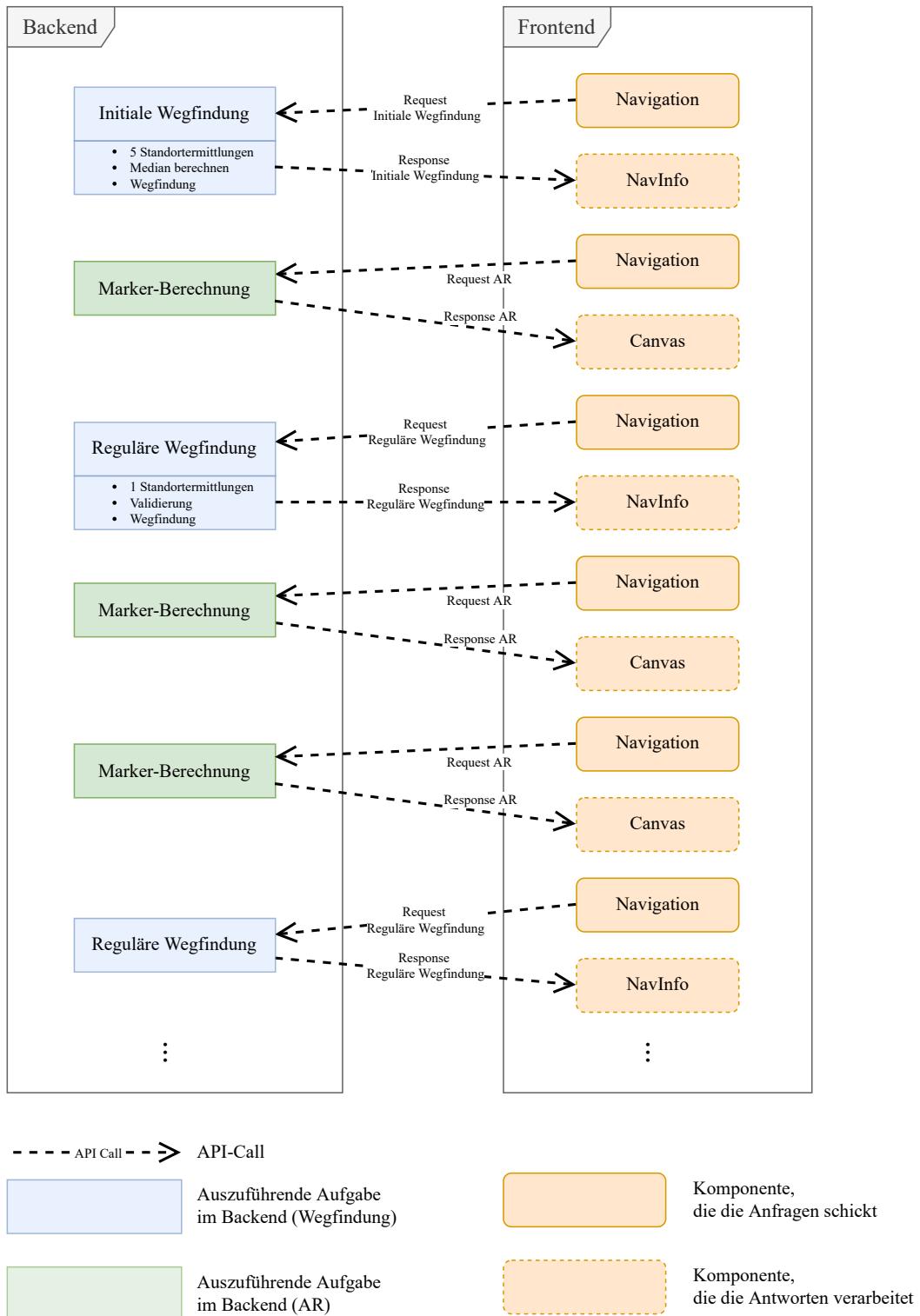


Abbildung 5.4: Kommunikationsfluss

Initiale Navigation

Die Payload des Datenpakets, welches der Client schickt, ist im JSON-Format und ist in Listing 5.5 dargestellt.

```

1 {
2     "images": (binary),
3     "images": (binary),
4     "images": (binary),
5     "images": (binary),
6     "images": (binary),
7     "destination": "A2.06"
8 }
```

Listing 5.5: Anfrage der Initialen Navigation vom Client

Das Feld "images" beinhaltet je ein Bild, kommt aber fünf mal vor, sodass alle Bilder über den Bezeichner "images" ausgelesen werden können. Diese werden als URL kodiert aufgenommen, vorher jedoch in ein File konvertiert und über ein Formular im Binärformat übertragen, wie das in Listing 5.5 in Zeile 2 bis 6 zu sehen ist.

Außerdem wird das Ziel in der Form <Gebäudeteil><Stockwerk>.<Raumnummer> als String übermittelt, also zum Beispiel "A2.06", wie in Zeile 7 zu sehen.

Der Server sendet ein Paket mit den Feldern "dist2goal", "directions", "positions", "dist_next_turn", "dir_next_turn" und "pos_dir" als Antwort, wobei nur die Felder "dir_next_turn" und "pos_dir" notwendig sind. Die anderen Felder werden zu Debugzwecken mitgeschickt, um im Frontend genauer die Berechnungen des Backends nachvollziehen zu können. In Listing 5.6 sind Beispielwerte für die Belegung zu sehen.

```

1 {
2     "dist2goal": 17.5
3     "directions": [2, 2, 2, 1, 4],
4     "positions": ['E103', 'E104', 'E105', 'E106', 'E111', 'E1
5 .08],
6     "dist_next_turn": 7.5,
7     "dir_next_turn": 'left',
8     "pos_dir": 'E1032',
```

Listing 5.6: Antwort der Initialen Navigation vom Server

Der gesamte Pfad setzt sich aus `directions` und `positions` zusammen. `directions` gibt die Richtungen an, die ein/e NutzerIn von einer Position zur nächsten gehen muss, um dorthin zu gelangen. `positions` enthält die Namen der Positionen, die aus den Knoten des Pfads extrahiert wurden. `dist_next_turn` gibt an, wann ein/e NutzerIn abbiegen muss und `dir_next_turn` gibt die entsprechende Richtung an. `pos_dir` enthält die ermittelte Position, inklusive der Blickrichtung, aus der die gesendeten Bilder aufgenommen wurden. `dist2goal` gibt die Entfernung zum Ziel in Metern an.

Normale Navigation

Bei der regulären Navigation sieht das Datenpaket noch einfacher aus. Hier wird lediglich ein Bild geschickt, die Antwort des Servers sieht aus wie bei der initialen Navigation.

AR

Auch bei der AR-Anfrage schickt der Client nur ein Bild, wie bei der normalen Navigation. Als Antwort erhält der Client ein Datenpaket, wie in Listing 5.7 mit Beispielwerten dargestellt.

```

1 {
2     "up_down_dir": "up",
3     "up_down_pos": [[120, 160], [160, 160], [220, 160], [320,
4         160]],
5     "left_right_dir": "left",
6     "left_right_pos": [100, 160]}

```

Listing 5.7: Antwort der AR-Anfrage vom Server

Das Feld `"up_down_dir"` enthält entweder die Information `"up"` oder `"down"`, und sagt aus, ob Pfeile auf dem Boden nach oben oder unten gezeichnet werden sollen. `"up"` bedeutet, dass der/die NutzerIn in die richtige Richtung geht, `"down"` bedeutet, in die entgegengesetzte Richtung gehen zu müssen. `"up_down_pos"` enthält die Positionen, an denen Pfeile gezeichnet werden sollen als Koordinate (x, y). Dieses werden als Array übermittelt, die Koordinaten selbst sind ebenfalls als Array mit zwei Elementen dargestellt. Die Länge des Arrays `"up_down_pos"` ist variabel und hängt unter Anderem von den Gegebenheiten des Bildes und davon ab, ob sich das Navigationsziel in der Nähe befindet. `"left_right_dir"` sagt aus, ob die nächste Abbiegung, egal ob im Gang oder zum Zielraum selbst, links oder rechts ist, und `"left_right_pos"` gibt die Koordinate an, an der der Pfeil im Bild gezeichnet werden müsste. Mit diesem Datenpaket werden

4 Pfeile gezeichnet werden, die den User geradeaus leiten und ein weiterer Pfeil, der an der entsprechenden Position nach links zeigt.

5.5 Indoor Navigation

In diesem Abschnitt wird die Realisierung der Teilanwendung Indoor Navigation beschrieben. Diese besteht zum einen aus der Entwicklung der Software, zum anderen aus manuellen Tätigkeiten zur Datenerhebung.

Zur Übersicht wird als erstes ein Klassendiagramm vorgestellt und erläutert und anschließend auf die verwendeten Trainingsdaten und die Datenerhebung eingegangen. Das Training mit diesen Daten sowie der Aufbau der Modelle wird in den nachfolgenden Unterpunkten beschrieben. Abschließend wird die Wegfindung erörtert.

5.5.1 Softwaredesign

Dieser Teil der Anwendung nutzt zwei Packages namens `wayfinding` und `model`, sowie das `main`-Skript und die `Navigator`-Klasse. Im Folgenden werden alle Klassen und Skripte beschrieben und ihr Zusammenspiel im Detail erklärt. Im `model`-Package werden für die Navigation nicht benötigte Teile ausgespart, ebenso wie Teile, die nur für das Training benötigt werden. Die Hauptfunktionalitäten der einzelnen Klassen und Skripte sowie die Zusammenhänge untereinander, sind in dem Klassendiagramm 5.5 zu sehen.

main.py

Dies ist eine Python-Skriptdatei, also keine Klasse. Sie fungiert als Einstiegspunkt im Backend und ist somit nicht Teil eines Packages. Sie nimmt Anfragen aus dem Frontend entgegen und leitet anhand derer die entsprechenden Berechnungen ein. Das ist zum einen die Berechnung der Position mit dem übertragenen Bild, worauf die Berechnung für die Navigation im Navigator erfolgt, zum anderen die Berechnung der Marker und der Markerpositionen. Dies ist Teil der AR-Teilanwendung.

Navigator

Eine Navigator-Instanz enthält alle für die Navigation und Validierung wichtigen Daten. Das sind die Machine Learning-Modelle für die Lokalisierung (`localizer`) und Segmentierung (`segmenter`).

5 Realisierung

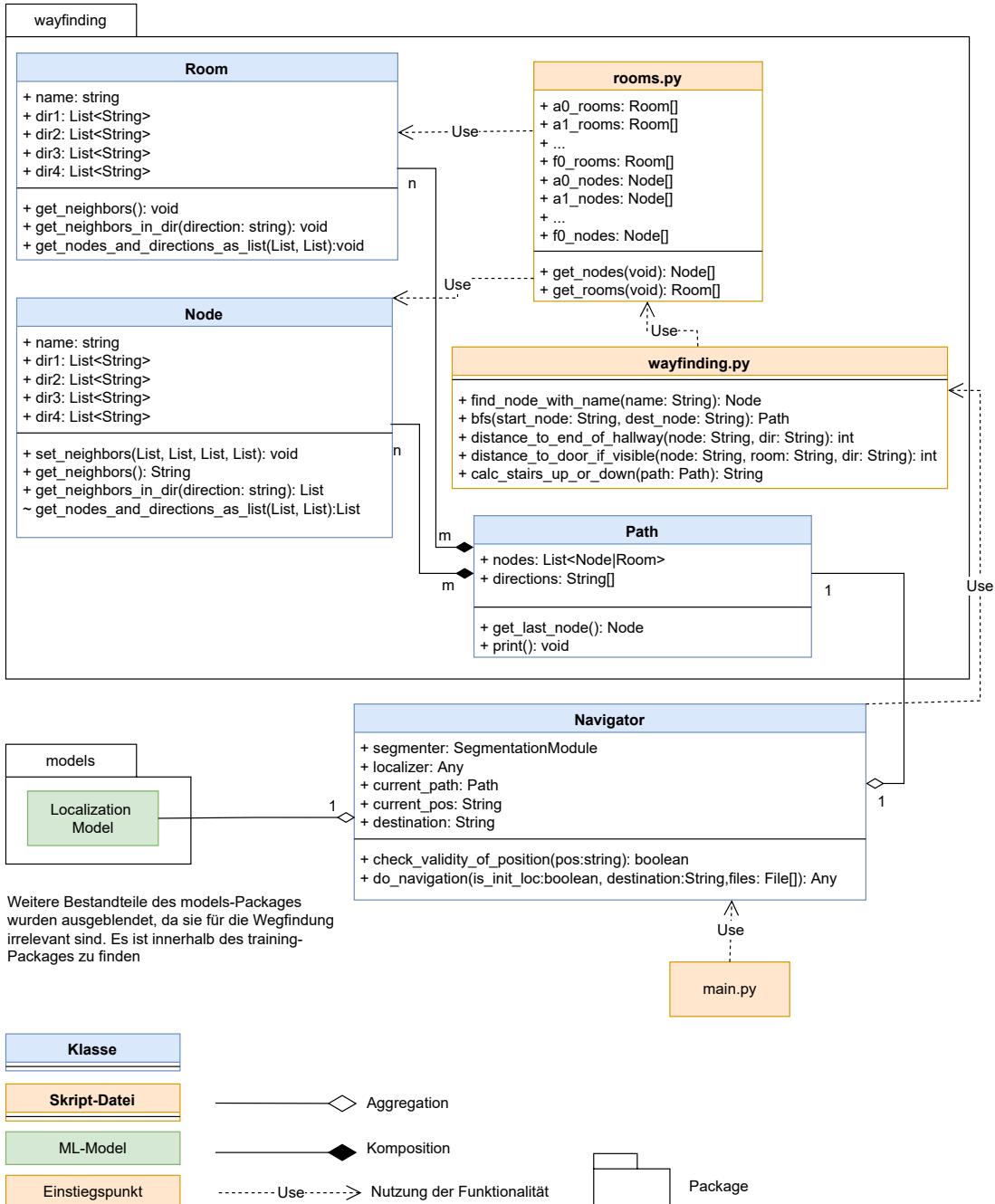


Abbildung 5.5: Klassendiagramm Indoor Navigation

Die AR-Berechnung benötigt einen Pfad, der aus der Berechnung der Indoor Navigation bestimmt wird, welcher in dem Feld `current_path` gespeichert wird. Da die beiden Komponenten aufgrund der verschiedenen Echtzeit-Anforderungen getrennt voneinander funktionieren, muss dieser Pfad im Navigator-Objekt persistiert werden.

Die Validierung einer neu ermittelten Position benötigt die zuletzt ermittelte Position, welche in `current_pos` persistiert wird und die Wegfindung benötigt einen Zielknoten, der ebenfalls über den Zeitraum der Navigation persistiert wird.

Die Navigator-Funktionalität wird durch das `main`-Skript aufgerufen und führt die angeforderte Navigation mit `do_navigation` aus. Hier wird unterschieden in die initiale Navigation und die reguläre Navigation. Die initiale Navigation ermittelt lediglich einen Standort und eine Blickrichtung, während die reguläre Navigation auch den Standort mit `check_validity_of_position` validiert. Sollte das Ergebnis der Standortermittlung unter einem bestimmten Schwellwert liegen, wird die zuletzt bekannte Position verwendet.

Die Wegfindung wird über das Skript `wayfinding` durchgeführt.

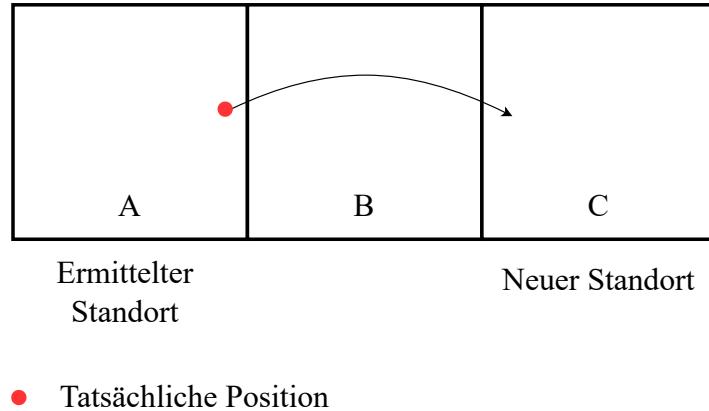


Abbildung 5.6: Positionsvalidierung

Der Standort ist valide, wenn die Distanz zwischen der neu ermittelten und der letzten Position nicht mehr als zwei Positionen ist, was maximal $3 \cdot 2,5 = 7,5$ Metern entspricht. Das ist weit mehr als ein Mensch gehend in einer halben Sekunde zurückliegt, deckt aber leichte Ungenauigkeiten durch Netzwerklatenzen oder die Standortlokalisierung ab. In bestimmten Konstellationen ist die Distanz zwischen Positionen, die durch eine weitere Position verbunden sind, jedoch nur etwas mehr als 2,5 Meter. Ein solches Szenario ist in Abbildung 5.6 dargestellt. Der rote Punkt stellt die tatsächliche Position dar, während der ermittelte Standort die Position A ist. Geht der/die NutzerIn nun ein schnelles Tempo oder kommen Netzwerklatenzen dazu, ist es ihm/ihr möglich, Position B zu

durchqueren, bevor eine erneute Standortermittlung ausgelöst wird, die als aktuellen Standort Position C ermittelt. Ist die Position aber weiter entfernt als diese maximal 7,5 Meter, wird von einer falschen Standortermittlung ausgegangen.

Neben dem Pfad selbst, berechnet der Navigator die Strecke und die Richtung bis zur nächsten Abbiegen und liefert diese Informationen an den Aufrufer, also das `main`-Skript, zurück, welcher die Information direkt an das Frontend weitergibt.

Room

Ein `Room` ist immer das Ziel einer Navigation und repräsentiert alle Räume der Fachhochschule, wie etwa Hörsäle, Rechenzentren, Lernabteile, Seminarräume, Labore oder WCs. Ein Raum hat eine Raumbezeichnung, angegeben durch das Attribut `name`, welches sich aus dem Gebäudeteil inklusive Stockwerk und der Raumnummer zusammensetzt. Die Namen von Räumen, wie etwa Aristoteles oder Archimedes für Seminarräume, Hörsaal 3 oder Seminarraum 7, werden nicht für die Navigation verwendet, sondern müssen vorher in die entsprechende Bezeichnung umgewandelt werden. Ein `Room` hat keine Nachbarn, das heißt, `dir1`, `dir2`, `dir3` und `dir4` enthalten niemals Einträge von Nachbarknoten, da dies das Ziel der Navigation ist, und somit von dort aus keine weitere Wegfindung betrieben werden muss. Entsprechend liefern alle Methoden `void` zurück. Diese Methoden sind jedoch für die Nutzung des Wegfindungsalgorithmus notwendig, sodass `Nodes` und `Rooms` gleich behandelt werden können.

Node

Ein `Node` stellt eine Position innerhalb eines Ganges und somit einen Knoten im Suchbaum für die Wegfindung dar. Entsprechend der Blickrichtungen sind in `dir1` bis `dir4` die Nachbarknoten in einem Array eingetragen. Das Attribut `name` enthält den global eindeutigen Namen der Position bestehend aus Gebäudeteil inklusive Stockwerk sowie einer Position im Flur. Die Klasse `Node` enthält verschiedene Methoden, die jeweils einen oder mehrere Nachbarn zurückgeben, je nach Angabe der Richtung.

Path

Eine `Path`-Instanz repräsentiert den Pfad, der mit dem Breitensuche-Algorithmus berechnet wurde. Er setzt sich aus dem Attribut `nodes`, einem Array aus `Node`- oder `Room`-Instanzen, und `directions`, einem Array, welches die Blickrichtungen durch Integer von 1 bis 4 repräsentiert, zusammen. Ein Room entspricht einem Blatt und ist somit das Ende eines Pfades und immer am Ende der Liste zu finden. Alle anderen Listenelemente

sind vom Typ `Node`. `directions` gibt an, über welche Richtung zwei Knoten miteinander verbunden sind und gibt somit an, wann NutzerInnen abbiegen müssen.

Diese Klasse kann den letzten Knoten des Pfads ermitteln, was dem Ziel der Navigation entspricht sowie eine Konsolenausgabe des zugrundeliegenden Pfades machen.

rooms.py

Diese Datei enthält im Wesentlichen die Adjazenzliste, die die interne Struktur der FH Wedel repräsentiert. Sie enthält die Räume pro Gebäudeteil und Stockwerk, sowie die Positionen innerhalb der Flure, welche ebenfalls die Namen der Nachbarn enthalten. Mit den beiden Methoden werden Räume und Positionen (`Room`- und `Node`-Objekte) jeweils zu einer Liste konkatiniert.

wayfinding.py

`Wayfinding` ist ebenfalls ein Skript und bietet Methoden zur Wegfindung an, ist aber keine Klasse, denn es besitzt beziehungsweise benötigt keine eigenen Attribute. Die Funktion `find_node_with_name` findet einen Knoten (`Node` oder `Room`) anhand des Namens, was die Grundlage zur Verknüpfung der einzelnen Elemente der Adjazenzliste ist.

Die Funktion `bfs` ist das Herzstück dieses Moduls und enthält die Implementation der Breitensuche. Weitere Funktionen zur Berechnung der Distanz zu einem Zielraum (in `distance_to_door_if_visible` implementiert) oder der nächsten Abbiegung (in `distance_to_end_of_hallway` implementiert) finden sich ebenfalls in diesem Skript. Des Weiteren muss ermittelt werden können, ob ein Etagenwechsel erfolgt und wenn ja, in welche Richtung, also aufwärts oder abwärts. Dies kann mit der Funktion `calc_stairs_up_or_down` getan werden. Die Angabe von Zwischenzielen ist in dieser Implementation nicht vorgesehen.

Localization Model

Dieses Objekt besteht eigentlich aus mehreren Dateien, fungiert aber als ein Model und wird daher hier als ein Objekt dargestellt. Das Model wird im Navigator zur Positionsermittlung verwendet und wurde trainiert, eine Klassifizierung der Position inklusive der Blickrichtung anhand eines Bildes durchzuführen. Insgesamt gibt es über 350 Kategorien, die klassifiziert werden können.

5.5.2 Daten und Datenerhebung

Das Model wird mit Bilddaten trainiert, die bei ausreichend guter Performance des Models möglichst klein sein sollen. Da nicht vorhersehbar war, bei welcher Auflösung das der Fall ist, wurde zur Aufnahme FullHD gewählt, das Bildmaterial wurde also zunächst in einer Auflösung von 1920×1080 aufgenommen. Außerdem wurden die Farbkanäle beibehalten, da unklar war, ob diese für eine bessere Performance nötig sind.

Das Aufnehmen der Bilder ist eine manuelle und zeitintensive Aufgabe, daher sollte diese möglichst effizient ablaufen. Statt einzelner Bilder wurden deswegen Videos von circa 30 Sekunden Länge aufgezeichnet und per Skript Einzelbilder generiert. Die Videos sind in Ordner in drei Ebenen sortiert.

1. Gebäudeteil und Stockwerk, also A0, A1, A2 usw. bis F0
2. Position innerhalb eines Gangs, zum Beispiel von 00 bis 11
3. Blickrichtung

Auf der dritten Ebene befinden sich die Videos, welche jeweils mit der Blickrichtung benannt wurde, in der das Video aufgenommen wurde. Teilweise wurden Videos zu verschiedenen Zeitpunkten aufgenommen¹, sodass mehrere Videos in die gleiche Blickrichtung in einem Ordner existieren. Diese werden durchnummeriert.

Die Funktion `calc_num_frames_to_skip` (Zeile 1 in Listing 5.8) berechnet den Abstand der einzelnen Bilder in Frames, also wie viele Frames zwischen den Bildern liegen sollen, die für das Training verwendet werden. Dafür werden in einem gegebenen Ordner die Frames aller Videos pro Blickrichtung addiert (Zeile 9) und anschließend durch die gewünschte Anzahl der Bilder `NUM_IMAGES_PER_CAT` geteilt, wie in Zeile 13 zu sehen ist.

Die Funktion `create_frames` nimmt die zuvor berechnete Anzahl an Frames, die übersprungen werden soll, entgegen, wandelt die entsprechenden übrigen Frames in Bilder um und speichert sie (Zeile 33 und 34 in Codeausschnitt 5.8). Die Anzahl der Einzelbilder sind für alle Kategorien gleich, abgesehen von kleinen Abweichungen, da die Frames diskrete Werte sind, die zu überspringenden Frames aber nicht.

Die Frames müssen übersprungen werden, da über die Videos verteilt unterschiedliche Bildausschnitte zu sehen sind. Würden nicht gleichmäßig über das gesamten Video

¹Ein Beispiel dafür ist Gang D0, der aufgrund der Fenster einmal bei Sonnenschein und einmal bei bewölktem Wetter aufgenommen wurde.

Bilder erzeugt werden, würde nur ein sehr ähnlicher Bildausschnitt am Anfang des Videos für das Training verwendet werden, andere Ausschnitte am Ende des Videos würden aber gar nicht verwertet werden.

```

1 def calc_num_frames_to_skip(path):
2     skip_frames = [0.0, 0.0, 0.0, 0.0]
3     # findet alle Dateien in dem Ordner path
4     for root, _, files in os.walk(path):
5         for file in files:
6             video = cv2.VideoCapture(os.path.join(root, file))
7             num_frames = video.get(cv2.CAP_PROP_FRAME_COUNT)
8             idx = int(file[0]) - 1
9             skip_frames[idx] += float(num_frames)
10
11    # Berechnet, wie viele Frames zwischen zwei Bildern liegen
12    for i in range(len(skip_frames)):
13        skip_frames[i] = skip_frames[i] / NUM_IMAGES_PER_CAT
14
15    return skip_frames
16
17 def create_frames(src_path, dest_path, filename):
18     # Read the video from specified path
19     skip_steps_size = calc_num_frames_to_skip(src_path)
20     skip_frames = skip_steps_size[int(filename[0]) - 1]
21     currentframe = 0
22     num_frames_captured = 0
23
24     cam = cv2.VideoCapture(src_path + "\\\" + filename )
25     while(True):
26         # liest neuen Frame
27         ret, frame = cam.read()
28         if ret:
29             # Aufnahme des Bildes
30             if currentframe > (num_frames_captured*skip_frames):
31                 name = '.\\\" + dest_path + "\\\" + filename[0] +
32                         '_f' + str(currentframe) + '.jpg'
33                 frame = cv2.resize(frame,
34                                    (IMG_WIDTH_TRAIN, IMG_HEIGHT_TRAIN))
35                 cv2.imwrite(name, frame)

```

```

36         num_frames_captured += 1
37         currentframe += 1
38     else:
39         break
40
41 # Speicher freigeben, nachdem alle Bilder aufgenommen wurden
42 cam.release()
43 cv2.destroyAllWindows()

```

Listing 5.8: Einzelbilderzeugung

Die Aufnahmen wurden mit der Smartphonekamera des Samsung Galaxy S8² aufgenommen. Während der Aufnahme wurde der Bildausschnitt durch Schwenken der Kamera leicht variiert, wie in Abbildung 5.7 zu sehen ist. Das Schwenken dient auch zu einer genaueren initialen Standortermittlung, da der/die NutzerIn über das Frontend aufgefordert wird, die Kamera etwas zu schwenken. Durch die Rasterung in Positionen im Flur, wie in Abbildung 5.3 beispielhaft zu sehen, wurden circa alle 2,5 Meter in der Mitte einer solchen Position Videos aufgenommen. Es wurde Bildmaterial in alle Richtungen, in die NutzerInnen weitergehen können, aufgenommen, innerhalb der Räume oder Labore nicht.

Die Videos wurden zu zwei verschiedenen Zeitpunkten aufgenommen. Zum ersten Zeitpunkt fand kein Vorlesungs- oder Prüfungsbetrieb in der Hochschule statt, sodass die Flure weitgehend menschenleer waren. Zum zweiten Zeitpunkt fand der Vorlesungsbetrieb wieder statt, sodass auch Aufnahmen mit Menschen für das Training vorhanden sind. Zudem war das Wetterverhältnis zum Zeitpunkt der ersten Aufnahme sonnig, während es beim zweiten bewölkt war. Das macht vor allem in den Gängen D0 und F0 einen großen Unterschied, wie in Bild 5.8 zu sehen ist.

Große Teile beider Gänge sind mit einer Fensterfront ausgestattet. So werden an sonnigen Tagen, wie in Foto 5.8b zu sehen, auf dem Boden Muster gezeichnet, die es an wolkigen Tagen nicht gibt, s. Abbildung 5.8a. Sind solche Variationen in den Trainingsdaten nicht vorhanden, kann das negative Auswirkungen auf die Klassifikation haben. Das gleiche kann passieren, wenn ein Model nur auf Trainingsdaten mit menschenleeren Fluren trainiert, bei der Nutzung selbst dann aber Menschen durch die Gänge laufen. Daher befinden sich auch Personen auf Bild 5.8a.

²Modell SM-G950FD

5 Realisierung

(a) Bildausschnitt links



(b) Bildausschnitt rechts



(c) Bildausschnitt oben



(d) Bildausschnitt unten



Abbildung 5.7: Bildausschnitte aus dem Gang A1, Position 5, Blickrichtung 4



Abbildung 5.8: Gang D0 mit verschiedenen Wetterkonditionen

5.5.3 Training

Das Training wurde mit Keras realisiert, teilweise als Python Skript, teilweise als Jupyter Notebooks in Google Colab, da durch deren Infrastruktur bessere Hardware zur Verfügung gestellt werden kann.

Keras bietet, in Listing 5.9 zu sehen, mit `image_dataset_from_directory` eine Möglichkeit an, Bilddaten über die Ordnerstruktur zu labeln, die in diesem Projekt genutzt wurde. Dies ist mit dem Parameter `labels="inferred"` zu realisieren. Daneben können weitere Angaben zu zum Beispiel Farbe oder Bildgröße gemacht werden. So können einfach gelabelte Trainings- und Testdaten erstellt und für das Training eines Keras Models verwendet werden. Die Erklärungen aller weiteren Parameter kann der Quelle [Tenc] entnommen werden.

Jeder Ordner muss dieselbe Bezeichnung wie die entsprechende Kategorie haben und alle zugehörigen Bilder enthalten. Die Struktur darf nicht geschachtelt sein, das heißt, alle Kategorien müssen innerhalb eines Ordners sein und dürfen nicht durch weitere Ordner in Gebäudeteile oder Stockwerke strukturiert werden.

```

1 ds_train = tf.keras.preprocessing.image_dataset_from_directory(
2     SRC_DIR,
3     labels="inferred",
4     label_mode="categorical",
5     color_mode="grayscale",
6     batch_size=batch_size,
7     image_size=(IMG_HEIGHT_TRAIN, IMG_WIDTH_TRAIN),
8     shuffle=True,
9     seed=123,
10    validation_split=0.1,
11    subset="training",
12 )

```

Listing 5.9: Automatisches Labeling der Trainings- und Testdaten

Eine gängige Aufteilung von Trainings- und Testdaten ist nach [G17, S. 29] ein Verhältnis von 80% Trainingsdaten zu 20% Testdaten. Hier sind jedoch nur begrenzt Daten vorhanden. Das Skript aus 5.8 ist zwar in der Lage, mehr Bilder aus den Videos auszuschneiden, jedoch sind die Bildausschnitte dann zunehmend ähnlicher. In der Arbeit von Björn Barz und Joachim Denzler [BD20] wurde die Auswirkung von Duplikaten und sehr ähnlichen Trainingsdaten untersucht. Sie kamen zu dem Ergebnis, dass Duplikate oder Nahezu-Duplikate im Trainingsdatensatz zu einer schlechteren Generalisierung und somit zu einer schlechteren Performance auf neuen Daten führen. Durch eine zufällige Aufteilung verteilen sich Duplikate oder Nahezu-Duplikate möglicherweise auch auf den Trainings- und Testdatensatz, sodass mit dem Trainingsdatensatz nicht vollständig geprüft werden kann, ob das Model eine gute Generalisierung der Daten gelernt hat. In diesem Projekt sind die Daten sehr ähnlich, auch dann, wenn weiteres Videomaterial gesammelt werden würde. Daher wurden 90% der Daten als Trainingsdatensatz verwendet und die übrigen 10% als Validierungsdatensatz.

Hyperparameter

Hyperparameter sind die Parameter, die von außen für das Model vorgegeben werden. Sie sind nicht zu verwechseln mit jenen, die innerhalb eines Models über das Training angepasst werden. Hyperparameter sind zum Beispiel die Bildgröße, die Aktivierungsfunktion oder die Anzahl der Trainingsdurchläufe.

5 Realisierung

In Tabelle 5.1 sind verschiedene Modelle mit verschiedenen Parameter-Konstellationen aufgelistet, um Hyperparameter zu identifizieren, unter denen ein Model grundsätzlich gut performt. Die Model-Architektur wurde dabei nicht verändert.

Untersuchte Hyperparameter sind:

1. Datensatz: gibt die Größe des Datensatzes an. Der Datensatz ist aufgeteilt in Teil und Gesamt. Gesamt enthält alle Kategorien, Teil enthält nur die Kategorien A1, A2, D1 und D2, um ein schnelleres Ergebnis zu erhalten³.
2. Klassifikation: (Bei mehreren Modellen) gibt an, welche Klassen ein Modell klassifizieren können soll.
3. Bildgröße: gibt die Größe der Bilder an. Das Format ist Hochformat, die Angabe ist Höhe × Breite zu lesen.
4. Farbe: gibt an, ob Farbinformationen (RGB) verwendet wurde oder nicht (Grastufen).
5. Bilder/Kategorie: gibt die Anzahl der Bilder pro Kategorie, inklusive der 10% für das Testen des Models, an.
6. Epochen: gibt die Anzahl der Epochen im Trainingsprozess an, also wie oft der Trainingsalgorithmus den Trainingsdatensatz vollständig verarbeitet. Diese wurden zunächst klein gehalten, da schon bei einer geringen Anzahl an Epochen abgesehen werden konnte, ob ein Model gut performen wird .
7. Genauigkeit und Loss: geben an, wie gut ein Model performt. Beides sind Einheiten, um die Performanz eines Models auf neuen Daten (Testdaten) zu messen. Genauigkeit, (engl. Accuracy, im Folgenden mit Acc. abgekürzt) ist der Anteil richtig klassifizierter Daten. Die Angabe Loss enthält, wie gut die klassifizierten Daten dem Label entsprechen. Mit dem Parameter `categorical_crossentropy` für die Loss-Funktion kann Keras die Abweichung zwischen ermittelten und tatsächlicher Klasse berechnen.

Die Auswirkung der Learningrate wurde nicht untersucht, sondern der Standardwert von 0.001 beibehalten. Beim Training der Transfer Learning-Modelle wurde eine Learningrate von 0,00001 verwendet.

³Diese Versuche wurden lokal auf einem Laptop ohne GPU, sondern nur mit einem Intel(R) Core (TM) i7-8665U mit 4 Kernen ausgeführt.

5 Realisierung

	Datensatz	Bildgröße	Farbe	Bilder pro Kategorie	Epochen	Acc. und Loss
1	Teil	320×180	RGB	50	20	Acc: 0, 5873 Loss: 4, 3937
2	Teil	320×180	Graustufen	50	15	Acc: 0, 5926 Loss: 3, 0087
3	Teil	320×180	Graustufen	100	15	Acc: 0, 8003 Loss: 3, 3121
4	Teil	320×180	Graustufen	100	50	Acc: 0, 8682 Loss: 5, 1363
5	Teil	160×90	Graustufen	100	15	Acc: 0, 9201 Loss: 0, 6259
6	Teil	160×90	Graustufen	100	30	Acc: 0, 0133 Loss: 4, 8881
7	Gesamt	320×180	Graustufen	50	15	Acc: 0, 7562 Loss: 1, 6806
8	Gesamt	320×180	Graustufen	100	15	Acc: 0.7662 Loss: 1.1182
9	Gesamt	160×90	Graustufen	100	15	Acc: 0.7965 Loss: 0.9695

Tabelle 5.1: Training mit verschiedenen Hyperparametern

Tabelleneintrag 1 der Tabelle 5.1 enthält den ersten Trainingsprozess eines Modells. Während des Trainings stieg die Genauigkeit auf 0,9789 an, und der Loss auf 0,1292. Die Genauigkeit auf den Testdaten allerdings liegt, in der Spalte Acc. und Loss angegeben, nur bei 0,5873 beziehungsweise 4,3937. Das Model funktioniert also nur sehr gut auf den Trainingsdaten, auf den Testdaten aber nicht. Dieses Phänomen nennt man Overfitting [G17, S. 27]. Um dieses Problem zu vermeiden, wurden die Attribute der Trainingsdaten verringert. Dafür wurden die Farbinformationen aus den Trainingsbildern entfernt. Dies hat nicht die gewünschte Verbesserung gebracht - auch hier lag Overfitting in einem ähnlichen Maße vor. Es hat aber aufgezeigt, dass die Farbinformationen mit dieser Implementation keinen Vorteil bieten und deswegen verworfen werden kann. Overfitting kann auch durch zu wenige Datensamples entstehen [G17, S. 27]. Daher wurde in Zeile 3 die Anzahl der Bilder pro Kategorie verdoppelt. Dieses Vorgehen resultierte in einer Verbesserung der korrekt erkannten Klassen um fast 21%. In Zeile 4 wurde die Anzahl der Epochen erhöht, was zu einer weiteren Verbesserung von über 6% geführt hat. Allerdings hat sich auch der Loss signifikant erhöht.

In Eintrag 5 wurde die Bildgröße halbiert. Je kleiner das Bild, desto weniger aufwendig der Trainingsprozess. Zudem wird das Bildrauschen verringert und potentiell unnötige Details reduziert, was ebenfalls Overfitting vermeiden kann. Im Vergleich zu dem Model aus Zeile 3 hat sich die Genauigkeit deutlich erhöht, von 80,03% auf 92,01%. Auch der Loss ist deutlich gesunken. Durch eine Erhöhung der Epochen auf 30 ist die Genauigkeit jedoch wieder stark um 0,9068 gesunken. Overfitting kann hier auch ausgeschlossen werden, da die Genauigkeit während des Trainings 0,0128 nicht übersteigt. Dies kann mit einer unglücklichen, zufälligen Initialisierung zusammenhängen.

Ab Zeile 7 wurde der gesamte Datensatz verwendet. Für eine Verkürzung der Trainingszeit wurde die Menge der Bilder pro Kategorie verringert, da mit dem vollständigen Datensatz statt 63 Kategorien über 350 erkannt werden müssen. Die Bildgröße wurde wieder auf 320×180 gesetzt, um eine Vergleichbarkeit zu anderen Models mit nur einem Teil des Datensatzes zu schaffen. Im Vergleich zu Zeile 2 konnte eine Verbesserung von 0,1636 erreicht werden, obwohl die Anzahl der Kategorien deutlich erhöht wurde. Mit der doppelten Anzahl der Bilder, also 100 pro Kategorie, konnte aber kaum eine Verbesserung erzielt werden (Zeile 8) während bei dem Teildatensatz ein deutlicher Anstieg der Genauigkeit zu sehen ist (Zeile 2 und 3). In Zeile 9 wurde die Bildgröße wieder auf beiden Dimensionen halbiert, was im Vergleich zum vorigen Model zu einer leichten Verbesserung von ungefähr 3%, mit 0,7965 aber noch deutlich unter der in Kapitel 3 geforderten Genauigkeit von 95% liegt.

In Tabelle 5.2 wurden mehrere Modelle trainiert. Dafür wurde die Klassifizierung anders realisiert (Zeile 1), indem Gebäudeteil, Stockwerk und Blickrichtung mit einem Model erkannt werden und die Position innerhalb eines Ganges mit einem separaten Model, oder die Datensätze verschieden aufgeteilt werden (Zeile 2 und 3).

Da Positionen innerhalb von Gängen in eine Blickrichtung sehr ähnlich aussehen und sich von Position zu Position nur in den Proportionen des Gebäudes unterscheiden, erzielte das Model 1.1 mit 0,7895 eine recht gute Genauigkeit. Mit dem 2. Model ließ sich die Position mit 0,1104 nur sehr schlecht bestimmen. Eine mögliche Erklärung ist, dass sich Positionen innerhalb eines Gangs zwar ähneln, jedoch nicht der gleichen Kategorie angehören, gleiche Positionen in verschiedenen Gängen sich aber nicht zwangsläufig ähneln und deswegen zu einer unzureichenden Klassifizierung führen.

Die Ergebnisse aus Tabelle 5.1 legen nahe, dass die Reduzierung der zu klassifizierenden Labels eine positive Auswirkung auf die Performance des Models haben kann. Daher

	Klassifizierung	Bildgröße	Farbe	Bilder pro Kategorie	Epochen	Acc. und Loss
1.1	Gebäudeteil, Stockwerk und Blickrichtung	320 × 180	Graustufen	50	15	Acc: 0.7895 Loss: 1.3529
1.2	Position innerhalb des Flures	320 × 180	Graustufen	50	15	Acc: 0.1104 Loss: 2.9468
2.1	A, E, F (183 Klassen)	320 × 180	Graustufen	175	15	Acc: 0.8679 Loss: 0.6311
2.2	B, C, D (172 Klassen)	320 × 180	Graustufen	175	15	Acc: 0.1072 Loss: 4.2268
3.1	A, B, C (108 Klassen)	320 × 180	Graustufen	175	15	Acc: 0.9814 Loss: 0.8830
3.2	D (143 Klassen)	320 × 180	Graustufen	175	15	Acc: 0.9543 Loss: 1.7108
3.3	E, F (104 Klassen)	320 × 180	Graustufen	175	15	Acc: 0.9766 Loss: 1.2075

Tabelle 5.2: Training mehrerer Modelle

5 Realisierung

wurden die Datensätze in Zeile 2 und 3 in ungefähr gleichgroße Datensätze aufgeteilt. Die Anzahl der Bilder wurde auf 175 erhöht, da jedes Model nun wieder rund die Hälfte der Kategorien erkennen muss. Bei der Positionsbestimmung werden alle Modelle ausgewertet. Die Klassifikation mit der höchsten Wahrscheinlichkeit wird als die richtige betrachtet.

Model 2.1, welches die Klassifizierung für alle Kategorien in Gebäudeteil A, E und F übernimmt, performt mit 86,79% richtig klassifizierten Ergebnissen passabel, das Klassifizieren der Kategorien aus Gebäudeteil B, C und D ist aber mit 0,1072 unbrauchbar. Warum das so ist, ist unklar. Das Training wurde mit einer anderen Initialisierung erneut gestartet, brachte jedoch kein besseres Ergebnis hervor (Acc.: 0,1072, Loss: 4,2268). Das kann zwei Ursachen haben: entweder wurde das globale Maximum durch ungünstige Startkonstellationen nicht gefunden oder das globale Maximum liegt bei nur etwa 10,7%.

Im nächsten Versuch wurden drei Modelle mit je rund einem Drittel des Datensatzes trainiert. Hierbei performten die Modelle weit besser als alle Modelle zuvor. Dies kann zum einen durch die Reduzierung der Anzahl der pro Modell zu klassifizierenden Kategorien sowie der Erhöhung der Anzahl der Bilder pro Kategorie röhren. Der Nachteil an dieser Lösung ist jedoch, dass pro Positionsbestimmung drei Auswertungen gemacht werden müssen.

Um den Nachteil der mehrfachen Auswertung pro Positionsbestimmung zu vermeiden, wurden per Transferlearning weitere Modelle trainiert. Hierbei wurde wieder der gesamte Datensatz verwendet, also wieder alle Bilddaten der 355 Kategorien. In Listing 5.10 ist das Training eines solchen Transfer Learning Models zu sehen. Die Datenvorbereitung ist in diesem Codeabschnitt nicht zu sehen, unterscheidet sich allerdings auch nur in einem Punkt von den zuvor trainierten Modellen: das vortrainierte Basismodel benötigt Bilder in RGB.

```

1 base_model = tf.keras.applications.MobileNetV2(input_shape =
2     input_shape, include_top = False, weights = "imagenet")
3
4 model = tf.keras.Sequential([base_model,
5     tf.keras.layers.GlobalAveragePooling2D(),
6     tf.keras.layers.Dropout(0.2),
7     tf.keras.layers.Dense(nClasses_1, activation="softmax") ])
8 base_learning_rate = 0.00001
9 model.compile(optimizer=tf.keras.optimizers.Adam(lr=
10    base_learning_rate), loss=tf.keras.losses.BinaryCrossentropy(
11    from_logits=True), metrics=['accuracy'])
12
13 history = model.fit(ds_train_m1, epochs = 250 , validation_data =
14    ds_validation_m1)

```

Listing 5.10: Transfer Learning

In Zeile 1 wird dieses Basismodel, welches speziell für Bildverarbeitung und somit auch für Bildklassifizierungsaufgaben konzipiert ist, geladen. Es handelt sich hierbei um das MobileNetV2 von Google - ein Neuronales Netz, welches das Augenmerk auf Computer Vision mit mobilen Geräte legt[SHZ⁺18]. Der Parameter `weights` gibt an, welche Gewichte in das Model geladen werden sollen, also im Wesentlichen, welches Model geladen werden soll. In diesem Fall sind das die Gewichte eines bereits auf dem Datensatz ImageNet⁴ trainierten Models. Da der Output Layer überschrieben werden soll, muss der Parameter `include_top` auf `False` gesetzt werden, damit dieser nicht geladen wird.

In Zeile 4 bis 7 wird das gesamte Model zur Positionsermittlung erstellt. Es besteht aus dem Basismodel `base_model` sowie aus drei weiteren Layern: einem Average Pooling Layer, einem Dropout Layer sowie einem Dense Layer. Wie diese Layer funktionieren, wird in 5.5.4 genauer erläutert, da diese Art Layer auch in dem eigens konzipierten CNN Verwendung finden.

Zeile 9 kompiliert das Model, sodass es in Zeile 11 trainiert werden kann.

⁴<https://www.image-net.org/about.php>, Zugriff 24.07.2022

ImageNet ist eine sehr große Datenbank, die gelabelte Bilder enthält und vor allem im Bereich Objekterkennung Verwendung findet.

	Bildgröße	Farbe	Bilder pro Kategorie	Epochen	Acc. und Loss
1	320×180	RGB	175	250	Acc: 0.9911 Loss: $8.5731e - 04$
2	180×90	RGB	175	250	Acc: 0.9841 Loss: 0.0011
3	320×180	RGB	100	250	Acc: 0.9786 Loss: 0.0018

Tabelle 5.3: Ergebnisse Transfer Learning

Insgesamt wurde in 250 Epochen mit der Optimizer-Funktion `Adam` sowie der Loss-Funktion `Cross Entropy` trainiert. Der `Adam`-Algorithmus ist ein Optimizer, der für das Anpassen der Gewichte während des Trainingsprozesses zuständig ist. Die Loss Funktion ist `BinaryCrossEntropy`. Der Parameter `from_logits` gibt an, welche Form die ermittelten Werte haben, die miteinander verglichen werden sollen.

Das Ergebnis des Trainings mit Transfer Learning findet sich in Tabelle 5.3. Es wurden zwei Trainingsdurchgänge mit je 250 Epochen durchgeführt, einmal mit einer Bildgröße von 320×180 , einmal mit einer Bildgröße von 180×90 . Ab circa 240 Epochen pendeln Genauigkeit und Loss leicht um das Maximum (0,9906 – 0,9916 bei Model 1, 0,9830 – 0,9842 bei Model 2), daher wurde eine Anzahl von 250 gewählt, also etwas mehr als 240, damit sichergestellt werden kann, dass sich das Ergebnis ausreichend an das Maximum angenähert hat. Zudem wurde ein weiteres Model mit weniger Bildern trainiert, um zu testen, ob zu viele beziehungsweise zu ähnliche Trainingsdaten zu einer Verfälschung der Ergebnisse geführt haben könnten. Da mit 100 Bildern pro Kategorie ein ähnliches Ergebnis beziehungsweise nur eine etwas geringere Genauigkeit und ein etwas höherer Loss resultierte, ist nicht davon auszugehen, dass 250 Bilder pro Kategorie unverhältnismäßig viele sind.

5.5.4 Architektur der Modelle

Die Modelle aus Tabelle 5.1 und 5.2 wurden nach dem Vorbild eines Tutorials [GM] erzeugt. In Abbildung 5.9 ist der Aufbau eines Models abgebildet. Es besteht aus vier Blöcken, wobei die ersten drei Blöcke identisch aufgebaut sind. Der letzte Block enthält den Output Layer, und ist daher anders aufgebaut.

Die Blöcke B1, B2, und B3 bestehen jeweils aus zwei Convolutional Layern, einem Max Pooling Layer sowie einem Dropout Layer. Der Dropout Layer verhindert Overfitting, indem jedes Neuron mit einer bestimmten Wahrscheinlichkeit während eines Trainings schritts ignoriert wird. Ausgenommen davon sind die Output Neuronen. Nach dem Training werden keine Neuronen mehr ignoriert [G17, S. 304].

Der Flatten-Layer im vierten Block wandelt den Input in ein eindimensionales Array um. Das ist nötig, denn der Output soll keine Matrix, sondern ein Vektor mit Wahrscheinlichkeiten der einzelnen Kategorien sein. Danach folgt ein Dense Layer, in dem jeder Input zu jedem Neuron in diesem Layer geleitet wird. Anschließend folgt ein weiterer Dropout Layer und ein weiterer Dense Layer, der die gleiche Anzahl an Neuronen wie auch zu ermittelnde Kategorien besitzt.

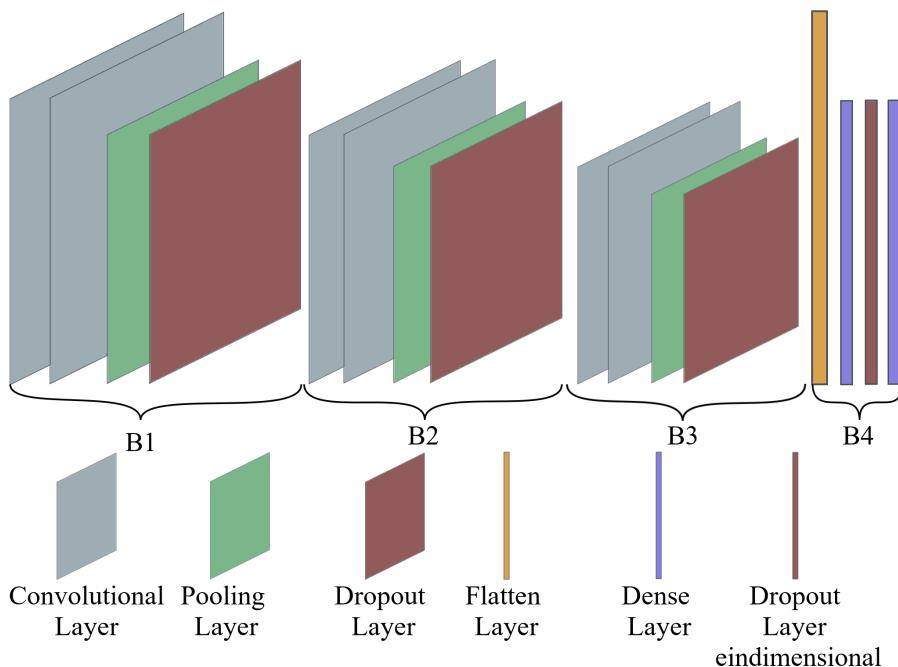


Abbildung 5.9: Architektur der Modelle

Listing 5.11 zeigt die konkrete Form des Models. Der erste Layer hat die Input-Dimension $320 \times 180 \times 32$. 320 entspricht dabei der Bildhöhe, 180 der Bildbreite. Die 32 gibt die Anzahl der Filter an. Das heißt, das Bild wird mit 32 verschiedenen Filtern gefiltert und die verschiedenen Ergebnisse sozusagen übereinander gestapelt. Die Filter haben eine Größe von 3×3 . Der Output dieses Layers ist unverändert $320 \times 180 \times 32$. Als Activation Function nutzen alle hier verwendeten Layer '`relu`',⁵.

⁵'relu' steht für rectified linear unit. Dies ist eine spezielle Funktion, deren Ergebnis angibt, ob ein Neuron durch die verbundenen Neuronen aus dem vorigen Layer aktiviert wird oder nicht

```

1 Model: "sequential"
2 -----
3 Layer (type)          Output Shape         Param #
4 =====
5 conv2d (Conv2D)        (None, 320, 180, 32)    320
6 conv2d_1 (Conv2D)      (None, 318, 178, 32)    9248
7 max_pooling2d (MaxPooling2D) (None, 159, 89, 32) 0
8 dropout (Dropout)      (None, 159, 89, 32)    0
9
10 conv2d_2 (Conv2D)       (None, 159, 89, 64)   18496
11 conv2d_3 (Conv2D)       (None, 157, 87, 64)   36928
12 max_pooling2d_1 (MaxPooling2D) (None, 78, 43, 64) 0
13 dropout_1 (Dropout)     (None, 78, 43, 64)    0
14
15 conv2d_4 (Conv2D)       (None, 78, 43, 64)   36928
16 conv2d_5 (Conv2D)       (None, 76, 41, 64)   36928
17 max_pooling2d_2 (MaxPooling 2D) (None, 38, 20, 64) 0
18 dropout_2 (Dropout)     (None, 38, 20, 64)    0
19
20 flatten (Flatten)      (None, 48640)        0
21 dense (Dense)          (None, 512)
22                         24904192
23 dropout_3 (Dropout)     (None, 512)        0
24 dense_1 (Dense)         (None, 183)        93879
25 =====
26 Total params: 25,098,444
27 Trainable params: 25,098,444
28 Non-trainable params: 0

```

Listing 5.11: Form der Layer

Durch den Convolution-Prozess durch den zweiten Layer wird die Dimension, quasi die Bildgröße, um 2 Pixel in Höhe und Breite verkleinert. Dies hat mit fehlendem Padding (Auffüllen der Ränder) zu tun.

Der Max Pooling Layer hat ein Pooling Window von 2×2 . Damit wird das Bild in beiden Dimensionen halbiert, also 318×89 . Die 32 Dimensionen durch die Filter bleiben erhalten.

Die Wahrscheinlichkeit, das Neuronen im Dropout Layer ignoriert werden, wird, auch für die nächsten beiden Dropout Layer, auf 0,25 gesetzt.

Die Kombination aus zwei Convolution-, einem Max Pooling- sowie einem Dropout Layer wird noch zwei Mal wiederholt. Die Anzahl der Filter in den Convolutional Layern wird nun allerdings auf 64 verdoppelt, um weitere Features zu extrahieren.

Im vierten Block (ab Zeile 20) erfolgt zunächst die Reduktion auf eine Dimension. Durch den Dense Layer sind alle 512 darin enthaltenen Neuronen zu allen Neuronen des vorigen Layers verbunden, durch den Dropout Layer werden diese jedoch mit einer Wahrscheinlichkeit von 0,5 ignoriert. Der letzte Layer ist ebenfalls ein Dense Layer, der genauso viele Neuronen wie zu klassifizierende Kategorien besitzt.

In diesem Fall sind das 183 Neuronen, denn es handelt sich hier im Beispiel um ein Model, welches die Kategorien beginnend mit A, E und F erkennt. Ein Model, welches alle Klassen erkennt, sieht identisch aus, nur die Anzahl der Output Neuronen unterscheidet sich entsprechend.

In der Spalte Params ist jeweils angegeben, wie viele Gewichtungen trainiert werden müssen. Beim ersten Convolutional Layer sind das beispielsweise 320, denn jeder Filter hat $3 \times 3 = 9$ Gewichte sowie ein Bias-Gewicht, welches auf jedes gefilterte Element addiert wird[Cho].

5.5.5 Wegfindung

Die Wegfindung wurde als einfache Breitensuche implementiert, s. Listing 5.12. Dafür werden in Zeile 2 und 3 Start- und Zielknoten anhand des Positions-⁶ oder Raumnamens ermittelt. In Zeile 4 wird die sogenannte Frontier initialisiert, welche immer alle Pfade enthält, die zu dem Zielknoten führen könnten. In der Liste `visited` befinden sich alle Namen der Knoten, die bereits besucht wurden.

⁶ohne Blickrichtung

Die Frontier ist hier eine FiFo-Liste (First in first out), das erste Element der Liste hat also die höchste Priorität und wird zuerst abgearbeitet. Neue Elemente werden am Ende dieser Liste hinzugefügt.

```

1 def bfs(start_node_name, dest_node_name):
2     start_node = find_node_with_name(start_node_name)
3     dest_node = find_node_with_name(dest_node_name)
4     frontier = [Path([start_node], [])]
5     visited = []
6
7     #bfs Frontier is a fifo
8     while len(frontier) > 0:
9         path = frontier[0]
10        frontier = frontier[1:]
11
12        if path.get_last_node().name == dest_node.name:
13            return path
14
15        for neighbor_node_name, direction in
16            Node.get_neighbors_as_list(path.get_last_node()):
17
18            if not neighbor_node_name is NO_NODE and
19                not neighbor_node_name in visited:
20                neighbor_node =
21                    find_node_with_name(neighbor_node_name)
22                if not neighbor_node is None:
23                    frontier.append(Path
24                        (nodes=path.nodes + [neighbor_node],
25                         directions=path.get_directions() +
26                         [direction]))
27
28    print("No Path could be found")
29    return None

```

Listing 5.12: Implementation der Breitensuche

Solange die Frontier Elemente besitzt und der Zielknoten nicht gefunden wurde, wird immer der erste Pfad aus der Frontier weiter exploriert (Zeile 9) und entfernt (Zeile 10). Enthält dieser Pfad den Zielknoten, wird dieser in Zeile 13 zurückgegeben, ansonsten werden alle Nachbarn des letzten Knotens des Pfades in der Frontier untersucht. Hierfür

werden alle Nachbarn ermittelt und iteriert (Zeile 15). Wenn ein Nachbarknoten gefunden wird, der noch nicht besucht wurde, wird die entsprechende Knoteninstanz ermittelt und der Pfad zu diesem Knoten an das Ende der Frontier angehängt (Zeile 20 bis 26). Die Namen der besuchten Knoten werden in einer Liste aus Strings gespeichert, anstatt als Referenzen zu den tatsächlichen Knotenobjekten, da andernfalls immer der Name eines Knotens referenziert werden muss. Ein Feld in den Knoten selbst ist ebenfalls unpraktikabel, da ansonsten Veränderungen in der Adjazenzliste vorgenommen werden, die bei jeder Berechnung der Route entfernt werden müssen. Zeile 27 und 28 werden nur dann erreicht, wenn keine Route gefunden werden konnte.

Der Breitensuche-Algorithmus ist nicht der effizienteste Algorithmus, garantiert aber zum Beispiel im Vergleich zur Tiefensuche die kürzeste Route, wie in Kapitel 2 bereits erwähnt. Daneben ist die Implementierung sehr nachvollziehbar. Der dem Algorithmus zugrundeliegende Graph hat lediglich 160 Knoten und 105 Räume, welche im Suchbaum immer Blätter sind. Zusammen spiegeln sie alle Positionen und Räume innerhalb der Fachhochschule Wedel wider. Da die bereits besuchten Knoten in einer Liste festgehalten werden, kann der Suchbaum maximal 160 Knoten enthalten, weil kein Knoten mehrfach vorkommen kann. Die Antwortzeiten bewegen sich im Bereich weniger Millisekunden, siehe Kapitel 6, sodass eine Optimierung nicht notwendig ist.

5.6 Augmented Reality

In diesem Abschnitt wird die Realisierung der Teilanwendung Augmented Reality beschrieben. Zur Übersicht wird als Erstes ein Diagramm vorgestellt und erläutert, welches die Verbindungen zwischen einzelnen Skripten und Klassen nach der Art eines Klassendiagramms darstellt. Anschließend wird auf die zwei verwendeten Berechnungsmethoden bezüglich der Objekterkennung eingegangen.

5.6.1 SoftwareDesign

Das Diagramm 5.10 beinhaltet bis auf den Navigator keine Klassen, sondern zeigt, wie die einzelnen Python-Dateien und Funktionen sich untereinander aufrufen. Hierbei gibt es jedoch Überschneidungen zum Klassendiagramm aus dem vorigen Abschnitt zur Indoor-Navigation. Diese sowie alle beteiligten Klassen und gegebenenfalls Funktionen werden nachfolgend erläutert.

5 Realisierung

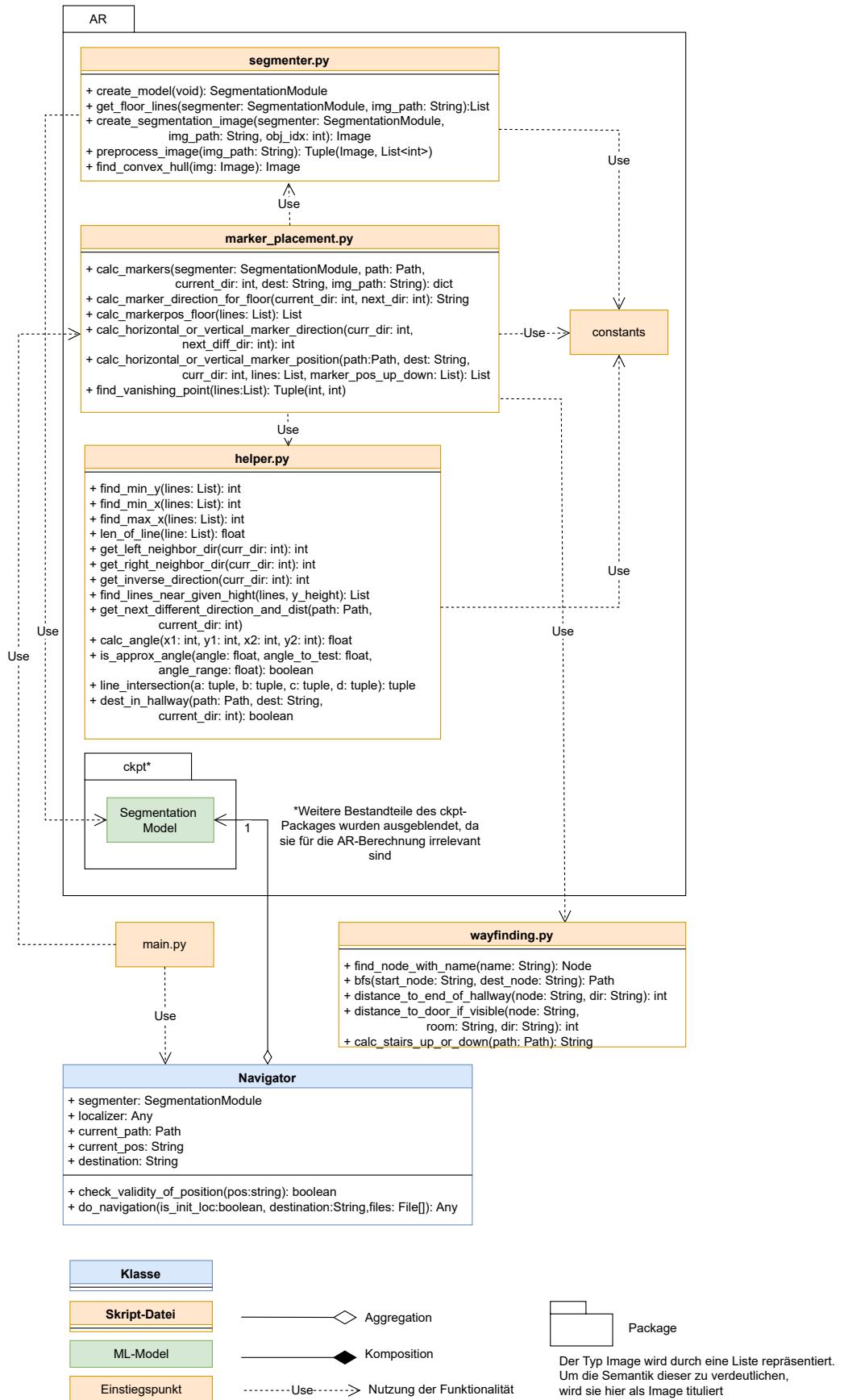


Abbildung 5.10: Diagramm der Zusammenhänge der Python-Files

Gemeinsamkeiten zur Indoor Navigation

Auch hier werden die Navigator-Klasse und das `main`-Skript verwendet. Das `main`-Skript ist ebenfalls der Einstiegspunkt in die Anwendung und initiiert die durch den Client angefragte Aktion. In dieser Teilanwendung ist das die Berechnung der Art, Ausrichtung und Position der Icons, die NutzerInnen den Weg weisen sollen, welche als Marker bezeichnet werden. Dafür wird der `segmenter` aus dem Navigator benötigt, welcher vereinfacht dargestellt wird und aus dem `ckpt`-Package stammt.

Ein gespeichertes Model besteht hier aus den Dateien `keras_metadata.pb` - den Metadaten - und `saved_model.pb`, in der die Architektur und die Trainingskonfiguration gespeichert sind. Außerdem gibt es einen Ordner `variables`, der die Gewichtungen des Models enthält [Tenb]. Diese Einzelheiten werden in Abbildung 5.10 jedoch ausgeblendet.

Das Segmentierungsmodel wurde mit dem ADE20K Datensatz trainiert, welcher speziell für Segmentierung erstellt wurde [Vis] .

`marker_placement.py`

Diese Datei übernimmt die Hauptaufgabe dieses Teilprojekts: die Berechnung der Marker in Art, Ausrichtung und Position. Dafür wie die Funktion `calc_markers` aufgerufen. Diese benötigt den `segmenter` aus der Navigator-Klasse, sofern die Objekterkennung mittels Segmentierung durchgeführt wird, sowie den Pfad, die aktuelle Blickrichtung, das Ziel, und ein Bild, auf dem die Berechnung durchgeführt wird. `calc_marker_directions_for_floor` berechnet die Ausrichtung der Marker, `calc_markerpos_floor` die Position der Marker. Letztere Funktion nutzt dafür die Hilfsfunktion `find_vanishing_point`, mit dem eine perspektivische Berechnung ermöglicht wird. Das Ergebnis dieser Berechnung sind Linien, bestehend aus zwei Punkten im Bild, welche die ermittelten Bodenflächen umrahmen. Die Berechnung eines Markers, der die nächste Abbiegung anzeigt, ist in `calc_horizontal_or_vertical_marker_direction` (Berechnung der Ausrichtung) und `calc_horizontal_or_vertical_marker_position` (Berechnung der Position) implementiert.

`segmenter.py`

In diesem Python-Skript finden sich alle Funktionen, die mit der Bildsegmentierung zu tun haben. `create_model` erzeugt das Segmentierungs-Model, welches durch den Navigator genutzt wird. Sobald das Model existiert, kann mit `create_image_segmentation` das Bild segmentiert werden. Obwohl das Model mehr Objekte erkennen kann, wird hier

nur die Segmentierung eines Objekts weiterverwendet. Alle anderen erkannten Objekte werden verworfen. In diesem Projekt ist das der Boden. Damit dieser besser klassifiziert werden kann, sollte das Inputbild vorher mit `preprocess_image` vorbereitet werden. Dafür werden Kantenerkennung und Weichzeichner eingesetzt. Die Bodensegmentierung wird durch Linien umrahmt. Durch die Berechnung einer konvexen Hülle sollen Artefakte durch eine falsche Segmentierung ausgeschlossen werden.

helper.py

Diese Skriptdatei enthält Funktionen, die in `marker_placement` genutzt werden. Viele davon sind Berechnungen auf den durch den `segmenter` ermittelten Linien um den Boden, andere enthalten aber auch Berechnungen zur Bestimmung der Ausrichtung der Marker.

constants.py

Diese Datei enthält verschiedenste Konstanten, zum Beispiel die verwendete Bildgröße, Richtungsangaben, Pfade zu Bildern oder Ordnern, oder Winkelangaben, durch die verschiedene Linientypen voneinander abgegrenzt werden können.

5.6.2 AR-Implementationen

Eine der wichtigsten Grundlagen, um markerlose AR zu realisieren, ist eine gute Objekterkennung. In diesem Projekt ist besonders wichtig, den Boden zu erkennen, da die Marker auf diesen und nicht an der Decke oder Wänden gezeichnet werden sollen. Zudem sollen die Türen erkannt werden, die NutzerInnen als Ziel angegeben haben.

Es wurden zwei verschiedene Varianten implementiert. Eine nutzt reine Bildverarbeitungsmethoden, die andere zusätzlich Machine Learning. Diese werden im folgenden vorgestellt.

AR mit reiner Bildverarbeitung

Dieser Ansatz nutzt reine Bildverarbeitung, wurde jedoch aufgrund der Instabilität bezüglich Boden- und Wetterverhältnissen verworfen, enthält aber Methodiken, die auch in der zweiten Implementation Verwendung finden.

Objekterkennung durch Bildverarbeitung

Da die Gänge eines Gebäudes üblicherweise sehr symmetrisch und gradlinig sind, lag die Vermutung nahe, durch Kantenerkennung und Bildverarbeitung die wichtigen Bildkomponenten, also vor allem Boden und Türen, extrahieren zu können.

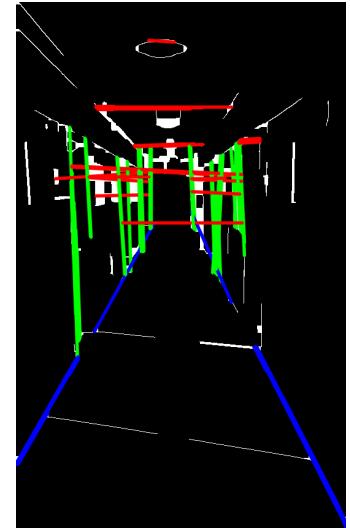
(a) Aufbereitete Aufnahme



(b) Kantenerkennung



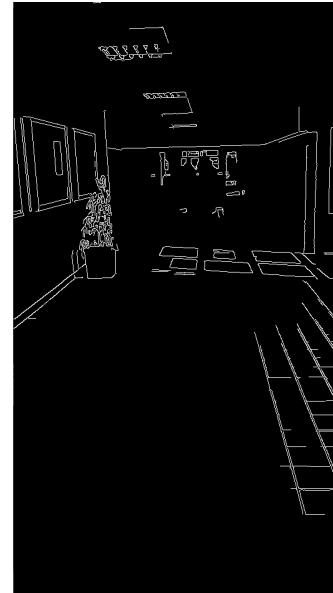
(c) Gefilterte erkannte Linien



(d) Aufbereitete Aufnahme



(e) Kantenerkennung



(f) Gefilterte erkannte Linien

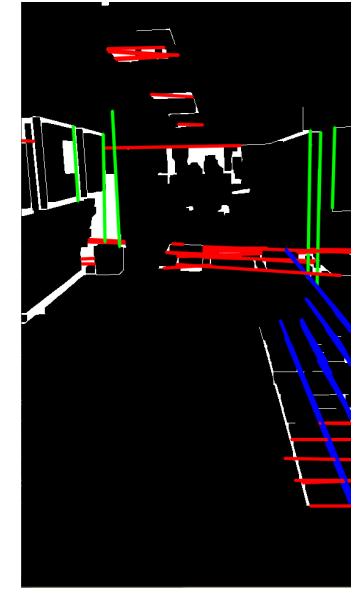


Abbildung 5.11: Objekterkennung mit Bildverarbeitungsmaßnahmen

Die obere Zeile enthält gute Resultate, Ergebnisse in der unteren Zeile sind unzureichend

Dafür wurde zunächst das Bild aufbereitet. Zum einen wurde der Kontrast erhöht, eine Helligkeitsanpassung anhand der ermittelten Durchschnittshelligkeit durchgeführt und

5 Realisierung

das Bild in Graustufen umgewandelt. Das Ergebnis zweier verschiedener aufbereiteter Aufnahmen ist in Abbildung 5.11a und 5.11d zu sehen. Anschließend wurde die Kantenerkennung mit dem Canny-Algorithmus durchgeführt. Dafür muss das Bild zunächst weichgezeichnet werden, um Rauschen zu vermindern [RCG18, S. 165]. Das Ergebnis dieser Operation ist in 5.11b und 5.11e zu sehen. Dieses wurde mit Dilatation und Erosion weiter verarbeitet, in diesem Fall wurde Closing [Clo] eingesetzt. Durch dieses Verfahren werden kleinere Löcher in den durch Canny erkannten Kanten geschlossen. Dafür wird zunächst eine Dilation gefolgt von einer Erosion durchgeführt. Eine Dilation verbreitert die Linien und schließt somit kleine Löcher, die die Linien unterbrechen. Um die ursprüngliche Liniendicke wiederherzustellen, wird eine Erosion angewendet, die die Linien wieder schmälert. Dieses neu entstandene Bild wird als Grundlage für eine generelle Linienerkennung verwendet.

Bei der Linienerkennung wird der Hough Line Transform-Algorithmus verwendet. Dieser ist im Stande, gerade Linien zu erkennen. Da jedoch nicht alle Linien benötigt werden, werden diese gefiltert. Die Filterkriterien beziehen sich vor allem auf den Winkel der Linien bezüglich der x-Achse und auf ihre Länge. Sehr kurze Linien werden als irrelevant betrachtet und verworfen. Linien, die ungefähr eine 60° -Neigung besitzen, werden als Linien betrachtet, die den Gang zu den Seiten hin begrenzen. Sie sind in blau eingefärbt (Abbildung 5.11c und 5.11f). Vertikale Linien werden als Türen oder Wände betrachtet (grün eingefärbt) und horizontale Linien als Ende eines Gangs oder Teil der Wände (rot eingefärbt). Die erste Horizontale wird dabei immer als das Ende des Gangs angenommen. Die Winkel werden immer innerhalb eines Bereichs detektiert, falls NutzerInnen die Kamera nicht exakt gerade halten und weil die Gänge teilweise nicht exakt gleichbreit sind und somit einen leicht anderen Winkel haben.

Das Resultat der Filterung ist in Unterabbildung in Abbildung 5.11c und 5.11f zu sehen. In 5.11c wurden alle relevanten Linien erkannt. Die blauen diagonalen Linien zeigen die seitliche Begrenzung des Gangs an und die erste Horizontale zeigt korrekt das Ende des Gangs an. Lediglich bei den grünen Vertikalen wurden viele nicht relevante Linien nicht herausgefiltert, die erste Tür (linke Seite, erste Vertikale) ist dennoch deutlich anhand der Länge der Linie zu erkennen. Daneben werden aber auch beispielsweise Bilder oder Pinnwände an Wänden erkannt, die sich aufgrund der perspektivischen Verzerrung nicht anhand der Länge der Linie herausfiltern lassen.

Teilweise werden Linien auch nicht oder nicht vollständig erkannt, wie zum Beispiel bei der zweiten Vertikale von links, die ebenfalls zum Türrahmen der ersten Tür links gehört, die bei ungefähr der Hälfte der eigentlichen Länge abbricht. Insbesondere in Bild 5.11f sind die Schwächen dieses Ansatzes klar zu sehen. Zum Einen werden die Fugen für Begrenzungen gehalten und je nach Lichtverhältnissen erkannt oder nicht erkannt. Die linke Seite auf Bild 5.11d ist etwas dunkler als die rechte Seite, da sich dort die Glasfront zum Innenhof befindet. Daher werden dort die Fugen erkannt, auf der linken Seite jedoch nicht. Das allein würde nur zu einer Schmälerung des Ganges führen, gegebenenfalls auch nur auf einer Seite, sodass die Marker nicht mehr in der Mitte des Ganges platziert werden würden, die Navigationsinformationen würden jedoch weiterhin korrekt auf dem Boden platziert werden. Ein größeres Problem ist jedoch die teilweise stark variierende Helligkeit innerhalb eines Bildes. Zum einen ist in Bild 5.11d das Ende des Flurs sehr dunkel. Die Muster, die bei sonnigem Wetter durch die Fenster auf dem Boden gezeichnet werden, sind aber vergleichsweise sehr hell und werden von der Kantenerkennung als horizontale Linien erkannt, wie in Bild 5.11f zu sehen ist. Daher müsste in diesem Fall keine Kontrasterhöhung sondern lokal eine Verminderung des Kontrastes durchgeführt werden. Zudem werden in den Fugen weitere horizontale Linien erkannt, was das richtige Filtern erschwert. Ein weiterer Negativpunkt ist die benötigte Bildgröße für eine akkurate Linienerkennung. Die meisten relevanten Linien werden erst ab einer Größe von 800×450 Pixeln erkannt, jedoch nicht alle. Dies zeigt Tabelle 5.4, bei der die Objekterkennung mit verschiedenen Bildgrößen getestet wurden. Bei sehr kleinen Bildgrößen, wie etwa 90×160 , werden gar keine Objekte beziehungsweise Linien erkannt, die Anzahl der Linien steigt aber mit Erhöhung der Bildgröße sukzessive an. Zunächst werden vertikale oder horizontale Linien erkannt, diese indizieren aber nicht zwangsläufig die zu detektierenden Features des Bildes. Die Horizontalen begrenzen in allen Unterabbildungen nicht das Ende des Ganges wie gewünscht, sondern Schaukästen oder durch die Sonne entstandene Muster auf dem Boden. Erst ab einer Auflösung von 360×640 werden teilweise Diagonalen, die seitlich den Boden begrenzen, erkannt. Die Erkennung der diagonalen Linien wird mit zunehmender Größe genauer und ist in diesem Beispiel erst ab einer Größe von 540×960 Pixeln für eine Objekterkennung nutzbar. Daher wurde die Idee, reine Bildverarbeitung zu verwenden, verworfen.

5 Realisierung

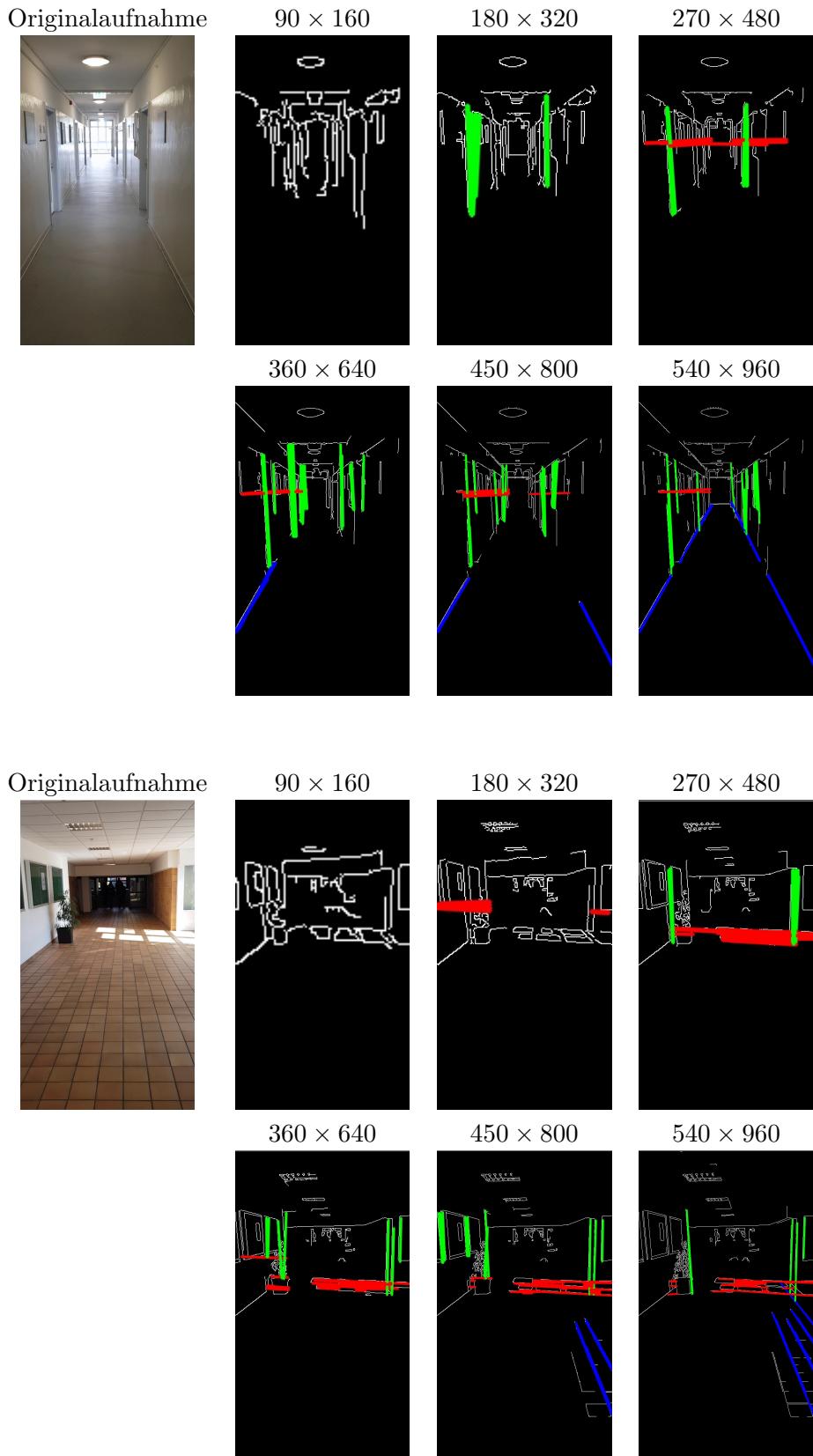


Tabelle 5.4: Bodenerkennung mit Bildverarbeitungsmaßnahmen
Die verschiedenen Teilbilder zeigen die Objekterkennung in verschiedenen Bildgrößen

AR mit Machine Learning

Stattdessen wurde der Ansatz der reinen Bildverarbeitung, der eine Objekterkennung mit Hilfe von Linienerkennung realisiert, um Machine Learning erweitert und ins Projekt integriert. Ein Machine Learning Model soll dabei die Segmentierung des Fußbodens durchführen. Daneben benötigt die Berechnung der AR-Marker weitere Bildverarbeitungs- sowie Rechenschritte. Diese werden nachfolgend detailliert dargestellt und erläutert.

Der Einstiegspunkt der gesamten AR-Berechnung ist die Funktion `calc_markers`, welche als Parameter das Segmentierungsmodell, den aktuellen Pfad, die aktuelle Blickrichtung, das Ziel sowie ein Bild, anhand dem die Berechnung stattfinden soll, entgegennimmt. Der erste Schritt ist, zu prüfen, ob der Nutzer bereits in die richtige Richtung schaut oder ob er sich umdrehen muss. Muss er sich umdrehen, ist die Berechnung beendet und nur diese Information wird zurückgegeben.

Objekterkennung

Andernfalls ist der zweite Schritt die Berechnung einer Umrandung des Bodens. Hierbei wird das Bild, wie im vorigen Ansatz mit reiner Bildverarbeitung auch, vorbereitet. Es wird weichgezeichnet und die Größe angepasst. Der Unterschied zur vorigen Vorbereitung ist die Umwandlung in ein Graustufenbild, denn in diesem Schritt wird die Farbinformation beibehalten. Mit Hilfe dieser können zu einem Objekt gehörige Pixel besser von anderen unterschieden werden. Das so angepasste Bild wird in das Segmentierungsmodell gegeben. Das Resultat daraus ist eine Liste an Wahrscheinlichkeiten für jedes Objekt für jeden Pixel.

5 Realisierung

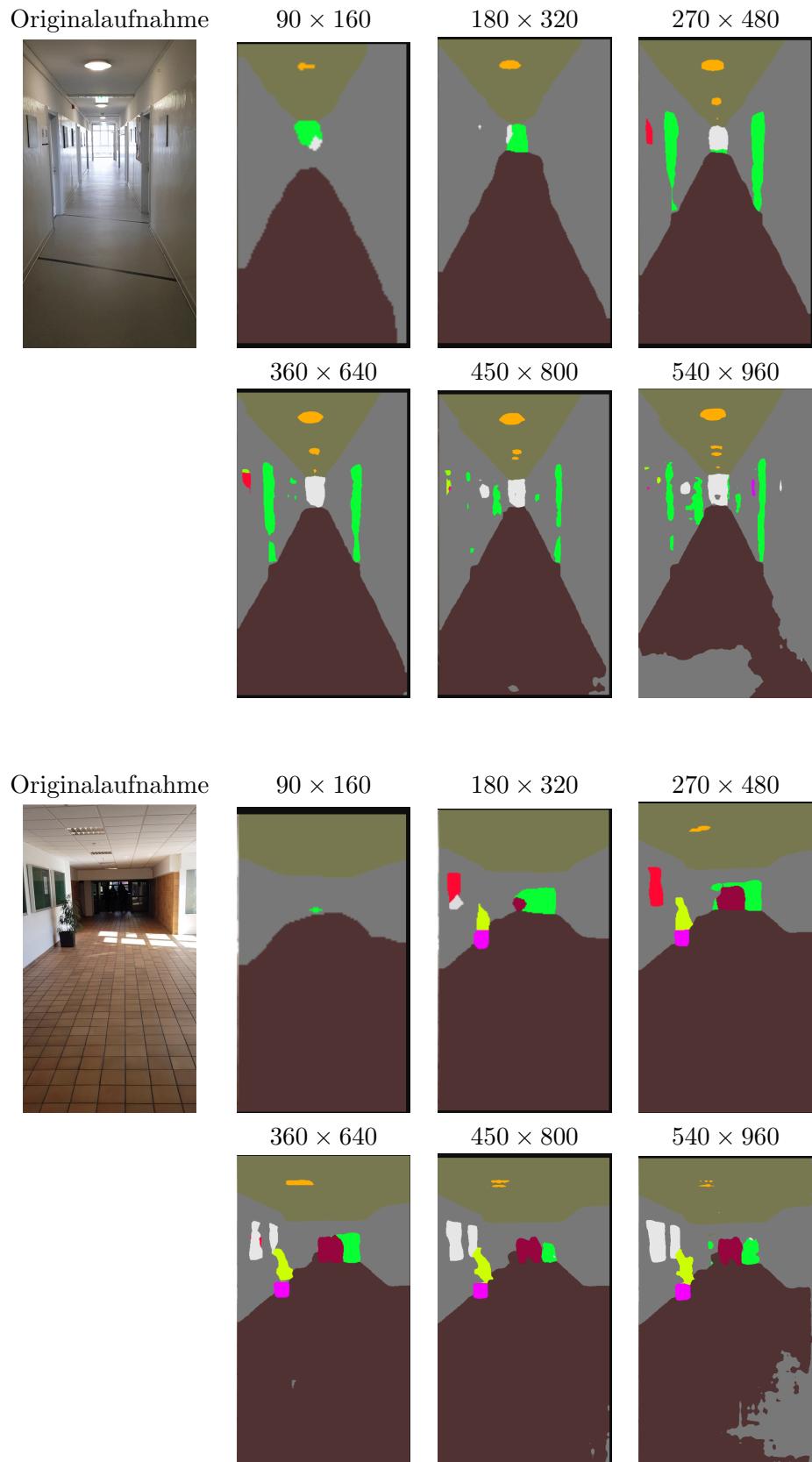


Tabelle 5.5: Bodenerkennung mit Segmentierung in verschiedenen Bildgrößen

Ergebnis der Segmentierung Die Färbung der einzelnen Segmente können wie folgt den Objektkategorien zugeordnet werden⁷:

- Wände - grau
- Decke - olivgrün
- Boden - braun
- Türen - hellgrün
- Fenster - hellgrau
- Lampen - orange
- Personen - dunkelpink
- Pflanzen - gelb
- Blumentopf - pink

In Abbildung 5.5 wurden verschiedene Bildgrößen getestet, um die kleinstmögliche Größe zu ermitteln, bei der die Objekterkennung genau genug ist. Hierbei ist zu erkennen, dass eine Segmentierung im Vergleich zu 5.4 schon ab einer Größe von 90×160 Pixeln brauchbare Ergebnisse liefert. Mit reiner Bildverarbeitung können bei dieser Auflösung gar keine Objekte erkannt werden. Ab einer Größe von 180×320 Pixeln können bereits einzelne Details wie Lichter oder Blumentöpfen erkannt werden. Je größer das Bild, desto genauer werden diese Details erkannt und je genauer ist die Klassifizierung kleinerer Objekte. Ist die Auflösung allerdings zu hoch, können sich Artefakte bilden, bei denen die Segmentierung falsche Ergebnisse liefert. Das kann damit zusammenhängen, dass bei der Segmentierung die Nachbarpixel mitbetrachtet werden. Wenn diese ähnlich aussehen, werden sie ebenfalls als ein falsches Objekt klassifiziert. So ergeben sich je nach Größe des Bildes verschieden große Artefakte, wie in Tabelle 5.5 durch die Bilder mit den Bildgrößen 450×800 sowie 540×960 zu erkennen ist.

⁷Kleinere Segmente wurden der Übersichtlichkeit halber außen vor gelassen

Konvexe Hülle und Linienerkennung

Nachdem der Boden erfolgreich segmentiert ist, wird eine konvexe Hülle um alle Bodensegmente gezeichnet, um Artefakte auszubessern. Dies kann gemacht werden, da der Boden eines Gangs in der FH bis auf einige Ausnahmen eine konvexe Form hat - ein Rechteck, dessen Breite abhängig der Entfernung zur Kamera zu- oder abnimmt. Sollten sich Hindernisse oder Personen innerhalb der konvexen Hülle der Bodensegmente befinden, werden diese jedoch mit eingeschlossen. Die Berechnung der Marker schließt diese Information also nicht ein, daher werden Marker potenziell über Hindernisse oder Personen gezeichnet.

Die konvexe Hülle wird als Grundlage für die Linienberechnung verwendet, die wie in Abschnitt 5.6.2 ebenfalls über Hough Lines Transform durchgeführt wird und die eine Liste der erkannten Linien um das Bodensegment zurückgibt.

Berechnung der Markerpositionen des Bodens

Der nächste Schritt ist die Berechnung der Positionen der Marker auf dem Boden. Dies wird mit der Funktion `calc_markerpos_floor` nach der Methode Albertis[GG87] realisiert. Dabei wird als Erstes der Fluchtpunkt bestimmt, der in Abbildung 5.12 als Punkt V bezeichnet ist. Um diesen zu berechnen, müssen anhand der zuvor ermittelten Linien die richtigen Diagonalen ermittelt werden, die den Gang seitlich begrenzen. Von dem Punkt V ausgehend wird ein neuer Punkt P auf der selben Höhe, aber in der x-Achse verschoben, etabliert. Dieser ist Ausgangspunkt mehrerer Strahlen, die die x-Achse äquidistant schneiden. Auf der Seite, auf der sich auch Punkt P befindet, wird eine Vertikale \overline{Q} gezogen. Hierbei ergeben sich verschiedene Schnittpunkte mit den Strahlen auf der Höhe y_1 bis y_8 .

Die Position der Marker ergibt sich nun aus x_v und den y-Koordinaten y_1 bis y_5 . y_6 bis y_8 werden verworfen, da sich diese nicht mehr auf der Bodenfläche befinden würden.

Berechnung der Markerposition der nächsten Abbiegung

Neben der Information, in welche Richtung NutzerInnen weitergehen sollen, wird auch eine Anzeige benötigt, die die Richtung der nächsten Abbiegung anzeigt. Dafür muss zunächst ermittelt werden, in welche Richtung abgebogen werden soll. Da die im Pfad berechneten Richtungen global und durchnummieriert sind, muss dafür berechnet werden, ob die nächste Richtung von der aktuellen Position aus eine Links- oder Rechtsabbiegung ist.

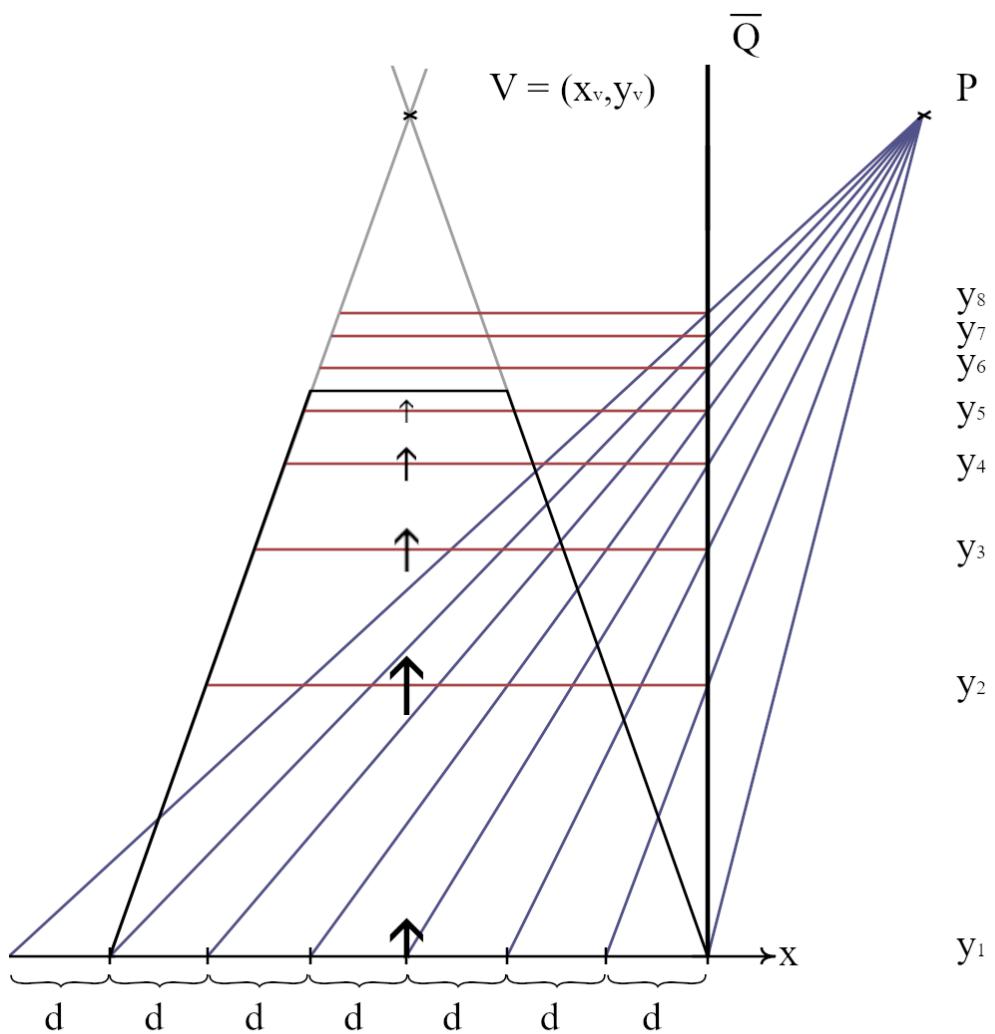


Abbildung 5.12: Berechnung der Markerpositionen auf dem Boden

5 Realisierung

Um die richtige Position des Markers zu finden, wurde ein geometrischer Ansatz gewählt, da dieser aus der Implementierung mit reinen Bildverarbeitungsmethoden großteils übernommen werden konnte. Es kann jedoch auch das Segmentierungsmodell verwendet werden. Wenn zum Beispiel Türen jedoch gar nicht sichtbar sind, kann diese Art Marker nicht bestimmt werden.

Bei dieser Berechnung wird unterschieden, ob am Ende des Ganges abgebogen werden muss oder sich das Ziel im selben Gang befindet. Ist ersteres der Fall, wird die Position des letzten Markers, in Abbildung 5.12 der Punkt mit den Koordinaten (x_v, y_5) , für den Boden als Grundlage genommen und nur ein konstanter Offset auf die y-Koordinate addiert.

Ansonsten wird die Entfernung zum Ende des Flurs sowie der Tür zum Ziel berechnet. Durch diese Informationen lässt sich das Intervall zwischen zwei Boden-Markern berechnen, in welchem ein konstanter Skalierungsfaktor angenommen wird.

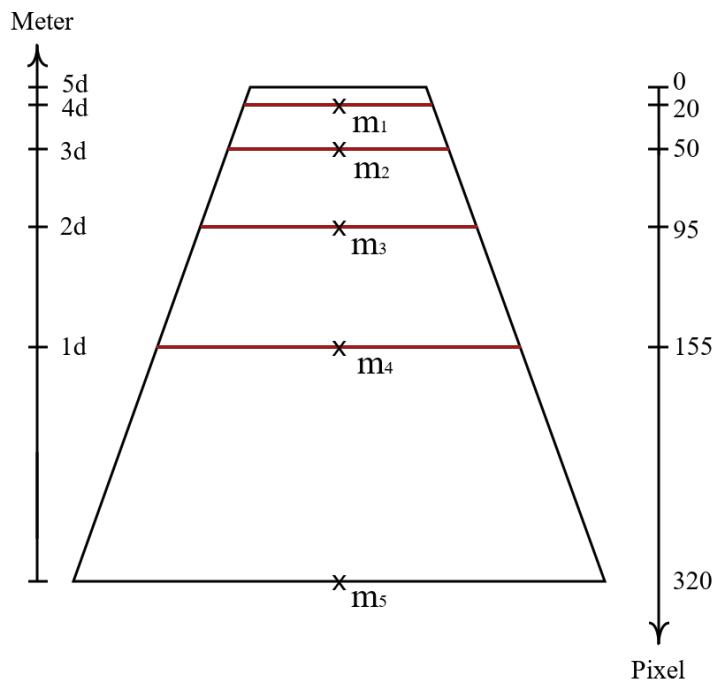


Abbildung 5.13: Ermittlung des richtigen Intervalls

In Abbildung 5.13 sind die Markerpositionen mit m_1 bis m_5 auf dem Bodensegment eingezeichnet. Auf der linken Seite befindet sich eine Achse, die die Entfernung eines/r NutzrIn in Metern anzeigt, auf der rechten Seite eine Achse, die sie in Pixeln anzeigt. d bezeichnet den realen Abstand zwischen zwei Markern.

Mit der Entfernung zum Ziel lässt sich bestimmen, zwischen welchen Markern das Ziel liegt. Ist es zum Beispiel $2,2d$ entfernt, liegt es entsprechend zwischen m_3 und m_2 . Durch die perspektivische Verzerrung wird eine Strecke, wenn sie weiter entfernt ist, immer kürzer. Der Abstand der Marker m_3 und m_2 auf der Pixel-Achse ist mit 45 Pixeln kürzer als der Abstand von m_4 und m_5 mit 165 Pixeln, da diese weiter entfernt liegen. Diese Verzerrung ist nicht diskret, wird für die Berechnung hier jedoch diskretisiert, das heißt, für ein Intervall zwischen zwei Markern wird eine feste Skalierung s angenommen.

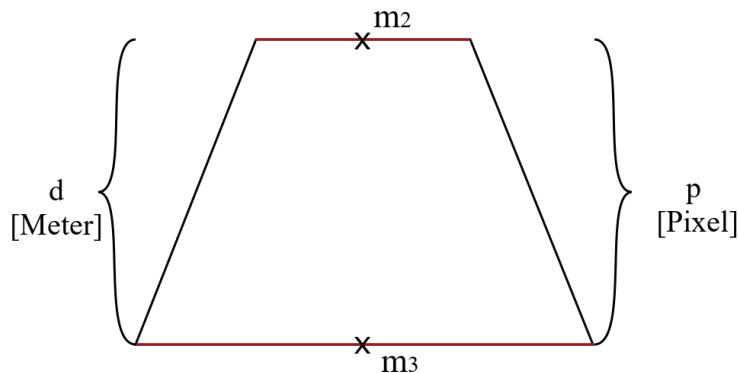


Abbildung 5.14: Berechnung der Markerposition innerhalb des Intervalls

Nun muss die Position des zielanzeigenden Markers innerhalb des Intervalls der entsprechenden zwei Marker bestimmt werden. Dieser Prozess ist in Abbildung 5.14 dargestellt. Zunächst wird bestimmt, welcher Länge ein Pixel entspricht, was mit $d/p = s$ berechnet wird. p ist dabei die Distanz zwischen den Markern m_3 und m_2 in Pixeln.

d' sei die Differenz der Entfernung von Marker m_3 , also dem Marker, der näher an der aktuellen Position liegt, in Metern, und der Entfernung zum Ziel. Liegt Marker m_3 in $2d$ Entfernung und das Ziel befindet sich in $2,2d$ Entfernung, ist $d' = 0,2d$. Abschließend muss berechnet werden, wie vielen Pixeln d' entspricht. Dieser Wert ergibt das Offset an, welches der zielanzeigende Marker von dem Marker m_3 hat.

Nach der Berechnung werden alle Boden-Marker gefiltert, sodass keine Marker hinter dem zuvor berechneten Marker gezeichnet werden. Das Ergebnis ist ein JSON-Objekt wie in Listing 5.7 am Anfang zu sehen.

5.6.3 Vergleich der Implementationen

Bildverarbeitungsmethoden sind im Vergleich zu Machine Learning Ansätzen einfacher zu verwenden, zum einen weil der Nutzung von ML-Models ein Training vorausgeht, welches vor allem bei Bildverarbeitung sehr zeit- und rechenaufwendig aufwendig sein kann, zum anderen, weil die Berechnung auf einer gewöhnlichen CPU (Central Processing Unit) deutlich schneller ist als bei ML-Models.

Die Schwachstelle dieses Ansatzes ist allerdings, dass die Bodenerkennung nicht robust ist und durch Lichteinfall durch Fenster stark beeinträchtigt wird. Auch hat der Ansatz Schwierigkeiten mit geringem Kontrast und Dunkelheit. Um das Problem zu vermindern, könnten weitere Bildverarbeitungsmethoden eingesetzt werden. In welchem Grad das möglich ist, wurde in diesem Projekt nicht untersucht.

Stattdessen wurde eine Segmentierung mittels Machine Learning gewählt, mit der eine deutlich robustere Objekterkennung möglich ist. Hierbei ist die Güte der Segmentierung von der Größe des Bildes abhängig. Ist das Bild zu groß, enthält es zu viel Rauschen und zu viele Details, die eine korrekte Segmentierung beeinträchtigen. Um schnelle Berechnungen des Models zu ermöglichen, wird außerdem eine GPU benötigt.

5.7 Zusammenfassung und Fazit

In diesem Kapitel wird die Realisierung des praktischen Teils der Arbeit beleuchtet. Am Anfang des Kapitels werden die verwendeten Technologien aufgelistet, anschließend grob die Architektur sowie die Kommunikation zwischen Front- und Backend beschrieben. Der darauf folgende Abschnitt befasst sich mit der internen Darstellung der Fachhochschule Wedel. Diese wird vor allem für die Implementierung der Wegfindung benötigt, findet teilweise aber auch in der Implementierung des AR-Moduls durch die Nutzung der Pfadinformation Verwendung. Die darauf folgenden Unterpunkte beinhalten eine detaillierte Darstellung der Realisierung der beiden Komponenten Indoor Navigation und Augmented Reality.

Der Punkt Indoor Navigation enthält Informationen, wie die Daten erhoben wurden. In der FH Wedel wurden alle 2,5 Meter Videos eines Gangs aufgenommen und später in eine Menge an Bildern umgewandelt. Anschließend wurden mit diesen Daten in verschiedenen Variationen verschiedene ML-Modelle trainiert. Es wurden dabei CNNs

5 Realisierung

trainiert, die teils von Grund auf aufgebaut als auch durch Transferlearning trainiert wurden. Die Genauigkeit der Klassifizierung der Position durch die Models ist mit teilweise über 98% sehr gut, allerdings geht diesem Verfahren ein hoher manueller Aufwand der Datenbeschaffung und Modellierung des Gebäudes durch Adjazenzlisten voraus. Auch, wenn sich die Optik eines Teils des Gebäudes grundlegend ändert, muss mit hoher Wahrscheinlichkeit neues Bildmaterial aufgenommen werden. Dafür wird keine weitere Hardware für Emitter oder Receiver benötigt. Ebenso sind keine Marker innerhalb der Gänge notwendig.

Der Unterpunkt Augmented Reality enthält im Wesentlichen zwei verschiedene Ansätze: einen rein Bildverarbeitungsbasierten und einen ML-basierten. Zunächst war die Annahme, dass ein rein Bildverarbeitungsbasierter ausreichend gut ist, welcher vor allem Edge Detection nutzt. Dies hat sich allerdings nicht bewahrheitet, denn dieser ist nicht besonders robust gegen verschiedene Lichtverhältnisse sowohl von innen durch an- oder ausgeschaltete Lampen als auch von außen durch verschiedene Wetterbedingungen. Daher wurde ein rechenaufwendigerer Ansatz mit Machine Learning implementiert, welcher sich als deutlich robuster herausstellte. Wie auch die Indoor Navigation, benötigt dieser Ansatz keine Marker.

Nachdem die Realisierung des Projekts ausführlich erläutert wurde, wird im folgenden Kapitel das erzielte Ergebnis dieser Arbeit betrachtet und die zuvor aufgestellten Anforderungen hinsichtlich ihrer Erfüllung bewertet.

6

Ergebnis

Dieses Kapitel beschäftigt sich mit dem erzielten Ergebnis der Praxisleistung. Im ersten Abschnitt wird erklärt, wie die App zu starten und zu benutzen ist. Anschließend werden die in Kapitel 3 und 4 genannten Anforderungen mit dem tatsächlich erzielten Ergebnis verglichen. Dabei werden Indoor Navigation und Augmented Reality wieder getrennt betrachtet.

6.1 Starten und Nutzung der Anwendung

Um die Anwendung zu starten, müssen die Frontend- und Backendapplikationen einzeln gestartet werden.

Starten des Clients: In dem Ordner frontend/indoor-nav muss ein Terminal geöffnet werden. Anschließend muss der Befehl `npm start` eingegeben werden. Dann ist die Applikation über die URL `http://localhost:3000/navigation` erreichbar.

Starten des Backends: Zunächst sollte mit Conda eine Umgebung erstellt werden. Dafür kann das Environment-File `environment.yaml` im Root-Verzeichnis des Projekts verwendet werden. Dies kann mit dem Befehl in Listing 6.1 in Zeile 1 getan werden. Um die Umgebung zu aktivieren, muss der Befehl in Zeile 2 ins Terminal eingegeben werden. Die Anwendung kann in der aktivierte Umgebung im root-Verzeichnis mit dem Befehl in Zeile 3 gestartet werden. Jetzt läuft der Server und kann Anfragen des Clients entgegennehmen.

```
1 conda env create -f environment.yml
2 conda activate indoor-nav
3 python backend_indoor_nav/main.py
```

Listing 6.1: Befehle zum Starten des Backends

Abbildung 6.1 zeigt den Ablauf einer Navigation anhand von Screenshots. Als erster Schritt muss ein Ziel per Dropdown-Menü ausgewählt werden, zum Beispiel Seminarraum 6 (Abbildung 6.1a), der intern in die entsprechende Raumbezeichnung A2.06 übersetzt wird. AnwenderInnen werden dann aufgefordert, die Kamera zu schwenken, wie in 6.1b zu sehen. Anschließend beginnt die Berechnung der Position, des Pfades und der Marker und die nötigen Informationen werden im Frontend angezeigt, wie in 6.1c zu sehen. Ist das Ziel erreicht, ist der Text *Ziel erreicht* zu sehen und die Navigation ist abgeschlossen.



Abbildung 6.1: Aufnahmen der laufenden Applikation

6.2 Anforderungen Indoor Navigation

Funktional

1) Zieleingabe

Über ein Dropdown-Menü kann über die Benutzeroberfläche eine Zieleingabe vorgenommen werden.

2) Abweichung vom berechneten Pfad

Durch die neue Standortermittlung bei jeder Anfrage, kann der/die NutzerIn vom Pfad abweichen und bekommt mit der nächsten Abfrage eine aktualisierte Route.

3) Machine Learning

Für die Standortbestimmung wurde hauptsächlich Machine Learning verwendet.

4) Falsch ermittelte Standorte verwerfen

Durch die Validierung und die zuvor erwähnte Mindestwahrscheinlichkeit der ermittelten Kategorie, werden die meisten falsch erkannten Standorte verworfen, wenn diese zu weit von der zuvor ermittelten Position entfernt sind.

Problematisch sind einige Sonderfälle, wie etwa, wenn der Standort zwar richtig, die Ausrichtung aber falsch klassifiziert wird. Dann kann der Standort nicht verworfen werden, die AR-Komponente geht dann allerdings davon aus, dass der/die NutzerIn sich umdrehen muss.

Noch problematischer ist, wenn ein ermittelter Standort in die entgegengesetzte Richtung der tatsächlichen Laufrichtung berechnet wird. Dann funktioniert die Standortermittlung möglicherweise nicht mehr, weil die Validierung sich immer auf die zuletzt ermittelte Position und eine feste Strecke, die zwischen zwei Standortermittlungen realistisch gegangen werden kann, bezieht. Wird die Position wieder korrekt erkannt, kann es vorkommen, dass der/die NutzerIn bereits eine zu große Strecke zurückgelegt hat. Dann muss das System rekalibriert werden, was dieser Prototyp derzeit nicht unterstützt.

5) Cross-Platform

Die Applikation soll betriebssystemunabhängig sein. Daher wurde das Projekt als Webanwendung realisiert, welche über einen beliebigen Browser aufgerufen werden kann und somit die Anforderung Cross-Plattform erfüllt. Das Gerät muss lediglich einen Internetzugang sowie eine Kamera haben.

6) Anzeige der nächsten Abbiegung

Die Weginformationen werden im Frontend aufbereitet angezeigt und teilen NutzerInnen mit, wann und in welche Richtung als nächstes abgebogen werden muss.

7) Ziel erreicht

NutzerInnen wird angezeigt, wenn das Ziel erreicht wurde.

8) JSON-Format

Die Kommunikation zwischen Frontend und Backend geschieht über das JSON-Format.

Nicht Funktional

1) Echtzeit

Die Berechnung setzt sich aus der Standortermittlung sowie der Routenfindung zusammen. Die Anforderung war eine Berechnung in unter 0,5 Sekunden. Eine Durchschnittslaufzeit wurde in diesem Unterpunkt immer mit 50 Durchläufen ermittelt.

Die Auswertungslaufzeit inklusive vorbereitender Bildverarbeitung einer Standorterkennung wurde mit der Erweiterung *autotime* in Google Colab durchgeführt. Die Durchschnittslaufzeit betrug 70,02 Millisekunden auf einer Tesla T4 GPU. Lokal wurde dieses Model ebenfalls getestet und eine Laufzeit von 89,36 Millisekunden auf einem Intel(R) Core(TM) i7-8665U ermittelt.

Dabei wurde bei beiden Versuchen ein Transfer-Learning Model verwendet, welches mit der Bildgröße 320×180 arbeitet, da dieses die beste Genauigkeit in der Klassifizierung erzielte.

Die Laufzeit der Wegfindung variiert je nach Länge des Pfades, dauert aber im schlechtesten Fall im Durchschnitt¹ 8 Millisekunden. Auch hier wurden 50 Testläufe lokal auf der CPU durchgeführt.

Die beiden Auswertungslaufzeiten werden hier vor allem getrennt betrachtet, da die Durchführung lokal mangels GPU nicht mit der entsprechenden Hardware getestet werden konnte und somit nicht die minimale Rechenlaufzeit ermittelt werden kann. Im Rahmen dieses Projekts wurde jedoch kein Deployment auf eine Umgebung mit entsprechender Hardware realisiert.

Insgesamt ist die Auswertungsgeschwindigkeit rein auf der CPU mit im Schnitt 132,22 Millisekunden trotzdem schnell genug, um (je nach Latenz durch Netzwerkverbindung) eine Antwortzeit in weniger als 0,5 Sekunden zu realisieren. Das Frontend wartet im Durchschnitt 180,38 Millisekunden auf eine Antwort des Servers. Bei dieser Messung waren jedoch Client und Server in einem Netzwerk.

2) Kürzeste Route

Durch die Implementierung einer Breitensuche ist sichergestellt, dass immer die kürzeste Route gefunden wird [RN10].

¹Die Strecke von E109 nach A2.09 ist die längste in der FH (Siehe Abbildung 7.1 im Anhang)

	Datensatz	Anzahl Kategorien	korrekte Klassifizierung	Video-Quelle
1	Gesamt	355	46, 34%	AF.mp4
2	AF	115	56, 87%	AF.mp4
3	Gesamt	355	68, 46%	D0.mp4
4	D0	45	48, 16%	D0.mp4
5	Gesamt	355	46, 02%	A0.mp4
6	A0	23	91, 26%	A0.mp4
7	Gesamt	355	70, 14%	F.mp4
8	F	36	81, 71%	F.mp4

Tabelle 6.1: Training mit verschiedenen Hyperparametern

3) Genauigkeit der Standortermittlung mit dem Trainingsdatensatz

Das Model, welches mit Transfer Learning mit einer Bildgröße von 320×180 trainiert wurde, erzielte mit dem Testdatensatz eine Genauigkeit von über 98%, somit ist diese Anforderung ebenfalls erfüllt. Hierbei wird nur eine Aussage über die Standortermittlung an den Positionen gemacht, die zur Videoaufnahme ausgewählt wurden.

4) Genauigkeit der Standortermittlung im Betrieb

Um das zu validieren, wurde ein Video bei jedem Frame auf die ermittelten Standorte überprüft und mit den realen Positionen abgeglichen. Hierbei wurde einem Zeitabschnitt genau eine Position a manuell zugeordnet.

Als vom Model richtig klassifizierte Position gelten Positionen mit einem Abstand von bis zu zwei Positionen von a , die Blickrichtung muss aber identisch sein. Diese Abweichung der Position wird toleriert, da zum einen die manuelle Zuordnung ungenau sein kann und hier eine Bewertung jedes Frames stattfindet, zum anderen gilt die Klassifikation des Models als korrekt, wenn a zwischen zwei Aufnahmepositionen liegt, und das Ergebnis des Models eine von beiden Positionen ist.

In Tabelle 6.1 sind die Ergebnisse mit verschiedenen Testvideos zu sehen.

Die Spalte *Datensatz* beschreibt, mit welchem Datensatz das verwendete Model trainiert wurde. In Zeile 1 wurde der gesamte Datensatz verwendet, das heißt, das Model kann alle 355 Kategorien bestimmen. In Zeile 2 kann das Model nur Kategorien der Gebäudeteile A und F erkennen. Die Experimente basieren auf einem Video, dessen Titel in der Spalte *Video-Quelle* angegeben ist. Die Videos befinden sich im Verzeichnis *test_videos*.

Der erste Versuch wurde mit vielen zu klassifizierenden Positionen ausgeführt - 39 an der Zahl, nämlich von Position F0043 bis A2082. Da dieses Ergebnis mit einer korrekten

6 Ergebnis

Klassifikation von 46,34% bei Weitem nicht genau genug ist, wurde ein Model nur mit den Kategorien aus Gebäudeteil A und F trainiert. Eine leichte Verbesserung auf 56,87% konnte dabei erzielt werden. Daher wurden sechs weitere Versuche auf Basis der Videos D0.mp4, A0.mp4 und F.mp4 durchgeführt.

D0 wurde ausgewählt, da der Gang besonders breit ist, dieser aber nur der Länge nach gerastert ist, jedoch nicht der Breite nach, sodass es hier zu Ungenauigkeiten kommen könnte, wenn die Aufnahmen nicht exakt mittig gemacht werden. In Zeile 3 konnte das Model eine Genauigkeit von 68,46% erreichen, mit einem kleineren Datensatz wurde, anders als antizipiert und in Versuch 2 gesehen, sogar ein schlechteres Ergebnis mit 48,16% erzielt.

A0 wurde ausgewählt, da dies ein schmaler Gang mit großen Ähnlichkeiten zu A1 und A2 ist. Mit 46,02% richtigen Klassifizierungen ist die Performance ähnlich gut wie in Versuch 1. Wird allerdings ein Model, welches nur mit dem Datensatz A0 trainiert wurde, verwendet, erhöht sich dieser Wert signifikant auf 91,26%. Mit einer Reduzierung der zu erkennenden Klassen geht also wie auch bei Versuch 2 eine bessere Genauigkeit einher.

F wurde gewählt, da dort testweise eine Rasterung vorgenommen wurde, die mit circa 1,5 Metern kleiner als die im übrigen Gebäude ist. Hierbei wurde schon im Versuch mit dem Model, welches mit dem gesamten Datensatz trainiert wurde, ein mit 70,14% überdurchschnittliches Ergebnis erzielt. Mit einem Training, welches nur die Trainingsdaten aus dem Gebäudeteil F enthält, konnte ein Model eine noch bessere Genauigkeit von 81,71% erreichen.

Bei drei von vier Versuchen ließ sich eine Verbesserung der Genauigkeit durch die Reduktion der zu klassifizierenden Kategorien erzielen. Die deutlichste Verbesserung konnte mit der geringsten Anzahl Kategorien erreicht werden.

6) Sinnvolle Informationen

Die angezeigten Informationen beschränken sich auf die nächste Richtungsinformation, also in wie viel Metern in welche Richtung abgebogen werden soll beziehungsweise ob eine Treppe hinauf- oder herabgestiegen werden muss. Diese werden mit einer Distanz mit einer Genauigkeit von circa 2,5 Metern angegeben. In wenigen Fällen ist diese Genauigkeit nicht eindeutig, sodass aus mehreren Räumen das tatsächliche Ziel ausgewählt werden muss, da diese zu nah beieinander liegen. Die Informationen

6 Ergebnis

können alle notwendigen Sachverhalte widerspiegeln, es wurde aber keine Umfrage mit TestnutzerInnen durchgeführt.

7) Minimale Datenübertragung

Eine weitere Anforderung war eine minimale Datenübertragung. Das beste Resultat wurde mit einer Auflösung von 320×180 erreicht. Das Transfer Learning Model benötigt das Bild in Farbe. Daher bewegt sich die Größe eines Datenpaketes je nach Kodierung des Bildes zwischen 60.000 und 75.000 Byte. Ein Bild in Originalgröße bewegt sich hingegen in der 10-fachen Größenordnung. Durch die Bildübertragung müssen viele Daten übertragen werden, im Vergleich zur Originalaufnahme konnten allerdings deutlich Daten eingespart werden. Es ist jedoch nicht auszuschließen, dass die Datenübertragung hinsichtlich der Quantität weiter optimiert werden kann.

8) Benutzerfreundlichkeit

Da keine Umfrage durchgeführt wurde, kann die Benutzerfreundlichkeit nicht abschließend geklärt werden. Die Einfachheit der Applikation kann dennoch bewertet werden. Die einzige Einstellungsmöglichkeit ist gegeben durch das Dropdown-Menü. Anschließend werden alle nötigen Informationen angezeigt, wie etwa das Schwenken für eine genauere initiale Standortermittlung. Die Applikation funktioniert aber auch, wenn der Nutzer oder die Nutzerin die Anweisung ignoriert. Anschließend müssen die Navigationsinformationen befolgt werden. Die Güte dieser Information ist an dieser Stelle durch andere Anforderungen, wie *1) Sinnvolle Informationen* abgedeckt.

Durch die minimale Interaktionsmöglichkeit wird für die Nutzung der Anwendung keine Einführung benötigt. Zudem sind alle gängigen Raumbezeichner vorhanden. Der Raum A2.06 beispielsweise kann sowohl als A2.06, SR6 als auch Aristoteles im Dropdown-Menü gefunden und ausgewählt werden.

6.3 Anforderungen Augmented Reality

Funktional

1) Objekterkennung

Eine gute Objekterkennung, hier mit besonderem Fokus auf dem Erkennen des Bodens, ist für die korrekte Markerberechnung notwendig. Um diese zu beurteilen, wurde ein Video erstellt, welches eine Originalaufnahme aus der Fachhochschule zeigt, daneben die erkannte Segmentierung. Es ist im Root-Verzeichnis der Applikation im Ordner *DemoVideos* zu finden und trägt die Bezeichnung *SegmentationDemo*. Dabei wurde in jedem Frame das Ursprungsbild segmentiert und das Ergebnis konateniert, sodass beide Bilder simultan zu sehen sind. Das Ursprungsvideo besitzt eine Auflösung von 270×480 .

Hierbei wird deutlich, dass insbesondere die Erkennung des Bodens, also der Hauptbestandteil der Markerberechnung, sehr gut funktioniert, andere Objekte werden teilweise weniger gut erkannt, denn die Segmentierung unterscheidet sich von Frame zu Frame. Ein Screenshot aus dem Demo-Video ist in Abbildung 6.2 zu sehen.

2) Korrekte Symbole

Die angezeigten Symbole müssen zur berechneten Route passen. Auch hier wurde zur Evaluierung des Ergebnisses ein Demo-Video mit dem Titel ARDemoFake erstellt. Die Symbole wurden im Backend per OpenCV auf das Bild gerendert, das heißt, hier wurde nicht die Frontend-Funktionalität zum Zeichnen verwendet, sondern lediglich simuliert. Die Positionsermittlung, die für die Markerberechnung notwendig ist, wurde manuell durchgeführt. Das Video zeigt, dass die Pfeile in die richtige Richtung zeigen.

3) Berechnung der Positionen der anzuzeigenden Symbole

In dem zuvor erwähnten Video lässt sich ebenfalls entnehmen, dass die Positionierung korrekt ist. Die Marker sind perspektivisch platziert und der Marker, der die Zieltür anzeigen, tut dies an geeigneter Stelle, sodass NutzerInnen den entsprechenden Raum finden können.

4) JSON-Format

Die Kommunikation zwischen Frontend und Backend geschieht über das JSON-Format.

6 Ergebnis

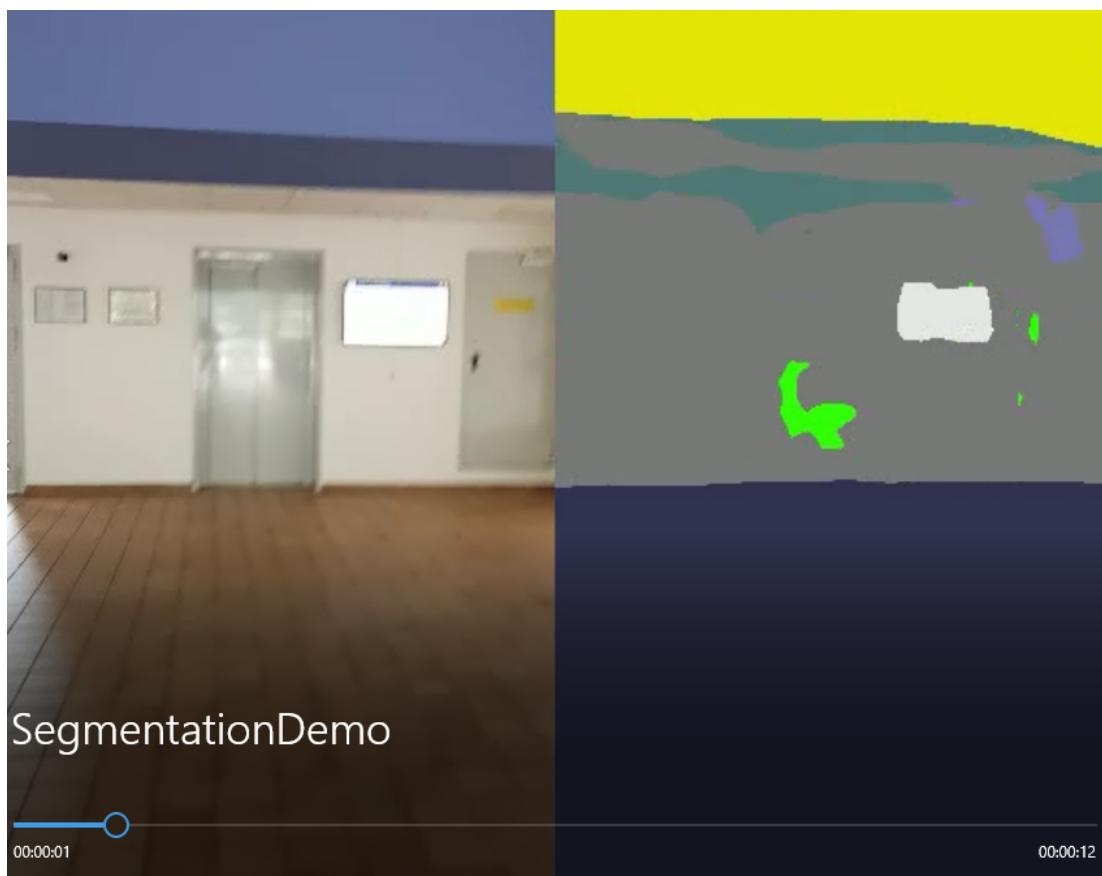


Abbildung 6.2: Demo-Video der Segmentierung

Nicht funktional

1) Echtzeit

Eine generelle Anforderung an Augmented Reality-Anwendungen ist, dass diese in Echtzeit genutzt werden kann. Dies soll auch in diesem Projekt umgesetzt werden.

Aus Grafik 5.5 ließ sich entnehmen, dass eine genügend gute Segmentierung schon ab einer Bildgröße von 270×480 Pixeln möglich ist. Daher wurde bei einer Auswertung nur diese Größe in Betracht gezogen, weil darüber hinaus keine weiteren notwendigen Informationen aus der Segmentierung gewonnen werden konnten.

Um den Unterschied einer Nutzung einer CPU und einer GPU bezüglich der Nutzung eines Machine Learning Models zu verdeutlichen, wurde sowohl eine Messung der Segmentierung lokal auf der CPU als auch auf Google Colab mit einer GPU durchgeführt. Die Berechnung mit CPU lag im Schnitt bei 1,91 Sekunden, während die Berechnung mit einer GPU 0,02928 Sekunden, also 29,28 Millisekunden dauerte. Auch hier wurden 50 Durchläufe durchgeführt.

Die restliche Bildverarbeitung auf der CPU dauerte im Durchschnitt 14,8 Millisekunden. Bei der Nutzung einer GPU zur Berechnung der Segmentierung würde die Berechnung der Markerpositionen insgesamt im Durchschnitt 44,08 Millisekunden benötigen.

Damit wären 22,686 Frames pro Sekunde (FPS) möglich, die Übertragungslatenz von Frontend zu Backend und zurück wird hierbei zunächst ausgelassen.

Für eine Framerate von 30 FPS muss die Rechenzeit pro Berechnung kleiner als 34 Millisekunden pro Berechnung sein. Mit einer Bildgröße von 180×320 dauert die Segmentierung auf Google Colab circa 17,48 Millisekunden, die gesamte Berechnung inklusive der weiteren Bildverarbeitung dauert entsprechend 32,27 Millisekunden. Damit wären 30 FPS realisierbar. Hierbei ist die Bildgröße jedoch so klein, dass etwa Türen nicht mehr erkannt werden können, wobei die Entfernung zu einer Tür und somit die Position eines Markers auch mittels geometrischer Berechnungen ermittelt werden kann und dies derzeit auch die Grundlage der benutzen Berechnung ist.

Die Segmentierung konnte bisher nicht live während einer Navigation getestet werden, da sie über Google Colab nicht in die Anwendung eingebunden wurde. Daher lässt sich auch nicht einschätzen, ob für den Anwendungsbereich der Navigation wirklich eine Framerate von 30 FPS notwendig ist oder ob eine geringere Aktualisierungsfrequenz ausreicht.

2) Ausreichende Genauigkeit

Die Symbole, die angezeigt werden können, sind ausreichend, um alle nötigen Sachverhalte widerzuspiegeln. Teilweise ist die Standortlokalisierung jedoch zu ungenau für eine gute Markerberechnung. In Video ARDemo würde ein Nutzer oder eine Nutzerin das Ziel vermutlich trotzdem mit der angezeigten Information finden. Hierbei wurde jedoch eine korrekte Standortermittlung simuliert.

6.4 Demo-Video mit Standorterkennung und AR

Es wurde auch ein Demo-Video mit dem Titel FullDemo erstellt, welches die Standorterkennung, Routenfindung und AR zusammen nutzt. Dabei ist ersichtlich, dass die Standorterkennung teilweise sehr ungenau ist und daher eine Berechnung der AR-Komponenten nicht korrekt möglich ist. Die ungenauen ermittelten Standorte stammen vermutlich von den Aufnahmen zwischen zwei Positionen, oder wenn die Aufnahme nicht direkt von in der Mitte des Gangs ausging, sondern einen Bildausschnitt weiter links oder rechts zeigt. Daher wackeln die Marker teilweise stark, weswegen das Anschauen der Bildaufnahme bei geringerer Framerate angenehmer ist, da sie das Wackeln der Marker reduziert.

6.5 Erkenntnisse

Durch die verschiedenen Versuche mit Bildverarbeitungsmethoden und Machine Learning konnten einige Erkenntnisse gewonnen werden.

1. Für reine Bildverarbeitung ist eine höhere Auflösung nötig als bei der Segmentierung.
2. Transfer Learning hilft, die Genauigkeit eines Models mit den gleichen Trainingsdaten bei ähnlicher Trainingszeit signifikant zu verbessern.
3. Die Standortlokalisierung zwischen verschiedenen Positionen des Models ist verbesserungswürdig, zum Beispiel durch eine Reduzierung der Anzahl der Kategorien, die erkannt werden müssen.
4. Objekterkennung mit reiner Bildverarbeitung zu realisieren, ist auch bei sehr geometrischen Objekten durch ungünstige Lichtverhältnisse instabil.
5. Je mehr Positionen innerhalb des Gebäudes erkannt werden müssen, desto mehr Bilddaten sind für die Klassifizierung notwendig

Insgesamt erfüllt die Anwendung zu großen Teilen ihren Zweck und die gestellten Anforderungen. Die Augmented Reality-Komponente funktioniert mit der Segmentierung sehr gut und erkennt trotz unterschiedlicher Lichtverhältnisse stabil den Boden. Die Standorterkennung funktioniert durch die Filterung, Validierung und das Beibehalten des letzten Standortes größtenteils, kann jedoch noch weiter optimiert werden.

7

Zusammenfassung und Ausblick

Dieses Kapitel gibt zuerst eine Zusammenfassung über die Praxisleistung dieser Arbeit. Anschließend werden Verbesserungsmöglichkeiten und mögliche weitere Funktionalitäten beleuchtet.

7.1 Zusammenfassung

In dieser Arbeit wurde das Indoor-Navigationssystem IndoorNav entwickelt, welches mit Machine Learning die Position von NutzerInnen ermittelt und diesen über AR und in Schriftform die nötigen Weginformationen liefert. Die Benutzeroberfläche ist als Webapplikation in React realisiert, während das Backend mit Python und Tensorflow arbeitet. Die Augmented Reality-Komponente nutzt ebenfalls Machine Learning, speziell Bildsegmentierung zur Objekterkennung, sowie weitere Bildverarbeitungsmethoden wie Kantenerkennung, Linienerkennung oder die Berechnung einer konvexen Hülle. Methoden zur Bildverarbeitung, die grundlegende Funktionsweise von Machine Learning oder verschiedene Herangehensweisen zur Indoor Navigation werden in Kapitel 2 geschildert. Anforderungen an Indoor Navigation und AR sowie verwandte Arbeiten werden in Kapitel 3 und 4 genannt. In Kapitel 5 sind unter anderem Architektur, Implementierungsdetails und Algorithmen erläutert. Daneben werden Vergleiche zwischen Models mit vor allem verschiedenen Hyperparametern, aber auch verschiedenen Architekturen sowie verschiedenen Implementationen zur Objekterkennung gezogen, um die optimalen Hyperparameter und die bessere Objekterkennungsmethode zu identifizieren. Dabei stellte sich Transfer Learning als gut geeignete Trainingsmöglichkeit heraus, sodass ein Model mit einer Genauigkeit auf dem Trainingsdatensatz von mehr als 98% erzeugt werden konnte. Auch die Bodenerkennung konnte mit einem Segmentierungsmodel robust realisiert werden. In Kapitel 6 ist das Ergebnis dieses Projekts besonders hinsichtlich der formulierten Anforderungen dargestellt.

7.2 Verbesserungsmöglichkeiten

In diesem Projekt wurde eine erste Version des Indoor-Navigationssystems entwickelt. Daher gibt es viele Optimierungsmöglichkeiten, die Verbesserungen ermöglichen und im Rahmen weiterer Arbeiten untersucht werden können.

Die Validierung der Position ist bisher sehr einfach gehalten, indem nur eine bestimmte Anzahl an Positionen zwischen zwei Standortermittlungen übersprungen werden dürfen, betrachtet aber nicht den Zeitpunkt der Berechnungen. Des Weiteren kann eine erneute initiale Standortlokalisierung mit Schwenken der Kamera implementiert werden. Auch können andere Informationsquellen herangezogen werden, wie etwa die Entfernung zu einem Router, um mögliche Fehlklassifizierungen zu erkennen und eine Rekalibrierung des Systems zu ermöglichen.

Außerdem können die Trainingsdaten aufgestockt werden, zum Beispiel bei anderen Tageszeiten, Lichtverhältnissen oder Jahreszeiten sowie anderem oder anders platziertem Mobiliar. Somit kann die Standorterkennung weiter stabilisiert werden. Außerdem wäre eine Verfeinerung der Positionen, die ein Model erkennt, denkbar, sodass statt 2,5 Metern alle 1,5 oder 1 Meter Aufnahmen gemacht werden. In breiten Gängen können auch Bildaufnahmen von weiter rechts und weiter links im Gang aufgenommen werden. Auch die Auswirkung der verwendeten Kamera kann untersucht werden, um kameraspezifische Abweichungen in der Standortermittlung ausschließen zu können.

Daneben können weitere Architekturen für Models untersucht werden, ob diese eine noch bessere Performance gegenüber dem derzeit am besten funktionierenden Model liefern können. Das muss nicht nur die interne Architektur mit verschiedenen Layern sein. Die Untersuchung kann auch die Nutzung mehrerer Modelle beinhalten, die, wenn die möglichen zu erkennenden Kategorien gleichmäßig auf diese Modelle aufgeteilt werden, ebenfalls eine hohe Genauigkeit der Klassifizierung erzielen, wie dies in Kapitel 6 festgestellt wurde. Dies kann zum Beispiel mit Markern wie etwa QR-Codes realisiert werden, die im Input-Bild erkannt und identifiziert werden. Anhand dieser kann dann das richtige Model, zum Beispiel eines für Gebäudeteil A, ausgewählt und verwendet werden. Solche Marker können ebenfalls zur Rekalibrierung und Validierung verwendet werden.

Auch die Implementierung der AR-Komponente bietet Raum für Verbesserungen und weitere Untersuchungen. Darunter fällt zum Beispiel die Bereitstellung einer GPU für die Segmentierungsberechnung, sodass diese auch in Echtzeit im Betrieb durchgeführt werden kann. Daneben kann untersucht werden, ob durch andere oder weitere Bildverarbeitungsmaßnahmen eine Detektierung der relevanten Objekte mit reinen Bildverarbeitungsmaßnahmen ausreichend ist oder die Berechnung der Ausrichtung, Skalierung und Position der Icons für die Anzeige der Navigationsinformationen direkt ins Frontend ausgelagert werden kann.

Die Anwendung soll idealerweise auch im Multiuserbetrieb nutzbar und über eine URL im Internet oder Netzwerk der Hochschule erreichbar sein, damit mehrere NutzerInnen die Navigation gleichzeitig verwenden können.

7.3 Weitere Funktionalitäten

Neben den Verbesserungen kann die Anwendung auch noch um weitere Funktionalitäten ergänzt werden, wie etwa das Miteinbeziehen von weiteren Gebäuden der Hochschule, welche in dem entwickelten Prototypen nicht betrachtet wurde. Auch die Berechnung der Zeit bis zum Ziel kann anhand der Beschleunigungssensoren innerhalb eines Mobiltelefons berechnet werden. Außerdem kann die Routenfindung um die Ermittlung eines barrierefreien Wegs ohne Treppen beziehungsweise mit Fahrstuhl ergänzt werden.

Auch Informationen zum Zielraum, wie beispielsweise welche Vorlesungen wann in einem bestimmten Hörsaal stattfinden oder wie Sprechzeiten oder Telefonnummern von Mitarbeitenden sind, könnten angezeigt werden oder das System wird direkt in die offizielle FH Wedel App eingebunden.

Zudem kann die gesamte Applikation als eine App ohne Backendservice erprobt werden. Somit müssen keine Bilddaten versendet werden, allerdings muss dafür eine Anwendung auf dem entsprechenden Gerät installiert werden.

Eine andere Berechnung der Marker über Surface Detection wie in der Arbeit von Karan et al [KMSF21] ist ebenfalls denkbar.

7 Zusammenfassung und Ausblick

Diese genannten weiteren Funktionalitäten verbessern die Anwendung. Ungeachtet dessen beherrscht der erste Prototyp von IndoorNav für die Fachhochschule Wedel die geforderte Grundfunktionalität der AR-unterstützten Navigation und kann durch weitergehende Optimierungen und Untersuchungen in der Praxis realisiert werden.

Literaturverzeichnis

- [ANK18] Jafar Alzubi, Anand Nayyar, and Akshi Kumar. Machine learning from theory to algorithms: An overview. *Journal of Physics: Conference Series*, 1142:012012, nov 2018.
- [ARF13] Amer Al-Rahayfeh and Miad Faezipour. Enhanced frame rate for real-time eye tracking using circular hough transform. In *2013 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pages 1–6, 2013.
- [BD20] Björn Barz and Joachim Denzler. Do we train on test data? purging cifar of near-duplicates. *Journal of Imaging*, 6(6), 2020.
- [BPEET20] Gustav Burström, Oscar Persson, Erik Edström, and Adrian Elmpterander. Augmented reality navigation in spine surgery: a systematic review. 2020.
- [BTVG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. volume 3951, pages 404–417, 07 2006.
- [Cho] François Chollet. Transfer learning and fine-tuning. https://www.tensorflow.org/tutorials/images/transfer_learning. Accessed: 2022-15-08.
- [Clo] Morphological transformations. https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html. Accessed: 2022-07-27.
- [CLZ⁺20] Wei Cui, Qingde Liu, Linhan Zhang, Haixia Wang, Xiao Lu, and Junliang Li. A robust mobile robot indoor positioning system based on wi-fi. *International Journal of Advanced Robotic Systems*, 17:172988141989666, 01 2020.
- [Col] Willkommen bei colaboratory. <https://colab.research.google.com/notebooks/intro.ipynb?hl=de>. Accessed: 2022-07-050.
- [Cor07] T.H. Cormen. *Algorithmen - eine Einführung*. Oldenbourg, 2007.

- [EFSd19] Yassir El Filali and Krit Salah-ddine. Augmented reality types and popular use cases. pages 91–97, 04 2019.
- [FC20] Giovanni Fusco and James Coughlan. Indoor localization for visually impaired travelers using computer vision on a smartphone. volume 2020, pages 1–11, 04 2020.
- [Fia05] M. Fiala. Artag, a fiducial marker system using digital techniques. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 590–596 vol. 2, 2005.
- [Fou] Python Software Foundation. Python. <https://www.python.org/>. Accessed: 2022-07-050.
- [Gí7] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow concepts, tools, and techniques to build intelligentsystems*. O'Reilly Media, 1 edition, 2017.
- [GG87] Judy Green and Paul Green. Alberti's perspective: A mathematical comment. *The Art Bulletin*, 69:641, 12 1987.
- [GM] Vikas Gupta and Anastasia Murzova. Image classification using convolutional neural networks in keras. <https://learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/>. Accessed: 2022-10-08.
- [Gmb] Statista GmbH. Das 9-euro-ticket. https://de.statista.com/themen/9462/9-euro-ticket/#topicHeader_wrapper. Accessed: 2022-19-08.
- [HG10] Haosheng Huang and Georg Gartner. *A Survey of Mobile Indoor Navigation Systems*, pages 305–319. 01 2010.
- [HHCW20] Bo-Chen Huang, Jiun Hsu, Edward T.-H. Chu, and Hui-Mei Wu. Arbin: Augmented reality based indoor navigation system. *Sensors*, 20(20), 2020.
- [HK10] Shuang Hua Yang Hakan Koyuncu. A survey of indoor positioning and object locating systems. *IJCSNS International Journal of Computer Science and Network Security*, VOL.10 No.5, 10, 5 2010.

- [IK88] J. Illingworth and J. Kittler. A survey of the hough transform. *Computer Vision, Graphics, and Image Processing*, 44(1):87–116, 1988.
- [Ink] InkHunter. InkHunter. <http://www.inkhunter.tattoo/>. Accessed: 2022-08-08.
- [KB99] H. Kato and M. Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, pages 85–94, 1999.
- [Kee17] Brian O Keefe. Finding location with time of arrival and time difference of arrival techniques. *ECE Senior Capstone Project 2017 Tech Notes*, 2017.
- [Ker] Keras. <https://keras.io/about/>. Accessed: 2022-07-05.
- [KKAmAA19] Jayakanth Kunhoth, AbdelGhani Karkar, Somaya Al-ma’adeed, and Asma Al-Attiyah. Comparative analysis of computer-vision and ble technology based indoor navigation systems for people with visual impairments. *International journal of health geographics*, 18:29, 12 2019.
- [KKAmAA20] Jayakanth Kunhoth, AbdelGhani Karkar, Somaya Al-ma’adeed, and Abdulla Al-Ali. Indoor positioning and wayfinding systems: a survey. *Human-centric Computing and Information Sciences*, 10, 12 2020.
- [KMSF21] Esin Karan, Muhammed Morkoc, Ahmet Sireci, and Kemal Fidanboylu. Indoor and outdoor navigation application development with augmented reality technology. pages 211–220, 06 2021.
- [KUN19] Dawar Khan, Sehat Ullah, and Syed Nabi. A generic approach toward indoor navigation and pathfinding with robust marker tracking. *Remote Sensing*, 11:3052:1–3052:22, 12 2019.
- [Lin12] Tony Lindeberg. Scale invariant feature transform. *Scholarpedia*, 7:10491, 2012.
- [LLC] Google LLC. Google maps. <https://www.google.com/maps/>. Accessed: 2022-08-08.

- [MP] Inc. Meta Platforms. React - a javascript library for building user interfaces. <https://reactjs.org/>. Accessed: 2022-07-05.
- [NAHF20] Ahasanun Nessa, Bhagawat Adhikari, Fatima Hussain, and Xavier N. Fernando. A survey of machine learning for indoor positioning. *IEEE Access*, 8:214945–214965, 2020.
- [Nia] Niantic. Pokémon go. <https://pokemongolive.com/en/>. Accessed: 2022-08-08.
- [NM15] Azad Noori and Farzad Moradi. Simulation and comparison of efficiency in pathfinding algorithms in games. *Ciência e Natura*, 37:230, 12 2015.
- [Pal] Pallets. Flask. <https://palletsprojects.com/p/flask/>. Accessed: 2022-07-050.
- [PM17] David L. Poole and Alan K. Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, USA, 2nd edition, 2017.
- [PSPS11] David Prochazka, Michael Stencl, Ondrej Popelka, and Jiri Stastny. Mobile augmented reality applications. *CoRR*, abs/1106.5571, 2011.
- [QCYL20] Jiuchao Qian, Yuhao Cheng, Rendong Ying, and Peilin Liu. A novel indoor localization method based on image retrieval and dead reckoning. *Applied Sciences*, 10:3803, 05 2020.
- [RCG18] Richard E. Woods Rafael C. Gonzalez. *Digital Image Processing, 4th Edition*. Pearson Education Limited, 2018.
- [RN10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.
- [RRG⁺20] Nida Romli, Amir Razali, Nur Hafizah Ghazali, Nik Adilah Hanin Binti Zahri, and Siti Ibrahim. Mobile augmented reality (ar) marker-based for indoor library navigation. *IOP Conference Series: Materials Science and Engineering*, 767:012062, 03 2020.
- [Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

- [SHZ⁺18] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.
- [Ste] Kim Steuerwald. Mit der ikea app per augmented reality einrichten. <https://www.ikea.com/de/de/this-is-ikea/corporate-blog/ikea-place-app-augmented-reality-puba55c67c0>. Accessed: 2022-08-08.
- [Tena] Tensorflow. Eine durchgängige open-source-plattform für maschinelles lernen. <https://www.tensorflow.org/>. Accessed: 2022-27-08.
- [Tenb] Tensorflow. Save and load keras models. https://www.tensorflow.org/guide/keras/save_and_serialize. Accessed: 2022-27-08.
- [Tenc] Tensorflow. tf.keras.utils.image_dataset_from_directory. https://www.tensorflow.org/api_docs/python/tf/keras/utils/image_dataset_from_directory. Accessed: 2022-18-08.
- [Vis] MIT CSAIL Computer Vision. Semantic segmentation on mit ade20k dataset in pytorch. <https://github.com/CSAILVision/semantic-segmentation-pytorch>. Accessed: 2022-18-07.
- [WFW20] Hong Wu, Zijian Fu, and Yizhou Wang. A medical network clustering method with weighted graph structure. *Measurement and Control*, 53(9-10):1751–1759, 2020.
- [YMY18] Shuxia Yang, Bing Mei, and Xiaoyu Yue. Mobile augmented reality assisted chemical education: Insights from elements 4d. *Journal of Chemical Education*, 95, 04 2018.
- [ZLLS21] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *CoRR*, abs/2106.11342, 2021.
- [ZQD⁺21] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2021.
- [ZZP⁺17] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings*

Literaturverzeichnis

of the IEEE Conference on Computer Vision and Pattern Recognition,
2017.

- [ZZP⁺18] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal on Computer Vision*, 2018.

Anhang

Versionsnummern

Die package.json-Datei enthält alle Versionsnummern der im Frontend verwendeten Abhängigkeiten im JSON-Format. Diese sind in Listing 7.1 zu sehen.

```
1 {
2   "name": "indoor-nav",
3   "version": "0.1.0",
4   "private": true,
5   "dependencies": {
6     "@ant-design/icons": "^4.7.0",
7     "@fortawesome/fontawesome-svg-core": "^6.1.1",
8     "@fortawesome/free-solid-svg-icons": "^6.1.1",
9     "@fortawesome/react-fontawesome": "^0.1.18",
10    "@react-icons/all-files": "^4.1.0",
11    "@testing-library/jest-dom": "^5.16.3",
12    "@testing-library/react": "^12.1.4",
13    "@testing-library/user-event": "^13.5.0",
14    "antd": "^4.19.3",
15    "react": "^18.0.0",
16    "react-dom": "^18.0.0",
17    "react-dropdown-select": "^4.8.4",
18    "react-scripts": "5.0.0",
19    "react-select": "^5.3.2",
20    "react-webcam": "^7.0.0",
21    "redux-thunk": "^2.4.1",
22    "web-vitals": "^2.1.4"
23  }
24 }
```

Listing 7.1: package.json

Listing 7.2 zeigt die `environment.yml`, welche alle Abhängigkeiten der Backend-Anwendung enthält.

```
1 name: indoor_nav
2 channels:
3   - conda-forge
4   - defaults
5 dependencies:
6   - aom=3.3.0
7   - asttokens=2.0.5
8   - backcall=0.2.0
9   - backports=1.0
10  - backports.functools_lru_cache=1.6.4
11  - blosc=1.21.1
12  - bzip2=1.0.8
13  - c-blosc2=2.1.1
14  - ca-certificates=2022.5.18.1
15  - cfitsio=4.1.0
16  - charls=2.3.4
17  - cloudpickle=2.1.0
18  - colorama=0.4.4
19  - cytoolz=0.11.2
20  - dask-core=2022.5.0
21  - debugpy=1.6.0
22  - decorator=5.1.1
23  - entrypoints=0.4
24  - executing=0.8.3
25  - freetype=2.10.4
26  - fsspec=2022.5.0
27  - giflib=5.2.1
28  - imagecodecs=2022.2.22
29  - imageio=2.19.2
30  - intel-openmp=2022.1.0
31  - ipykernel=6.13.0
32  - ipython=8.4.0
33  - jbig=2.1
34  - jedi=0.18.1
35  - jpeg=9e
36  - jupyter-client=7.3.1
```

```
37 - jupyter_core=4.10.0
38 - jxrlib=1.1
39 - krb5=1.19.3
40 - lcm_s2=2.12
41 - lerc=3.0
42 - libaec=1.0.6
43 - libavif=0.10.1
44 - libblas=3.9.0
45 - libbrotlicommon=1.0.9
46 - libbrotlidec=1.0.9
47 - libbrotlienc=1.0.9
48 - libcblas=3.9.0
49 - libcurl=7.83.1
50 - libdeflate=1.10
51 - libffi=3.4.2
52 - liblapack=3.9.0
53 - libpng=1.6.37
54 - libsodium=1.0.18
55 - libssh2=1.10.0
56 - libtiff=4.3.0
57 - libwebp=1.2.2
```

Listing 7.2: environment.yml

Literaturverzeichnis

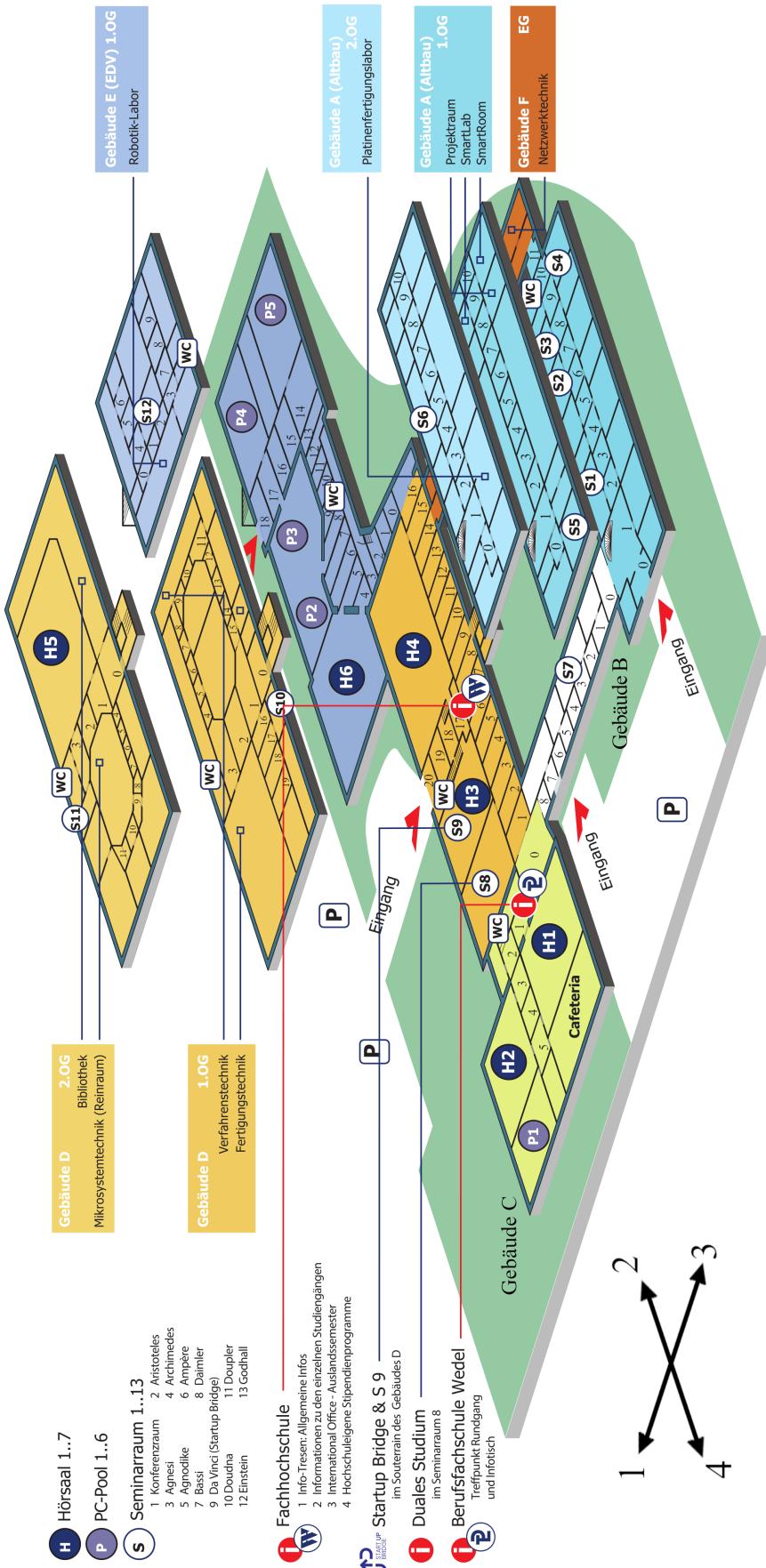


Abbildung 7.1: Gerasterter vollständiger Lageplan

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Ort, Datum

Mara Pape