Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Interaktion mit der MBP IoT-Plattform mittels der Microsoft HoloLens

Felix Scheerer

Studiengang:	Softwaretechnik
Prüfer/in:	Dr. rer. nat. Pascal Hirmer
Betreuer/in:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Beginn am:	15. April 2021
Beendet am:	15. Oktober 2021

Kurzfassung

Das Internet of Things gewinnt sowohl für Unternehmen als auch für Privatpersonen stetig an Relevanz. Ein großes Problem ist hierbei jedoch die Heterogenität der in IoT-Umgebungen verwendeten Systeme. Speziell im Smart-Home-Bereich äußert sich dies oft durch die Voraussetzung, für jedes Gerät eine weitere App installieren zu müssen. Die Multi-purpose Binding and Provisioning Platform (MBP) versucht, dieses Problem zu lösen, indem sie als Middleware für eine Vielzahl an kommerziellen als auch selbstentwickelten IoT-Geräten agiert.

Augmented Reality (AR) gerät speziell durch den Erfolg von Smartphone AR immer mehr als neues Visualisierungs- und Interaktionswerkzeug in den Fokus. So bieten AR-Brillen wie beispielsweise die Microsoft HoloLens das mobile Anzeigen von und die Interaktion mit Inhalten ohne konstantes Blockieren der Hände des Nutzers. Dies kann oft ein entscheidender Vorteil gegenüber Handheld AR wie Smartphones und Tablets darstellen, z. B. für industrielle Anwendungen.

In dieser Arbeit sollte am Beispiel der HoloLens untersucht werden, inwieweit sich AR-Brillen eignen, mit der MBP zu interagieren. Zu diesem Zweck sollte ein Prototyp entwickelt werden, der die wichtigsten Funktionen der MBP abdeckt, wie z. B. das Steuern von Aktuatoren, Auslesen von Sensorwerten und Überwachen der angeschlossenen IoT-Geräte. Dabei deckt diese Arbeit den Hintergrund zu IoT- und AR-Technologie ab und gibt eine ausführliche Betrachtung der Literatur auf der Schnittmenge beider Gebiete. Anhand der dabei gewonnen Erkenntnisse wurde eine prototypische Anwendung erstellt. Die Spiele-Engine Unity wurde mithilfe des Microsoft Mixed Reality Toolkits für die Entwicklung und QR-Codes als Tracking Marker verwendet. Das im Prototyp gesteuerte IoT ist ein Smart-Home-Szenario bestehend aus Raspberry Pis mit verschiedenen Sensoren und Aktuatoren, einer Bosch XDK IoT-Plattform und einem Xiaomi MiFlora-Pflanzensensor.

Zusammenfassend zeigte sich, dass die HoloLens für die Interaktionen mit der MBP gut geeignet ist. So lassen sich Sensorwerte und Monitoring-Daten anzeigen und Aktuatoren steuern, die mit der MBP verbunden sind. Trotzdem sind die Rechenleistung, geringe Akkulaufzeit und der Tragekomfort in manchen Situationen Nachteile der aktuellen HoloLens Generation.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Internet of Things (IoT) ist kein neuer Begriff, doch erst seit den letzten Jahren und speziell durch den Erfolg von Smart Homes und Industrie 4.0 hat dieser für private und industrielle Nutzer an Relevanz gewonnen [SGA19]. Aus den vielen verschiedenen Herstellern, die Produkte auf diesem Gebiet anbieten, folgt die hohe Heterogenität der Systeme. Für ein Smart-Home-Szenario hat dies zur Folge, dass ein Nutzer eine Vielzahl an Apps benötigt, um alle Geräte zu steuern [WRS+15]. Aus diesem Grund existieren IoT-Plattformen, wie die *Multi-purpose Binding and Provisioning Platform* (MBP). Hier können viele verschiedene kommerzielle oder selbstentwickelte IoT-Komponenten verbunden werden, um zentral gesteuert zu werden. Bisher wird dies oft über Apps oder Webinterfaces umgesetzt [BRS19].

Auch Augmented Reality (AR) ist an sich kein neues Themengebiet und existiert bereits seit den 1990er Jahren [DM91]. Doch durch die Verbreitung von Smartphone AR in den letzten Jahren ist der Begriff auch bei Endnutzern bekannter [AA17]. Augmented Reality eröffnen neuartige Möglichkeiten zur Anzeige von Informationen und zur Interaktion mit Computer-Systemen. AR-Brillen bieten gegenüber Handheld AR den Vorteil, keine freie Hand zu benötigen, um Informationen selbst einem sich bewegenden Nutzer konstant anzeigen zu können [PFP+17].

Das Ziel dieser Arbeit ist es, zu beantworten, ob ein Zusammenspiel aus IoT und AR möglich ist und ob sich so die MBP mithilfe einer AR-Brille wie der Microsoft HoloLens steuern lässt. Hierzu soll ein Prototyp entwickelt werden, der die grundlegenden Funktionen der MBP, wie das Auslesen von Sensorwerten, Steuern von Aktuatoren und Überwachen der IoT-Geräte, unterstützt. Anhand einer Literaturrecherche soll der aktuelle Stand der Technik auf dem Gebiet der AR-IoT-Mischanwendungen bestimmt werden. Der Prototyp soll anschließend unter den dabei gesammelten Erkenntnissen entwickelt werden.

1.2 Aufbau der Arbeit

Diese Arbeit ist in sechs Kapitel gegliedert:

Kapitel Kapitel 2 – Hintergrund fördert das grundlegende Wissen zum besseren Verständnis der übrigen Kapitel. Hier werden die Begriffe *Internet of Things* und *Augmented Reality* geschichtlich eingeordnet, definiert und erklärt. Außerdem werden die Grundlagen zur Spiele-Engine Unity gegeben.

Kapitel Kapitel 3 – Verwandte Arbeiten behandelt Werke, welche für diese Arbeit relevant sind. Hierbei werden solche beschrieben, die das Zusammenspiel aus AR und IoT behandeln. Da dieses Gebiet in seiner jetzigen Form relativ neu ist, wird dabei ein breiter Überblick über die verschiedensten Einsatzgebiete dieser Technologien gegeben: Von der Landwirtschaft über Industrie 4.0 bis hin zum Smart Home.

Kapitel Kapitel 4 – Smart-Home-Prototyp beschreibt den Aufbau des Smart-Home-Prototyps. Hier wird die Auswahl der Sensoren und Aktuatoren und der restlichen Hardware sowie deren Einrichtung via MBP beschrieben.

Kapitel Kapitel 5 – Augmented-Reality-Prototyp geht auf die Entwicklung des AR-Prototyps ein. Hier wird ein Augenmerk auf die Reproduzierbarkeit der Arbeit gelegt, indem das Entwickler-Setup mit der HoloLens und Unity im Detail erklärt wird. Außerdem wird eine Übersicht über die technische Umsetzung des Prototyps gegeben, unter anderem wie die Kommunikation und Interaktion zwischen der MBP und dem AR-Prototyp funktionieren und wie das Tracking der IoT-Komponenten umgesetzt wurde.

Kapitel Kapitel 6 – Fazit und Ausblick fasst die Arbeit schlussendlich zusammen und gibt einen Ausblick zum Thema.

2 Hintergrund

Im folgenden Kapitel werden die Grundlagen der Bereiche *Internet of Things* (IoT) und *Augmented Reality* (AR) behandelt. Dabei wird jeweils ein kurzer geschichtlicher Überblick gegeben. Anschließend folgen Definitionen und Erklärungen der Begriffe. Schlussendlich wird auf die Unity Spiele-Engine eingegangen.

2.1 Internet of Things

Die folgenden Abschnitte beschäftigen sich mit dem Internet of Things und den unterstützenden Technologien wie Cloud, Edge und Fog Computing. Außerdem werden die wichtigsten Einsatzgebiete wie Smart Home und Industrie 4.0 erklärt. Schließlich wird auf die IoT-Plattform, die in dieser Arbeit verwendet wird, eingegangen. IoT ist heutzutage ein fester Bestandteil von Industrie und Forschung, sowie zunehmend auch bei Heimanwendern vertreten. Dies zeigt unter anderem die Menge an IoT-Geräten, die bereits im Umlauf ist. Laut Gartner [Meu17] waren bereits 2016 über 6 Mrd. IoT-Geräte in Verwendung mit einer Prognose von 20 Mrd. Geräten bis Ende 2020. Die von IoT generierten Datenmengen wurden auf 22 Zettabytes in 2016, 52 Zettabytes in 2019 und 85 Zettabytes bis 2021 geschätzt [YLH+20].

2.1.1 IoT Übersicht

Der von Mark Weiser 1991 [Wei91] veröffentlichte Ansatz über *Ubiquitous Computing* gilt als der Grundstein des Internet of Things [IGF+17]. Die Vision hinter Ubiquitous Computing war, zu jeder Zeit und an jedem Ort, von Computern umgeben zu sein, diese jedoch nicht zu bemerken. Der Begriff „Internet of Things“ wurde angeblich 1999 von Kevin Ashton als Titel einer Präsentation zum Thema des Einsatzes von Radio-Frequency Identification (RFID)[1] zur Verbesserung der Lieferketten bei Proctor & Gamble verwendet [Ash+09]. Bill Joy war die erste Person, die die Idee von IoT formulierte [IGF+17]. Dies geschah auch 1999 als Teil einer Rede für das Weltwirtschaftsforum in Davos [Pon]. Hier beschrieb er ein sogenanntes „Device to Device Web“, was bereits in Richtung des modernen Verständnisses von IoT ging. Seitdem wurde die Bedeutung der *Things* des IoT stetig erweitert, auch außerhalb des Lieferkettenmanagements. Eine einheitliche Definition scheint jedoch noch nicht gefunden zu sein [DTD19; IGF+17]. Diese Arbeit definiert IoT anhand von Giusto et al. [GIMA10] wie folgt:

Das Internet of Things besteht aus „Things“, die überall um uns verteilt sind. Diese können Sensoren, Aktuatoren oder andere Geräte wie Handys sein. Jedes dieser Things hat eine eindeutige Adresse anhand derer die Things untereinander kommunizieren können.

[1] RFID: Funktechnologie zur Identifizierung von Objekten auf kurze Distanz

2.1.2 IoT Protokolle

In diesem Abschnitt werden beliebte Protokolle zur Kommunikation in IoT-Umgebungen behandelt.

Die folgende Erklärung basiert, wo nicht anders angegeben, auf Withanage et al. [WAYO14]. 2001 wurde mit Z-Wave ein proprietärer Standard zur kabellosen Kommunikation zwischen Smart-Home-Geräten auf den Markt gebracht. Dabei wird je nach Region auf 868.42 MHz (Europa) oder 908.42 MHz (USA) kommuniziert. Die maximale Reichweite von Z-Wave liegt bei 30 m (bzw. 120 m mit bis zu vier Hops). Es können 232 Geräte in einem Z-Wave Netzwerk verbunden werden [UL17]. Für die Kommunikation mit dem Internet benötigt ein Z-Wave-Netzwerk einen Hub, der mit beiden Technologien kommunizieren kann [UL17]. Die maximale Datenrate liegt bei 40 Kbps.

Dieser Abschnitt basiert, wo nicht anders angegeben, auf Ergen et al. [Erg04]. ZigBee wurde 2004 veröffentlicht, um kostengünstige und energiesparende Kommunikation zwischen Smart-Home- bzw. IoT-Geräten, die wenig Bandbreite benötigen, zu ermöglichen. Der Standard wird dabei auf dem Physical und Data Link Layer von der IEEE (802.15.4) und auf dem Network und Application Layer von der ZigBee Alliance (Mittlerweile Connectivity Standards Alliance) spezifiziert. ZigBee unterstützt Datenraten von bis zu 250 Kbps bei einer Frequenz von 2.4 GHz. Dabei können bis zu 6000 Geräte ein Netzwerk bilden [DV19]. ZigBee-Geräte genießen außerdem den Vorteil, in ca. 15ms aus dem ausgeschalteten Zustand aufwachen zu können, was es ihnen ermöglicht, die Sender so lange auszulassen bis etwas gesendet werden muss oder bis ein Empfangen erwartet wird. So erreichen batteriebetriebene ZigBee-Geräte theoretisch eine Laufzeit von mehreren Monaten bis zu Jahren.

Folgender Abschnitt basiert hauptsächlich auf Khorov et al. [KLKG15]. Der IEEE 802.11 Wi-Fi-Standard existiert seit 1997. Anfänglich mit ein bis zwei Mbps auf einem 2.4 GHz Netz, nun mit bis zu 10 Gbps auf 5 GHz via MIMO[2][KLA20]. Spätere Zusätze zum 802.11-Standard wie 802.11ah (Wi-Fi HaLow) zielen speziell auf die Kommunikation in IoT-Umgebungen ab. Im Gegensatz zu den restlichen 802.11 Wi-Fi-Standards setzt 802.11ah auf eine niedrige Frequenz (unter 1 GHz), um weniger Energie zu benötigen und höhere Reichweiten umzusetzen. 802.11ah ermöglicht außerdem die Kommunikation von bis zu 6000 Geräten pro Access Point. Dabei können trotzdem bis zu 7.8 Mbps erreicht werden [CPTT18]. Gleichzeitig können auch maximal 1 km Reichweite auf dem freiem Feld erreicht werden.

Ein weiterer für IoT beliebter Standard ist Bluetooth Low Energy (BLE) der Bluetooth Special Interest Group (SIG). Bluetooth Low Energy wurde 2010 mit Bluetooth Version 4.0 eingeführt [Tec]. In Bluetooth Version 5 wurden einige Verbesserungen an BLE vorgenommen. Die Datenrate liegt so mittlerweile bei 2 Mbps und es können beliebig viele Geräte verbunden werden. BLE sendet wie klassisches Bluetooth auf 2.4 GHz und erreicht somit eine Reichweite von bis zu 200 m. Trotzdem ist der Energieverbrauch dank Optimierungen in Bluetooth 5 deutlich niedriger als klassisches Bluetooth [CPTT18].

[2] Multiple Input Multiple Output: Mehrere Antennen verwenden, um parallel mehr Daten senden zu können

2.1.3 Cloud Computing

Wissenschaftler wie J. C. R. Licklider forschten bereits in den 1960er Jahren an Technologien, die später in dem resultierten, was heute als Internet bekannt ist und so den Grundstein für Cloud Computing legten [Kau09]. 1999 trat Salesforce als einer der ersten Anbieter von cloudbasierter Software, spezifisch Kundenbeziehungsmanagement, auf den Markt [Sal]. Spätestens durch den Start von Amazon Elastic Compute Cloud (EC2) in 2006 [Ama06] existiert Cloud Computing in der Form wie es bis heute bekannt ist. Das US-Amerikanische *National Institute of Standards and Technology* (NIST) definierte Cloud Computing 2011 als „*[...] ein Modell zur Ermöglichung eines allgegenwärtigen, bequemen On-Demand-Netzwerkzugriffs auf einen gemeinsamen Pool konfigurierbarer Computerressourcen (z. B. Netzwerke, Server, Speicher, Anwendungen und Dienste), die mit minimalem Verwaltungsaufwand oder Interaktion mit dem Dienstleister schnell bereitgestellt und freigegeben werden können.*“ [MG+11].

Hierzu existieren laut NIST [MG+11] drei unterschiedliche Service-Modelle, die im Folgenden erklärt werden:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

IaaS bietet dem Kunden Zugriff auf Server-Infrastruktur. Dabei werden der Server, das Netzwerk und der Speicher vom Anbieter zur Verfügung gestellt. Alles Übrige, wie das Betriebssystem, Middleware, Anwendungen und Daten werden weiterhin vom Kunden verwaltet. Ein Beispiel für IaaS wäre das Mieten eines dedizierten Servers.

PaaS stellt zusätzlich zur Infrastruktur auch das Betriebssystem und die Middleware zur Verfügung. Der Kunde muss somit nur seine Anwendung installieren und mit Daten füllen. Ein Beispiel hierfür wäre eine Amazon EC2-Instanz mit vorinstalliertem Linux, Java und Tomcat Webserver.

Bei SaaS stellt der Anbieter dem Kunden lediglich Zugriff auf eine Anwendung zur Verfügung. Der Kunde hat so nur seine Daten zu verwalten, wie beispielsweise bei den cloudbasierten Angeboten von Salesforce.

2.1.4 Edge und Fog Computing

1999 entwickelten Dilley et al. [DMP+02] vom Akamai Technologies das Akamai Content Delivery Network (CDN). Bei einem CDN werden Inhalte näher an den Standort des Nutzers verlagert, um die Latenzen für den Zugriff auf diese zu reduzieren [Sat17]. Dieselbe Strategie wird bei Edge bzw. Fog Computing verfolgt, weswegen das CDN als Grundstein für beide Technologien erachtet werden kann [Sat17]. Edge Computing verlagert hauptsächlich Compute-Ressourcen aus dem Rechenzentrum, der Umgebung von klassischem Cloud Computing, näher zu den Endgeräten. Der Begriff *Fog-Computing* wurde unter anderem maßgeblich von Bonomi et al. [BMZA12] und Cisco [Com15] geprägt. Fog Computing ist dabei zwar sehr ähnlich zu Edge Computing, indem es auch Ressourcen näher zu den Endgeräten bringt, jedoch nicht nur Compute, sondern auch Speicher, Netzwerke etc. [YFN+19].

IoT-Geräte sind oft nicht leistungsstark genug, um komplexe Berechnungen lokal durchzuführen oder benötigen / erzeugen große Mengen an Daten. Je nach Szenario kann typisches Cloud Computing hierbei nicht die Latenzanforderungen der Geräte erfüllen [SCM15]. In diesen Fällen kann Edge oder Fog Computing eingesetzt werden. Speziell für AR-Anwendungen ist die geringe Latenz sehr wichtig. Bereits Latenzen von über 20ms, die meist durch Netzwerklatenzen entstehen, können zu Bewegungskrankheit und so zu einem unbrauchbaren Nutzererlebnis führen [VLA+20].

2.1.5 Smart Home

Das Smart Home ist aktuell eines der wichtigsten Anwendungsgebiete von IoT [AZZ+17]. Zuerst verwendet wurde der Begriff *Smart Home* bzw. *Smart House* 1984 von der American Association of House Builders [Har03]. Die Idee, die Steuerung von Geräten im Haushalt zu automatisieren ist jedoch noch älter. So wurde das X10-Protokoll, welches Geräte über das im Haus vorhandene Stromnetz steuert, bereits 1975 veröffentlicht [WAYO14]. Anfang der 1980er wurde dazu noch ein Computer-Interface für das X10-System auf den Markt gebracht, was die Steuerung aus z. B. MS-DOS ermöglichte [Dri]. X10 ist selbst heute noch in Verwendung und lässt sich mittlerweile auch über Funk steuern [IAAH16]. Doch seitdem wurden, wie in der Übersicht über die Protokolle bereits erklärt, auch andere Technologien zur Kommunikation zwischen smarten Geräten entwickelt.

Frances K. Aldrich [Ald03] definiert ein Smart Home als die Verwendung von IT in einem Haus, um auf die Bedürfnisse der Bewohner einzugehen und so deren Komfort zu erhöhen.

Folgendes basiert auf López et al. [LQG17]. Gesteuert werden Smart-Home-Geräte oft von Smartphone-Apps [AZZ+17] oder intelligenten persönlichen Assistenten. Diese Assistenten verwenden natürliche Sprache als Eingabe zur Steuerung. Die erfolgreichsten Beispiele sind dabei *Apple Siri*, *Google Assistant*, *Microsoft Cortana*, *Amazon Alexa*. Siri war hierbei 2011 der erste auf dem Markt erhältliche Assistent dieser Art. 2012 folgte Google mit der Veröffentlichung von *Google Now* den Vorgänger des 2016 veröffentlichten Google Assistenten. 2014 folgten Cortana und Alexa.

2.1.6 Industrie 4.0

Ein weiterer wichtiger Anwendungsbereich von IoT ist das *Industrial IoT* (IIoT) im Industrie-4.0-Bereich. Folgendes basiert auf Yang Lu [Lu17].

Die erste industrielle Revolution war der von Dampfmaschinen und Wasserkraft angetriebene Wechsel von Handarbeit zu maschineller Produktion im 18. Jahrhundert. Der Umschwung zur Massenarbeit dank elektrischer Energie im 20. Jahrhundert gilt als zweite Industrielle Revolution. Die Automatisierung und Digitalisierung seit den 1970er Jahren gilt schließlich als dritte Industrielle Revolution.

Industrie 4.0 wurde 2011 durch die Deutsche Bundesregierung populär als Teil der deutschen Wirtschaftsstrategie. Industrie 4.0 bzw. die vierte industrielle Revolution beschreibt dabei die Vernetzung und computergestützte Steuerung in der Industrie. Dies wird durch weitere Automatisierung erreicht, um die Effizienz und Produktivität zu steigern.

IIoT ist die Grundlage von Industrie 4.0. Hier werden Sensoren verwendet, um den Zustand der Produktion zu bestimmen, die Prozesse anzupassen oder generelle Analysen durchzuführen. Dies kann z. B. verwendet werden, um frühzeitig Wartungen anhand der gesammelten Daten durchzuführen. Es können so auch Bottlenecks in der Produktion aufgedeckt werden. Das alles erhöht die Effizienz der Industrie [HSK+19].

2.1.7 Multi-purpose Binding and Provisioning Platform

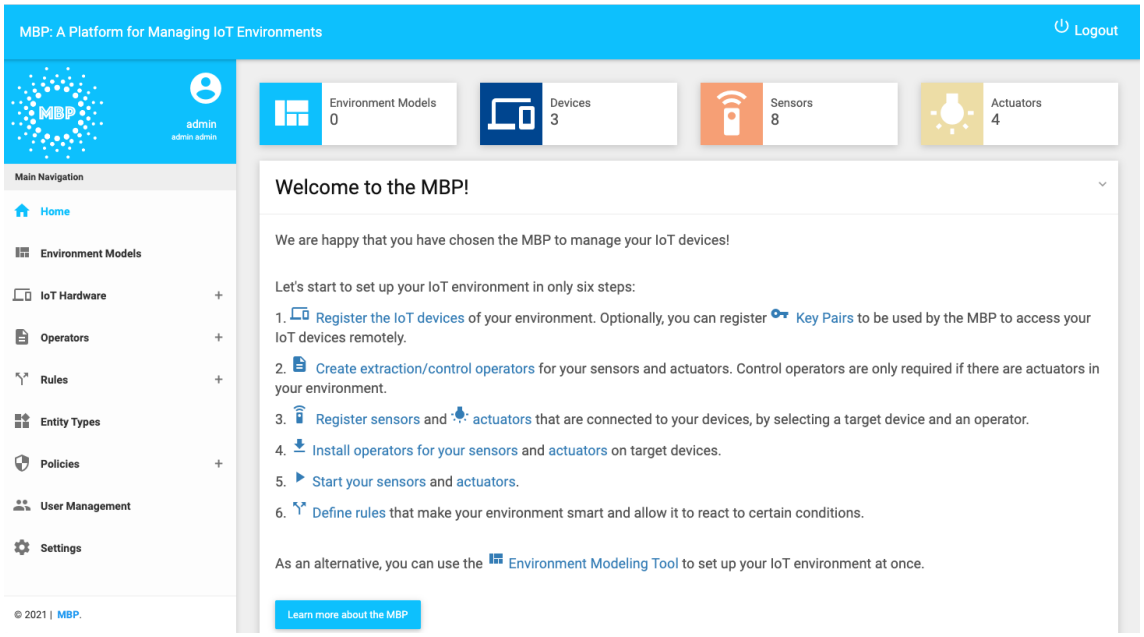


Abbildung 2.1: Screenshot der MBP-Hauptseite

Die Multi-purpose Binding and Provisioning Platform (MBP), entwickelt von Franco da Silva et al. [SHS+20], ist eine Open-Source-Plattform zur Verwaltung von IoT-Geräten. Im Gegensatz zu ähnlichen Projekten versucht sie, den Anteil manueller Operationen beim Einrichten und Verbinden von IoT-Geräten und beim Abrufen der Sensordaten zu minimieren. Dazu bietet die MBP die Möglichkeit IoT-Umgebungen und damit Verbindungen zwischen Geräten zu modellieren. So wird das Deployment vereinfacht und intuitiver. Es existiert außerdem eine Android-App die den Nutzer beim Einrichten der IoT-Umgebung unterstützt. Zusätzlich wird das Überwachen der Geräte angeboten. Z. B. indem die CPU-Auslastung und der Online-Status eines Raspberry Pis auf dem Dashboard der MBP angezeigt werden kann.

Die MBP bietet zusätzlich noch einige fertige sogenannte Operator-Skripte für verschiedene Geräte an. Diese werden verwendet, um entweder Sensordaten auszulesen oder Aktuatoren zu steuern. Die Kommunikation zwischen IoT-Gerät und MBP funktioniert in beiden Fällen via Message Queuing Telemetry Transport (MQTT)[3].

[3] MQTT: Publish/Subscribe Protokoll zum Transport von Nachrichten zwischen Geräten

Das Anschließen eines Helligkeitssensors funktioniert beispielsweise wie folgt:

Der Sensor wird mit demselben Netzwerk verbunden, in dem die MBP läuft, z. B. in dem er an einen Raspberry Pi angeschlossen wird, welcher dem Netzwerk via WLAN beitreten kann. Anschließend wird hierzu ein Gerät in der MBP angelegt, falls noch nicht geschehen. Daraufhin wird der Operator, für den Helligkeitssensors ausgewählt, auf den Pi deployt und gestartet. Im Frontend dieses Sensors können nun sowohl aktuelle, als auch historische Werte, sowie Minima und Maxima abgelesen und der Sensor verwaltet werden. Diese Sensoren können auch in Regeln verwendet werden, um Abläufe zu automatisieren. Falls für einen Sensor oder Aktuator noch kein Operator existiert, kann dieser in einer beliebigen Programmiersprache erstellt werden. Dabei muss lediglich der Struktur eines Operators gefolgt werden. Es werden Management-Skripte benötigt, wie z. B. ein „Install“-Skript, welches dazu verwendet wird, die Abhängigkeiten oder Bibliotheken des jeweiligen Operators zu installieren. Nähere Informationen zu den Skripten folgen in *Abschnitt 4.5*. So können beliebige, auch selbst entwickelte Sensoren und Aktuatoren mit der MBP verbunden werden.

Die Installation der MBP wird via Shell-Skript unter Linux oder mit Docker durchgeführt. Zugriff erfolgt über einen Webbrowser, da die MBP ein Angular JS basiertes Web Frontend verwendet. Der Code befindet sich auf GitHub[4].

2.2 Augmented Reality

Folgende Abschnitte behandeln die Hintergründe zu Augmented Reality. Beginnend bei einer kurzen geschichtlichen Einordnung, Erklärung der verwendeten Hardware und der populärsten Tracking-Methoden.

Augmented Reality existiert bereits seit einiger Zeit. So forschten Drascic und Milgram [DM91] beispielsweise bereits 1991 an der Möglichkeit Overlays über stereoskopisches Video zu legen, um Teleoperator-Systeme zu verbessern. 1993 entwickelten Milgram et al. mit ARGOS [MZDG93] eine Möglichkeit, das Arbeiten mit Telepräsenzrobotern durch AR zu vereinfachen. Dabei wurde dem Bediener ein stereoskopisches Bild via Videobrille gezeigt, welches unter anderem über einen steuerbaren dreidimensionalen Zeiger und ein Maßband zur Bestimmung von Abständen verfügte.

1994 stellten Milgram und Kishino das *Virtuality Continuum* [MK94] auch bekannt als *Reality-Virtuality Continuum* [MTUK95] vor. Dabei handelt es sich um eine Skala die in verschiedenen Abstufungen zwischen der realen Umgebung und der virtuellen Umgebung unterscheidet. Augmented Reality liegt so näher an der realen Umgebung als an der Virtuellen. AR wird hierbei definiert als die Fälle, *„[...] in denen die Darstellung einer ansonsten realen Umgebung durch virtuelle (computergrafische) Objekte erweitert wird.“* [MK94]. Das Gegenstück dazu, sprich eine Verwendung von realen Objekten in virtuellen Umgebungen, wird als *Augmented Virtuality (AV)* definiert. Außerdem wurde in dieser Arbeit der Begriff *Mixed Reality (MR)* das erste Mal verwendet [MK94]. Wie in Abbildung 2.2 zu sehen ist, definiert sich MR als all die Gebiete, die versuchen, die reale und virtuelle Umgebung miteinander zu kombinieren.

1997 definierte Ronald T. Auzuma [Azu97] Augmented Reality, auf dem Stand der damaligen Wissenschaft, über die folgenden drei wichtigsten Anforderungen:

[4]MBP GitHub: https://github.com/IPVS-AS/MBP

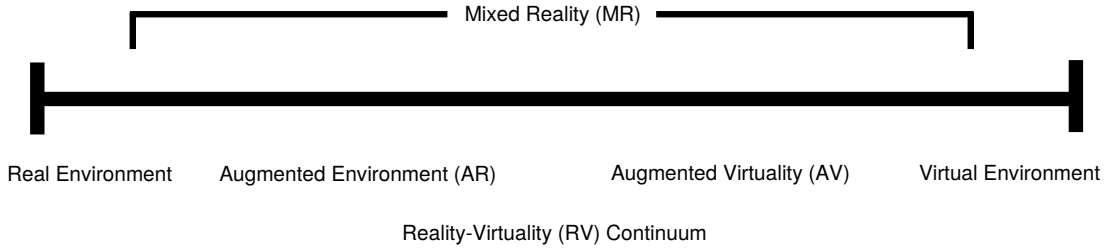


Abbildung 2.2: Reality-Virtuality Continuum nach [MTUK95]

- Kombination aus realen und virtuellen Inhalten
- In Echtzeit interaktiv
- Im dreidimensionalen Raum registriert

Auzuma erwähnte außerdem, dass sich AR für ihn nicht nur auf das Sehen, sondern auch auf alle Sinne beziehen kann. So existiert mittlerweile auch *Auditory Augmented Reality (AAR)*. Beispielsweise präsentieren Russell et al. [RDP16] mit *HearThere* einen Prototyp, um virtuelle räumliche Töne mit denen der realen Welt nahtlos zu verschmelzen.

Im Zuge dieser Arbeit wird jedoch hauptsächlich auf AR für visuelle Augmentierung eingegangen. So wird AR, angelehnt an Milgram et al. [MK94], definiert als das Augmentieren der realen Umgebung durch virtuelle Objekte.

Ein Beispiel hierfür ist *SecondSurface* von Kasahara et al. [KHLI12]. SecondSurface ist eine AR Anwendung für mobile Geräte, bei der Nutzer in der realen Welt verankerte Zeichnungen und Bilder mit anderen Nutzern teilen können. Die Inhalte dazu werden nach dem Erstellen auf einem Server in der Cloud gespeichert.

2.2.1 Hardware

Augmented Reality Hardware kommt in vielen verschiedenen Formen für die verschiedensten Anwendungsgebiete vor. Hauptsächlich werden als Anzeige der AR-Inhalte die folgenden Technologien verwendet [ZDB08]:

- Head Mounted Displays (HMD)
  - Optical See-Through Head Mounted Displays (OST-HMD)
  - Video See-Through Head Mounted Displays (VST-HMD)
- Projection-Based Displays (PD)
- Handheld Displays (HHD)

Head Mounted Displays

Ein Head Mounted Display ist eine Apparatur, die meist einer Brille ähnelt und dem Nutzer auf einem oder beiden Augen Bilder anzeigen kann. Bereits 1968 veröffentlichte Sutherland seine Forschung zu einem stereoskopischen Head Mounted Display [Sut68]. Dieses war damals für Virtual Reality (VR) entwickelt, somit weder OST noch VST, und unter Verwendung von zwei Miniaturröhrenbildschirme. Dieses HMD verfügte jedoch bereits über Kopf-Tracking, anfänglich noch mechanisch, später per Ultraschall. Caudell und Mizell beschrieben 1992 [CM92] wie eine AR-Anwendung mit OST-HMD verwendet werden könnte, um den Bau von Flugzeugen kostengünstiger und effizienter zu gestalten. Ein VST-HMD ist an sich lediglich ein undurchsichtiges HMD das ein Videosignal der realen Umgebung anzeigt. Hierbei liegen die Vorteile in besserer Umgebungsverdeckung, auch Occlusion genannt, sowie einer höheren Konsistenz zwischen realen und virtuellen Inhalten [ZDB08]. Das OST-HMD beinhaltet hingegen das Durchsehen durch eine holographische Anzeige z. B. durch ein halbverspiegeltes Medium. Die Vorteile liegen hier in der verbesserten Darstellung der realen Welt und geringerer Latenz bei Bewegungen [ZDB08]. Andererseits ist Occlusion ein schwer zu lösendes Problem, da die Anzeige transparent ist und reale Objekte so nicht verdeckt werden können.

Projection-Based Displays

Als Projection-Based Displays werden alle durch Projektion erzeugte Bildflächen genannt. 1991 entwickelten Cruz-Neira et al. CAVE [CSD93], ein projektorbasiertes VR-System. Hier projizieren vier 3D-Projektoren auf drei Seitenwände und den Boden eines Raumes. Die Nutzer werden dabei via elektromagnetischer Sensoren getrackt und tragen 3D-Brillen für den stereoskopischen Effekt. Auch für Augmented Reality können projektorbasierte Systeme eingesetzt werden. So verwendeten Doshi et al. ein projektorbasiertes System zur Hervorhebung von Punktschweißstellen auf Automobilteilen während der Montage [DSTB17]. Der Vorteil liegt hier darin, dass der Nutzer kein Gerät in der Hand halten oder auf dem Kopf tragen muss und potentiell beliebig viele Nutzer dasselbe System verwenden können [ZDB08].

Handheld Displays

Handheld Displays bezeichnet alles was ein Nutzer in der Hand bzw. am Arm tragen kann. Von einer Smartwatch über das Smartphone zum Tablet. 1997 präsentierten Feiner et al. mit *Touring Machine* [FMHW97] ein mobiles AR-System für Zusatzinformationen beim Weg über den Campus der Columbia University. Hier wurde unter anderem ein Tablet-Computer mit GPS-Hardware verwendet, um die passenden Informationen darzustellen. In Zeiten von Smartphones bieten sich HHDs dank ihrer Ubiquität, Portabilität und einfachen Bedienweise für AR-Anwendungen an [AA17]. Smartphone-AR-Plattformen wie ARKit sorgten zusätzlich dafür, dass Mobile AR massentauglich wurde [WCPC19]. Diese Vorteile zeigen sich z. B. durch den Erfolg des Handheld AR-Spiels Pokémon GO, mit anfangs 5 Millionen aktiven Nutzern pro Tag und 650 Millionen Downloads in 2016 [GSG18].

2.2.2 Tracking

Eine wichtige Eigenschaft von Augmented Reality ist, dass die Augmentierungen in der realen Umgebung verankert bzw. registriert sind [Azu97]. Dies geschieht via Tracking der realen Umgebung. Je nach Anwendungszweck existieren hier viele verschiedene Möglichkeiten des Trackings. Laut Zhou et al. [ZDB08] lässt sich die Forschung im Bereich Tracking hauptsächlich in drei Kategorien aufteilen:

- Visionsbasiertes Tracking
- Sensorbasiertes Tracking
- Hybrides Tracking

Visionsbasiertes Tracking

Ein sehr aktives Gebiet des Trackings scheint bereits seit vielen Jahren das visionsbasierte Tracking zu sein [KBB+18; ZDB08]. Hierbei wird in *Marker Tracking* und *markerloses Tracking* unterschieden.

Für Marker Tracking werden künstliche Marker in der realen Umgebung angebracht, welche so entworfen sind, dass ein Kamerasystem diese einfach erkennen kann [ZDB08].

Die folgende Erklärung von Marker Tracking basiert auf der Arbeit von Garrido-Jurado et al. [GMMM14]. Dabei bezieht sie sich auf quadratische Marker mit schwarzem Rand, wie in Abbildung 2.3 zu sehen (auch Fiducial Marker genannt), die z. B. auf Papier gedruckt und an Objekte angeklebt werden können.

Noch vor dem Tracking der Marker müssen diese generiert werden. Hierzu wird eine Zufallsfunktion verwendet, die für einen *n* Zellen großen Marker *n-2* viele unterschiedliche binäre Worte generiert. Die Anzahl liegt bei *n-2*, da jeweils eine Zelle für den schwarzen Rand verwendet wird. Aus diesem Prozess zeigt sich auch, dass somit die Anzahl an Zellen des Markers dafür verantwortlich ist, wie viele Marker insgesamt generiert werden können. Je höher das *n*, desto mehr unterschiedliche Marker können generiert werden. Gleichzeitig werden die einzelnen Zellen im Marker bei selber physischer Größe kleiner und schwerer zu tracken (siehe Abbildung 2.3). Die generierten Marker werden in einem sogenannten Dictionary gespeichert.

Abbildung 2.3: Unterschiedlich große Marker; von links nach rechts n=5, n=6, n=8 [GMMM14]

Um die Marker im Kamerabild zu detektieren, müssen alle potentiellen Kandidaten für Marker im Bild erkannt werden. Dafür wird zuerst eine Kantendetektion ausgeführt. Eine Kante besteht hier aus aneinanderliegenden Punkten, an denen sich der Helligkeitswert stark ändert. Zunächst werden die Konturen, sprich zusammenhängende Kanten, die ein Polygon bilden, extrahiert. Da die hier behandelten Marker die Form eines Quadrats bilden, wird anschließend überprüft, welche der Konturen ein vierseitiges Polygon bilden. Die übrigen Konturen werden verworfen. Bei den verbliebenen wird versucht, ausschließlich die Außenkonturen zu behalten und potentiell im inneren des Markers detektierte Konturen zu verwerfen.

Anschließend folgt die Extraktion der Marker-Daten. Bei allen noch verbliebenen Marker-Kandidaten wird zuerst die perspektivische Verzerrung korrigiert. Daraufhin werden die Bereiche, die Kandidaten enthalten, mit einem Gitter aus n Zellen überzogen und binarisiert. Dazu wird ein Schwellenwert ausgerechnet, anhand dessen für jede Zelle der Marker-Kandidaten entschieden wird, ob diese weiß oder schwarz ist bzw. den Wert 1 oder 0 darstellt. Auf dieser Binärmatrix wird überprüft, ob alle am Rand befindlichen Zellen den Wert 0 enthalten. So wird festgestellt, ob der eine Zelle breite, schwarze Rand um den Marker vorhanden ist. Anschließend werden aus jeder der vier möglichen Rotationen Binärworte extrahiert. Befinden sich diese im Dictionary wird der Marker-Kandidat als valide erachtet. Ein nicht valider Marker-Kandidat kann an dieser Stelle noch durch verschiedene Algorithmen zur Fehlerkorrektur gegeben werden, um einen validen aber fehlerhaft extrahierten oder teils verdeckten Marker trotzdem zu erkennen.

Anschließend kann die Lagebestimmung, auch *Pose Estimation* genannt, durchgeführt werden. Auch hier existieren verschiedenste Herangehensweisen, weshalb sich im Folgenden an Maidi et al. [MDAM10] bzw. Didier [Did05] orientiert wird. Generell kann die Lagebestimmung als dreidimensionale Matrix angesehen werden, die die Rotation und Translation der Objektkoordinaten des Markers in das Koordinatensystem der Kamera oder umgekehrt darstellen. Üblicherweise muss hierzu initial die Kamera kalibriert werden. Kalibrieren bedeutet hier, dass die intrisischen bzw. internen Kameraparameter wie z. B. die Brennweite oder mögliche Verzerrungen geschätzt und gespeichert werden [BPG10]. Anhand der dabei entstehenden sogenannten Kamera intrinsischen Matrix, den Koordinaten der vier Eckpunkte des Markers und der tatsächlichen Größe der Marker kann die Rotation und Translation des Markers berechnet werden. Die Translations- und Rotationsmatrix kann anschließend verwendet werden, um AR-Elemente an die Position des Markers zu zeichnen.

Es existieren auch andere Arten von Markern, die nicht ausgedruckt werden. So entwickelten Chaves-Diéguez et al. [CPG+15] ein System zum Tracking von Markern aus sichtbarem Licht. Dazu wurden acht LEDs in einem Quadrat angeordnet, mit einer Kamera gefilmt und via der Graphikbibliothek OpenCV[5] analysiert. Eine der LEDs diente als Referenz, drei für den Takt des Datensignals und vier zur Übertragung des Datensignals.

Generell muss sich für das Marker Tracking auf dem zu trackenden Objekt ein Marker befinden. Dies kann zum einen für den Nutzer unangenehm sein und zum anderen, bei größeren Anwendungen, auch Probleme mit der maximalen Anzahl an eindeutig identifizierbaren Markern bereiten [MAC18].

So existiert eine Alternative zum Marker Tracking, das *markerlose Tracking*. Der folgende Abschnitt zu markerlosem Tracking basiert auf der Arbeit von Barandiaran et al. [BPG10].

[5] OpenCV: https://opencv.org

Auch hier, wie beim Marker Tracking, wird eine Transformation und Rotation zwischen den Kamerakoordinaten und Objektkoordinaten gesucht. Hierzu wird zuerst wieder die Kamera intrinsische Matrix benötigt, um die Eigenheiten der Kamera zu berücksichtigen und Tracking-Fehler aufgrund dessen zu vermeiden. Anschließend wird versucht, die Transformation, von Ebenen der realen Umgebung (wie z. B. dem Boden oder Wänden) zu deren Projektionen in Kamerakoordinaten als 3x3 Matrix zu bestimmen. Diese Matrix wird auch *Homographie* genannt. Hierzu wird versucht übereinstimmende Punkte in beiden Ebenen zu finden. Diese werden auch als *Natural Features* bezeichnet. Ähnlich zum Marker Tracking werden hier mindestens vier übereinstimmende Punkte benötigt, für bessere Resultate, speziell in Bezug auf die Genauigkeit, werden meist mehr als vier Punkte verwendet. Es gibt hauptsächlich zwei Herangehensweisen, um die Homographie zu bestimmen. *Rekursives Tracking* oder *Tracking By Detection*. Rekursives Tracking geht von einem Initialzustand aus, bei dem die vier Punkte manuell ausgewählt werden müssen. Anschließend wird rekursiv unter Verwendung der Information des jeweils vorherigen Durchgangs die Homographie aktualisiert. Durch die Rekursivität können sich Fehler im Tracking anhäufen, was potentiell bis zum Ausfall und einer Reinitialisierung führen kann. Tracking by Detection folgt den selben Grundprinzipien wie rekursives Tracking. Es müssen jedoch vor dem Tracking Informationen über das zu trackende Objekt existieren. So kann beispielsweise ein 3D-Modell verwendet werden. Die Berechnung, wo die zu trackenden Punkte im Bild liegen, benötigt hierbei keine Information des vorherigen Durchgangs. Das Tracking wird für jeden Frame neu berechnet, sodass sich keine Tracking-Fehler ansammeln können. Dadurch können sogar Probleme wie teilweise Verdeckung gelöst werden. Die Objekte beim markerlosen Tracking können beispielsweise Schilder sein, wie bei Koch et al. [KNKA14]. Schilder eigenen sich laut Koch et al. als Marker, da sie gut sichtbar hängen, nicht zu klein sind und hervorstechende Farben haben.

Sensorbasiertes Tracking

Ein weiteres Teilgebiet des Trackings ist das sensorbasierte Tracking. Dabei werden verschiedene Sensoren verwendet, um die Position der Kamera bzw. des Endgeräts zu verfolgen [RU13]. Im Gegensatz zum visionsbasierten Tracking gibt es hier sehr viele unterschiedliche Möglichkeiten des Trackings, je nach beteiligten Sensoren. So verwendeten Xiao et al. [XLZH17] beispielsweise einen elektromagnetischen Sensor an einem Smartphone, zur Detektion von steuerbaren Geräten im Raum. Auch akustische Sensoren können für das Tracking verwendet werden. Wie bereits in diesem Kapitel gezeigt, verwendete Sutherland für sein HMD Ultraschall zum Tracking der Kopfposition des Nutzers [Sut68]. Selbst optische Methoden zum Tracking finden Verwendung. So verwendete die projektorbasierte AR-Anwendung *PICOntrol* [SMC12] von Schmidt et al. das Licht des Projektors, um via Photosensoren auf den zu steuernden Geräten, diese zu erkennen. Jede Art von Sensor bringt auch Probleme mit sich, Magnetfelder werden beispielsweise durch die Anwesenheit elektrischer Geräte gestört. Deshalb existieren auch eine Reihe von hybriden Herangehensweisen, die mehrere Sensoren kombinieren, um bessere Ergebnisse zu erhalten [RU13]. Inertiale Messeinheiten (IMUs)[6] sind deshalb gut für Tracking einsetzbar. So demonstrierten Oskiper et al. [OSK12] einen Ansatz zur Verwendung einer IMU für Indoor und Outdoor AR, unter

[6] Inertiale Messeinheit: Kombination aus meist Accelerometern, Gyroskopen und Magnetometern

anderem mit und ohne GPS. Jeon et al. [JCKL19] verwendeten niederfrequenten Ultraschall für AR Tracking von Personen. Zusätzlich wurde der Ultraschall mit einem PIR-Sensor[7] kombiniert, um fehlerhaftes Tracking durch reflektierenden Schall zu verhindern.

Hybrides Tracking

Nicht nur verschiedene Sensoren lassen sich kombinieren, sondern auch visionsbasierte Methoden mit sensorbasierten.

So entwickelten Hua et al. [HLRJ21] mit *ArcIoT* ein System zur Steuerung von smarten Geräten und IoT mit einem Smartphone. Dabei verwendeten sie visionsbasiertes Tracking via der Tracking Engine Vuforia[8] und sensorbasiertes via BLE[9] Beacons[10]. Zusätzlich wird noch die IMU des Smartphones verwendet, um eine Karte der Umgebung zu erstellen.

Xia et al. [XWW+21] entwickelten mit *INSIGHT* eine weitere Mobile-AR-Anwendung zur Interaktion mit IoT-Geräten. Dabei wird visionsbasiertes Tracking via Vuforia zum Tracken der IoT-Geräte und die IMU des Handys zum Abgleich mit einer Datenbank bzw. Karte verwendet. Wenn das Gerät dort nicht gefunden wird, wird es manuell angelegt. Ansonsten wird die dort gespeicherte IP-Adresse verwendet, um mit dem Gerät zu kommunizieren. Der Vorteil hierbei ist, dass diese Herangehensweise auch bei vielen gleich aussehenden IoT-Geräten, auch ohne Marker funktioniert, da Vuforia hier nur verwendet wird, um zu detektieren, ob ein Gerät gefunden wurde.

2.3 Unity Spiele-Engine

Folgendes basiert hauptsächlich auf dem Buch „Developing 2D Games with Unity“ von Jared Halpern [Hal19].

Unity ist eine 2005 veröffentlichte Spiele-Engine und gleichnamige Entwicklungsumgebung. Eine Spiele-Engine ist eine Software, die häufig benötigte Funktionalitäten wie Rendering und Spielphysik implementiert und so die Komplexität für den Entwickler reduziert. Unity ist bereits in erfolgreichen Spielen, wie *Temple Run*, *Hearthstone*, *Pokémon GO*, *Cuphead* und vielen weiteren eingesetzt worden. Wie in *Kapitel 3* zu sehen, ist Unity nicht nur bei der Spieleentwicklung, sondern auch zur Erstellung von AR- und VR-Anwendungen beliebt. Spiele und Anwendungen bestehen in Unity aus *Szenen*. Eine Szene ist hierarchisch in einen Szenengraph aus *GameObjects* aufgeteilt. Ein GameObject ist hierbei jedes in der Szene vorkommende Objekt, vom 3D-Modell über Kameras, Lichtquellen bis hin zu Nutzer-Interfaces. Die Hierarchie wird hierbei durch eine Eltern-Kind-Beziehung der GameObjects hergestellt. Ein GameObject kann so wieder mehrere GameObjects enthalten. Unity folgt hierbei einem sogenannten *Entity-Component Design*. Die Entity ist dabei das GameObject und die Component die sogenannten Komponenten die jedes GameObject besitzt.

[7] Pyroelektrischer Sensor: Infrarotbasierter Sensor, der Temperaturänderungen detektiert; wird oft in Bewegungsmeldern verwendet

[8] Vuforia Engine: https://developer.vuforia.com

[9] Bluetooth Low Energy: Energiesparende und reichweitenbeschränkte Version von Bluetooth

[10] BLE Beacon: Teilt anderen Geräten in der Umgebung seine ID mit, was zur Standortbestimmung verwendet werden kann

Eine dieser Komponenten ist zum Beispiel die *Transform*-Komponente, zur Positionierung von GameObjects. C#-Skripte können auch als Komponenten an ein GameObject angehängt werden. Ein GameObject inklusive Komponenten kann außerdem für die mehrmalige Wiederverwendung als sogenanntes *Prefab* gespeichert und instanziiert werden.

Ein Unity-Projekt besteht in der Regel hauptsächlich aus den Ordnern „Project Settings“, „Library“, „Packages“, „Temp“ und „Assets“. Der Project-Settings-Ordner enthält einige der gespeicherten Einstellungen des Projekts, unter anderem auch die Unity-Version, unter der das Projekt angelegt wurde oder verschiedene Build-Einstellungen. Der übrige Teil der Einstellungen wird zusammen mit den verwendeten Plugins im Library-Ordner gespeichert. Importierte Pakete befinden sich im Packages-Ordner. Der Temp-Ordner existiert nur zur Laufzeit von Unity und beinhaltet temporäre Dateien des offenen Projekts.

Der Assets-Ordner ist der Ordner der alle Ressourcen der Anwendung, wie Skripte, Szenen, Prefabs, Materialien etc. enthält. Materialien werden z. B. für die Färbung von GameObjects verwendet. Im „Scripts“-Ordner befindet sich der C#-Code der Anwendung. Die Einrichtung eines Unity-Projekts wird in *Abschnitt 5.3* anhand des für diese Arbeit erstellten Prototyps erklärt.

3 Verwandte Arbeiten

Das folgende Kapitel umfasst alle Arbeiten, die eine ähnliche Thematik zu dieser Arbeit besitzen. So wird meist ein Zusammenspiel von IoT und AR beschrieben, jedoch in vielen verschiedenen Kontexten. Anfänglich werden Arbeiten aus der Landwirtschaft sowie der Medizin und Pflege untersucht, gefolgt von solchen zum Thema Smart City. Die umfangreichen Arbeiten aus den Bereichen Industrie 4.0 und Smart Home bzw. Consumer IoT folgen darauf. Schließlich wird auf die Architektur von AR-IoT-Mischanwendungen eingegangen, gefolgt von weiteren Arbeiten aus noch nicht behandelten Themengebieten. Ein Fazit zieht final Rückschlüsse aus dem Kapitel und wendet sie auf diese Arbeit an.

3.1 Landwirtschaft

Auf dem Gebiet der Landwirtschaft finden sich einige Anwendungen, die sich die Visualisierungsfähigkeiten von AR und die Datenerfassung des IoT zunutze machen können. Ein Auszug dieser wird im Folgenden behandelt.

Pilaiwan Phupattanasilp und Sheau-Ru Tong [PT19] beschrieben ein System namens *AR-IoT*, welches Landwirte bei der Bepflanzung von Gewächshäusern unterstützen soll. Dabei werden die Pflanzen via markerlosem Tracking und mehreren Kameras erkannt. Das Tracking wurde hier unter Verwendung von MATLAB[1] selbst implementiert. Markerloses Tracking wurde gewählt, da sonst an jeder Pflanze ein Marker angebracht werden müsste. Zusätzlich wird zur Kalibrierung in regelmäßigen Abständen zwischen den Pflanzen ein Schachbrettmuster platziert. Gleichzeitig werden an den Pflanzen Sensoren für Feuchtigkeit, Temperatur, Helligkeit und Nährstoffe angebracht. So können diese Parameter überwacht und entsprechend kontrolliert werden. Der Landwirt wählt in der Anwendung eine Pflanze aus, um sich Informationen über diese anzeigen zu lassen. Zu sehen ist das in Abbildung 3.1. Eine Studie zeigte die Vorteile von AR-IoT speziell für unerfahrenere Landwirte.

Ponnusamy et al. [PN21] befassten sich mit einer Vision zur Erhöhung der Effizienz der Agrarwirtschaft. Hierzu soll unter anderem AR und IoT zusammen verwendet werden. Bodensensoren liefern beispielsweise Daten über den Zustand der Pflanzen, Nährstoffe im Boden, Feuchtigkeit. Diese Daten werden von den IoT-Geräten an die Cloud zur Auswertung gesendet. Hier sollen auch Vorteile der Cloud wie die Elastizität verwendet werden, um die Kosten so niedrig wie möglich zu halten. Zur Unterstützung der Bodensensoren sollen ebenfalls Drohnen eingesetzt werden. Diese dienen sowohl als Sensoren, durch ihre Kameras, als auch als Aktuatoren, um z. B. Spritzmittel zu verteilen, wo nötig. Anschließend können Visualisierungen der Daten aus der Cloud in AR durchgeführt werden, um die Ergebnisse dem Landwirt besser näherzubringen. Hierzu soll entweder ein Epson

[1] MATLAB: Programmiersprache und Software zur Lösung mathematischer Probleme

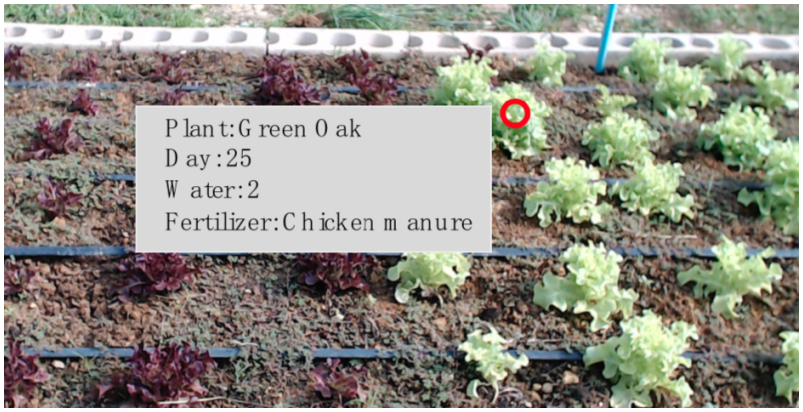


Abbildung 3.1: Beispiel der Zusatzinformationen einer Pflanze [PT19].

Moverio BT-350[2] OST-HMD oder ein selbst entwickeltes HMD, bestehend aus einem Raspberry Pi Zero, Kamera und LCD, verwendet werden. Es wird jedoch auch ein Smartphone-basierter Ansatz diskutiert, da Smartphones durch ihre hohe Verbreitung zugänglicher sind.

Katsaros et al. [KKS17] kombinieren AR und IoT, um eine Anwendung für Landwirte zu entwickeln, die dabei helfen soll, den Ertrag zu maximieren und zu entscheiden, ob bestimmte Pflanzen in ihrer Region wirtschaftlich sind. Dabei werden Parameter wie die Kosten für Dünger oder der Verlust durch Schädlinge anhand von gemessenen Sensordaten und Informationen aus einer Wissensdatenbank eingerechnet. Zur Visualisierung der Daten wird eine Android-App unter Verwendung von *FarmAR* und Vuforia verwendet. FarmAR ist eine von Katsaros et al. [KK17] entwickelte App, um Landwirten Zusatzinformationen zu Pflanzen zu geben, beispielsweise den Namen und häufig auftretende Krankheiten dieser Pflanze. Dabei wird eine Pflanze mit einem Smartphone gescannt und die nötigen Informationen dazu aus der Wissensdatenbank und von den Sensoren geladen und angezeigt.

Nigam et al. [NKD11] stellen eine Android-basierte Handheld-AR-Anwendung für Landwirte vor. Dabei werden verschiedene Insekten an Pflanzen identifiziert. Nach einer Analyse, ob diese schädlich sind, werden die nötigen Maßnahmen im Kamerabild angezeigt. Das Erkennen der Insekten wurde via Qualcomm QCAR SDK (heute Vuforia) umgesetzt.

Huuskonen und Oksanen [HO18] entwickelten einen Prototyp, um geeignete Stellen für Bodenproben auf Feldern aufzufinden. Die Basis dafür sind Drohnenbilder, die automatisch nach dem Umpflügen eines Feldes aufgenommen und analysiert werden. Die Bildverarbeitung ist dabei selbst implementiert und erkennt Unterschiede in der Farbe des Bodens. Anhand dessen werden die Stellen für Bodenproben identifiziert. Zur Navigation an die ausgewählten Stellen soll ein Android-basiertes OST-HMD verwendet werden. Die AR-Anwendung wurde hierfür auf Basis des Wikitude AR SDKs[3] entwickelt.

[2] Epson Moverio BT-350: https://www.epson.de/products/see-through-mobile-viewer/moverio-bt-350
[3] Wikitude: Plattformunabhängiges SDK zur Umsetzung von AR-Anwendungen

3.2 Medizin und Pflege

Auch in der Medizin und Pflege lassen sich AR und IoT für Anwendungen kombinieren, um sowohl Ärzte als auch Patienten zu unterstützen. Die folgenden Arbeiten zeigen Beispiele dieses Zusammenspiels.

Agrawal et al. [AMPT18] entwickelten eine Handheld-AR-Anwendung, die es Patienten erleichtern soll, Herzuntersuchungen nachzuvollziehen. Dazu soll ein virtuelles Herz auf einem Smartphone angezeigt werden, um besser zu verdeutlichen, wo die Beschwerden des Patienten liegen. Ein NodeMCU[4] Mikrocontroller misst dabei den Puls des Patienten, um die Visualisierung des Herzens, durch einen echten Herzschlag, realistischer zu gestalten. Die Kommunikation zwischen NodeMCU und Smartphone wird via HTTP über die IoT-Plattform *ThingSpeak*[5] in der Cloud durchgeführt. Die Smartphone-App verwendet dabei Unity für das Anzeigen der AR-Elemente und Vuforia zum Tracking.

Y. J. Park et al. [PRLH19] entwarfen ein Projection-Based-AR-System zur Unterstützung älterer Menschen. Dieses System verwendet unter anderem eine 3D-Tiefenkamera, um die Haltung von Personen im Raum zu überwachen und Alarm zu schlagen, sobald jemand fällt. Außerdem kann es helfen Personen zu erkennen, da ältere Menschen Probleme mit dem Zuordnen von Gesichtern haben können. So kann z. B. das Live-Bild einer Türkamera analysiert, mit Namen und anderen Informationen angereichert, auf eine Wand projiziert werden. Das System kann außerdem Objekte wie Medikamentendöschen erkennen. So können Informationen wie beispielsweise die letzte Einnahme, neben den Döschen auf einen Tisch projiziert werden.

Shinde et al. [SDMD21] geben weitere Beispiele über potentielle Anwendungsgebiete für ein Zusammenspiel aus AR und IoT, auch aus dem Gesundheitssektor. Eine Möglichkeit wäre die Anwendung in einem Krankenhaus, in dem IoT beispielsweise dazu verwendet wird, den Blutzucker einer Patientin zu überwachen. Gleichzeitig kann ihre Position im Krankenhaus zur Indoor-Navigation verwendet werden. AR wird hier beispielsweise dazu verwendet, den Patienten über Eingriffe aufzuklären. Auch visuelle Referenzen bei Operationen können mit AR umgesetzt werden, was den operierenden Arzt unterstützen oder unerfahrenen Praktikanten helfen kann. Ein weiteres beschriebenes Anwendungsbeispiel befindet sich in einer Smart-Home-Umgebung. IoT wird hier verwendet, um die Bewohner zu überwachen und so z. B. deren Gewicht und andere Gesundheitsdaten zu erfassen. So kann auch das Training in einem Heim-Fitnessstudio entsprechend angepasst werden. AR wird in diesem Beispiel verwendet, um z. B. älteren Menschen als Navigationshilfe im Haus zu dienen.

3.3 Smart City

Die Datenerfassung durch IoT ist eine der wichtigsten Grundlagen für eine Smart City. Seit einiger Zeit wird auch hier versucht, Augmented Reality damit zu verknüpfen. Im Folgenden werden solche AR-IoT-Themen aus dem Gebiet der Smart Cities betrachtet.

[4] NodeMCU: Betriebssystem für ESP8266 oder ESP32 Mikrocontroller
[5] ThingSpeak: https://thingspeak.com

Pokrić et al. [PKP14] entwickelten eine Anwendung, um den Busfahrplan an Haltestellen in AR darzustellen. Hierbei sind die Informationen über die Bushaltestelle in einem QR-Code kodiert, welcher von einem Smartphone via Vuforia getrackt und gelesen wird.

Später beschrieben Pokrić et al. [PKP+15] auch einen Ansatz, um mittels Gamification[6] Bürger einer Stadt mehr in die Smart City einzubringen. Eine in Unity implementierte Smartphone-App dient als Anzeige. Dazu wird via markerlosem Tracking und der IMU des Smartphones eine gerade Fläche, wie ein Tisch gesucht, worauf ein virtueller Avatar projiziert wird. Der Avatar ändert z. B. bei hohen oder niedrigen Temperaturen und guter oder schlechter Luftqualität sein Outfit. Eine Gasmaske deutet so beispielsweise auf schlechte Luftqualität hin. Zusätzlich kann der Nutzer auch den tatsächlichen Wert der Temperatur oder Luftqualität in der App erraten. Eine Studie zeigte, dass die App Nutzer auf Umweltprobleme hinweist und sowohl lehrreich als auch unterhaltsam sei.

Zhang et al. [ZCDE18] demonstrieren einen Prototyp, um Daten einer Smart City, am Beispiel von Toronto, zu visualisieren. Als Ausgabe wird hierfür die Microsoft HoloLens mit dem Mixed Reality Toolkit (MRTK)[7] und Unity verwendet. Ist eine gerade Fläche ausgewählt, kann ein 3D-Modell der Stadt darauf projiziert werden. Der Nutzer kann hierbei die einzelnen Gebäude durch Gesten auswählen, um die darüber vorhandenen Informationen anzuzeigen.

Garcia Macias et al. [GAEA11] entwickelten mit *UbiVisor* eine Art IoT-Browser, um smarte Geräte und Services in der Umgebung zu entdecken. Dies soll das Problem lösen, bei der immer weiter steigenden Anzahl an IoT-Geräten und Services, diese zu finden. UbiVisor sammelt zusätzlich auch Daten zum Kontext des Nutzers. Die Software soll in Smartphones, Tablets etc. eingesetzt werden können, da diese Geräte, dank Kamera, IoT-Hardware in der Umgebung erkennen können und da ihnen viele personenbezogene Daten, wie den Standort und Nutzerprofile zur Bestimmung des Nutzungskontexts zur Verfügung stehen. Ein smartes Gerät soll via RFID, Bar- oder QR-Code detektiert werden.

3.4 Industrie 4.0

Einer der Hauptbereiche für die Nutzung von IoT ist das *Industrial IoT* (IIoT) der Industrie 4.0. Auch hier kann oft auf die Vorteile von AR zurückgegriffen werden, was im Folgenden betrachtet wird.

Revetria et al. [RTD+19] entwickelten einen Prototyp, um die Belastung von beladenen Metallregalen zu überwachen. Dazu erhält jedes Regalfach einen Drucksensor, welcher mit einem Arduino verbunden ist. Die gesammelten Daten gelangen via Wi-Fi an die ThingSpeak-Plattform. Diese Daten werden auch für eine weitere Analyse in MATLAB verwendet und als historische Werte gespeichert. Ein QR-Code an den Regalen erlaubt den Zugriff auf die relevanten Daten, um diese auf einem OST-HMD darzustellen.

[6] Gamification: Spielmechaniken in die reale Welt übertragen, um Nutzer zu motivieren

[7] MRTK: Bibliothek von Microsoft, die nützliche Funktionen für verschiedene HMDs zur Verfügung stellt

Hoppenstedt et al. [HSK+19] verwenden eine Microsoft HoloLens zur Visualisierung der Gasverteilung innerhalb einer Brennstoffzelle. So kann der jeweilige Status der Brennstoffzelle überwacht und bei Problemen Alarm geschlagen werden. Die Daten gelangen via MQTT von den Sensoren der Brennstoffzelle an die HoloLens, wo diese in einem dreidimensionalen Modell der Brennstoffzelle in Echtzeit visualisiert werden.

Minoufekr et al. [MSZP19] stellten einen Prototyp vor, um CNC-Maschinen mithilfe von Augmented Reality zu überwachen. Der Nutzen von AR soll hier darin liegen, dass der CNC-Maschinist zusätzlich zu Sensordaten auch in Echtzeit sehen kann, wo sich das Fräswerkzeug befindet, selbst wenn dieses gerade z. B. durch das Werkstück oder die Kühlflüssigkeit verdeckt wäre. Dabei sendet die Steuerung der CNC-Fräse die Sensordaten via einem auf TCP basierenden Protokoll an eine Microsoft HoloLens. Zur Visualisierung wurden die beweglichen Teile der Maschine und Teile des Gehäuses als CAD-Modell in Unity importiert.

Auch im Schiffbau kann Augmented Reality eingesetzt werden. So beschreiben Blanco-Novoa et al. [BFFV18] und Fernández-Caramés et al. [FFSV18] unter anderem den Aufbau einer auf AR, IIoT und Fog Computing basierenden Industrie-4.0-Werft in Spanien. Dabei sollte die Effizienz durch ein System auf Basis von Industrie 4.0 erhöht werden. Hierfür wurden AR-Systeme in verschiedenen Bereichen der Werft getestet. Dabei wurde jeweils mit einem Smartphone, Tablet und OST-HMD getestet. Tracking wurde markerbasiert, unter Verwendung von Vuforia oder ARToolKit, durchgeführt. Der Einsatz von Fog Computing soll dazu dienen, die Latenzen beim Abrufen von Daten für die AR-Szenarien zu reduzieren.

Eines dieser Szenarien ist die Verwendung von AR, um die Papierdokumentation zu reduzieren. Getestet wurde hierfür ein AR-System, bei dem Rohre mit Zusatzinformationen, wie dem Material und Zielort, versehen werden. Abbildung 3.2 zeigt hier wie ein QR-Code für das Tracking der Rohre verwendet wird. Auch für die Qualitätskontrolle kann AR verwendet werden. So kann nach einer Montage das reale Werkstück mit einem 3D-Modell abgeglichen werden und dem Arbeiter die Unterschiede angezeigt werden. Während der Montage können auch Schritt-für-Schritt-Erklärungen eingeblendet werden. Arbeiter können zusätzlich von einem Experten aus der Ferne angeleitet werden. In den großen Montagehallen und Lagerhäusern einer Werft kann es hilfreich sein, mit AR die Position von Bauteilen hervorzuheben, um diese schneller zu finden. Selbst in den Schiffen wurden AR-Systeme getestet. So wurde versucht Rohre und Leitungen in Schiffswänden in AR darzustellen. Dies ist speziell zur Wartung und Reparatur von Vorteil.

Mourtzis et al. [MSZV19] entwickelten ein Handheld-AR-System zum Einsatz in Lagerhallen. Hierbei sollen sowohl die Regale als auch die Paletten mit Waren einen QR-Code erhalten. Dies kann zum Einordnen der Produkte verwendet werden, indem sowohl der QR-Code an der Palette, als auch der am Regal abgescannt wird. So speichert das System, welche Waren in welchem Regal lagern. Dies funktioniert auch umgekehrt, um ein Produkt aus dem Lager zu entnehmen und dem System so den aktuellen Zustand zu übermitteln. Zusätzlich existiert eine vogelperspektivische Ansicht der Lagerhalle, um zu sehen, welche Waren wo lagern und wie die Auslastung der Lagerhalle ist. Schlussendlich kann die Standortinformation auch für eine Navigation hin zu den Waren, anhand von gescannten QR-Codes oder Namen, genutzt werden. Hier wird das GPS des Handys verwendet, um den Arbeiter in AR zu den Waren zu leiten.

Hanas Subakti und Jehn-Ruey Jiang [SJ18] entwickelten einen Prototyp für eine markerlose Mobile-AR-Anwendung zur Verwendung in Industrie-4.0-Umgebungen. Dabei wird nicht auf QR-Codes oder ähnliche Marker zurückgegriffen, sondern ein Deep-Learning-System verwendet.

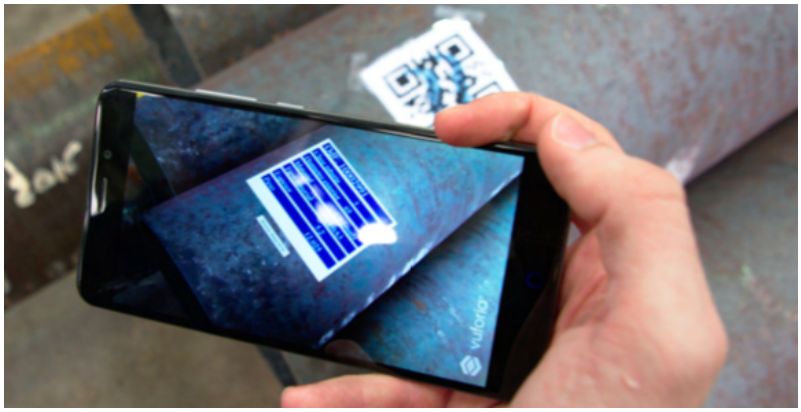


Abbildung 3.2: QR-Code basiertes Marker Tracking zur Darstellung von Zusatzinformationen für Rohre auf einem Smartphone [BFFV18].

Mithilfe von TensorFlow[8] wird es so möglich, eine vordefinierte Menge an Maschinen auf Basis von Videomaterial zu erkennen. Dieses Videomaterial kann vorher von einem Nutzer mit dem Smartphone aufgenommen werden und wird anschließend auf einem Computer mit TensorFlow verarbeitet. Die daraus generierten Daten werden wieder auf das Smartphone übertragen und dort verwendet, um die Maschine zu tracken. Gleichzeitig werden die Sensoren eines Tango-Smartphones[9] verwendet, um die AR-Inhalte an die Distanz vom Nutzer zur Maschine anzupassen.

Volker Paelke [Pae14] präsentiert einen Ansatz zur Verbesserung der Effizienz von Montagearbeitern in einer Industrie-4.0-Fabrik. Hierzu wird dem Arbeiter jeweils die Position des im nächsten Schritt benötigten Bauteils im Bauteileregal angezeigt. Sobald dieses herausgenommen wurde, wird ein dreidimensionales Bild des Montageschritts angezeigt. So kann der Arbeiter Schritt für Schritt die Montage durchführen, selbst ohne vorheriges Einlernen. Die Anzeige erfolgt über OST-HMDs, ist jedoch auch mit VST-HMDs kompatibel. Für das Tracking des Arbeiters, des Bauteileregals und der Bauteile wird die Kamera des HMDs, das Metaio SDK[10] und markerloses Tracking verwendet.

Pierdicca et al. [PFP+17] entwickelten auch eine AR-Anwendung zum Training von Montagearbeitern. Dabei soll die benötigte Zeit zum Einlernen von Mitarbeitern reduziert werden. So wurde eine Android-basierte Anwendung auf Basis von Vuforia und Unity entwickelt und auf einem Android-basierten VST-HMD ausgeführt. Das Tracking wurde durch 30 cm x 30 cm große Marker auf der Arbeitsfläche des Monteurs umgesetzt. Diese Größe wurde gewählt, da so Tracking trotz Verdeckung durch die Arme des Monteurs und des Werkstück möglich war. Die Interaktion erfolgte über einen Taster am HMD. Dem Monteur wird so in jedem Montageschritt durch ein 3D-Modell und eingeblendeten Text gezeigt, wie dieser Schritt durchzuführen ist. Die Anwendung sollte dabei gedruckte Montageanleitungen ersetzen und wurde zusammen mit einem Industriepartner getestet. Tests mit Experten und unerfahrenen Arbeitern zeigten, dass die Ausführungszeit der Montagearbeiten im Vergleich zu gedruckten Anleitungen, reduziert wurde. Ein weiterer Vorteil war, dass beide Hände des Monteurs zu jeder Zeit frei blieben. Es wurden jedoch auch Probleme festgestellt. Beispielsweise die schlechte Performance des ausgewählten HMDs und das Überhitzen

[8] TensorFlow: Googles Framework für maschinelles Lernen
[9] Tango: Eingestellte Android-AR-Plattform von Google für spezielle Smartphones
[10] Metaio SDK: AR SDK das 2015 von Apple übernommen wurde

des Geräts, was am Ohr des Anwenders spürbar war. Eine Interaktion ohne Taster via Sprache oder Gesten wurde auch als Anforderung für zukünftige Anwendungen genannt. Schlussendlich gab es auch Probleme mit dem verwendeten Marker Tracking, die möglicherweise durch markerloses Tracking verhindert werden können.

Evans et al. [EMP+17] liefern ein weiteres Beispiel für die Verwendung von AR bei Montagearbeiten. Hier wurde eine Anwendung mit Unity für die Microsoft HoloLens entwickelt. Die HoloLens ermöglicht zwar das räumliche Mapping der Umgebung, jedoch nicht in einer Genauigkeit, die für das Erkennen von Bauteilen ausreichend ist. So wurde hier und unter Verwendung von Vuforia Marker Tracking umgesetzt. Die Anwendung unterstützt den Monteur in jedem Montageschritt durch Informationen wie z. B. welche Teile eingesammelt werden sollen und wie diese montiert werden. Dabei zeigte sich, dass die HoloLens sehr gut für AR-Montagearbeiten verwendbar ist.

Henderson et al. [HF11] entwickelten eine AR-Anwendung zur Unterstützung von Feldmechanikern im Militär. Hier sollen diese speziell bei Reparaturen unterstützt werden. Zur Visualisierung wurde dabei ein VST-HMD verwendet. Tracking wurde via Infrarot-LEDs am HMD und retroreflektiven Markern an den Stellen, mit denen interagiert werden sollte, durchgeführt. Eine anschließende Studie, mit einem LCD für Instruktionen als Vergleich, zeigte, dass die Mechaniker das AR-HMD bevorzugten und damit die Aufgaben schneller erledigt hatten.

Alam et al. [AKBH17] erstellten ein System zur Unterstützung bei komplexen Wartungsarbeiten in extremen Umgebungen, wie schwer zugänglichen unterirdischen Anlagen. Als Beispiel dienen hier Wartungsarbeiten der Sensoren am Großen Hadronen-Speicherring des CERN[11]. Da die Mitarbeiter bei Wartungsarbeiten Strahlung ausgesetzt sind, ist es wichtig, schnell zu Arbeiten und alle wichtigen Parameter der Umgebung, wie Strahlungswerte zu messen. Hierzu verwenden Alam et al. ein OST-HMD-basiertes System mit IMU. Zusätzliche, am Körper befestigte Sensoren, erfassen z. B. die Strahlung. Auch das Detektieren von Schaltkästen geschieht via markerlosem Tracking bzw. hybridem Tracking mit den IMU-Daten zur Unterstützung.

Mansoni et al. [MFB+17] und Bordegoni et al. [BFC+14] stellen eine weitere Anwendung zur AR-Fernwartung vor. Die Anwendung basiert auf Unity und Vuforia. Ein Experte kann damit einen Arbeiter aus der Ferne einweisen und erklären, wie er die Wartungsarbeiten zu verrichten hat. Der Experte kann beispielsweise beim Arbeiter einen Pfeil anzeigen lassen, um zu verdeutlichen, in welche Richtung ein Ventil zu drehen ist. Das System kann sowohl via Tablet als auch via AR-Brille verwendet werden. Hierbei hatte die AR-Brille den Vorteil, dass der Arbeiter beide Hände frei hat und so die Wartungsarbeiten besser durchführen kann.

Tariq Masood und Johannes Egger [ME19] beschäftigten sich mit der Frage, welche Faktoren für den Erfolg von AR in der Industrie relevant sind. Hierzu wurde eine Studie mit 84 Teilnehmern als Online-Fragebogen durchgeführt. Der Fragebogen war in einen qualitativen und quantitativen Bogen unterteilt. Die Teilnehmer waren dabei alle bereits an einem AR-Projekt beteiligt. Es wurde nicht unterschieden, ob dieses erfolgreich war oder nicht. Die fünf wichtigsten Faktoren für den Erfolg von AR in der Industrie waren dabei:

- Ergonomie der Hardware
- Usability der AR-Nutzeroberfläche

[11] CERN: https://home.cern

- Verbesserungen der Effizienz durch AR
- Akzeptanz der Nutzer
- Sichtbarkeit von Informationen

Gleichzeitig wurde auch herausgefunden, dass in der Forschung oft lediglich auf die technischen Herausforderungen von AR eingegangen wird und nicht auf die, für die Industrie relevanteren, organisatorischen Probleme. Außerdem ist es wichtig für Unternehmen, vor dem Einsatz von AR zu überprüfen, ob die Technologie für den jeweiligen Einsatzzweck geeignet ist oder nicht. AR ist beispielsweise in Lagerhallen weit verbreitet, speziell für das Zusammentragen von Waren. Es ist auch zu bedenken, dass Prozesse oft für ein besseres Zusammenspiel mit AR adaptiert werden müssen und Personal geschult werden muss.

Yusuf et al. [YLH+20] befragten 200 Manager, die in Projekten bereits mit AR oder IoT oder beidem Erfahrungen gesammelt hatten. Viele sahen AR und IoT als sich ergänzende Technologien. 81% verwendeten dabei IoT und erprobten auch die Verwendung von AR. Von denen, die ausschließlich AR-Anwendungen entwickelten, gingen 76% davon aus, dass IoT ihre Anwendung bereichern würde. 51% verwendeten als AR-Geräte nur Smartphones, 39% verwendeten die Microsoft HoloLens und 18% Google Glass. Die Schwerindustrie war hierbei der Vorreiter bei AR-IoT-Mischanwendungen. Es werden auch drei Hauptarten der Verwendung von diesen Anwendungen beschrieben. Einmal, dass AR-IoT-Anwendungen Mitarbeitern helfen können, sich in der Umgebung zurechtzufinden, da die dreidimensional visualisierten Daten für die Nutzer einfacher verständlich sind. Diagnosearbeiten können auch deutlich erleichtert werden, da, ohne das Objekt zu zerlegen, Probleme identifiziert werden können, z. B. anhand von Daten von IoT-Geräten. Auch die Anleitung von Mitarbeitern durch Experten in der Ferne ist ein sehr verbreiteter Anwendungsfall für AR-IoT-Mischanwendungen.

3.5 Smart Home und Consumer IoT

Die folgenden Arbeiten gehen auf die Problematik, Augmented Reality zur Steuerung von Smart-Home-Geräten bzw. Consumer IoT zu verwenden, ein. Aus der Ähnlichkeit zum Thema dieser Arbeit folgt die hohe Relevanz und Anzahl der betrachteten, verwandten Arbeiten.

Blanco-Novoa et al. [BFAF20; BFVF20] versuchen, unter Verwendung der Microsoft HoloLens, das Problem der Heterogenität in IoT-Landschaften zu lösen. So wurde ein Middleware Framework unter Verwendung von MQTT mit dem MQTT Broker Mosquitto[12], HTTPS und dem Flow Editor Node-RED[13] entwickelt. MQTT wird zwischen den IoT-Geräten und der Middleware verwendet, REST zwischen Middleware und AR-Geräten. Im Rahmen eines Prototyps wird damit eine smarte Steckdose gesteuert. Die HoloLens scannt dazu einen auf dem Gerät befestigten QR-Code, um die ID der Steckdose zu erhalten. Tracking existiert darüber hinaus nicht. So ist die UI zum Steuern der Steckdose nicht an der Position der Steckdose verankert. Abbildung 3.3 zeigt diese UI.

Dongsik Jo und Gerard Jounghyun Kim [JK16] beschreiben mit *ARIoT* ein skalierbares AR Framework, um von überall mit IoT-Geräten zu interagieren. Die Besonderheit hinter diesem Framework ist das Tracking. Hierzu hat jedes IoT-Gerät Informationen über seine Natural Features

[12] Mosquitto: https://mosquitto.org
[13] Node-RED: https://nodered.org

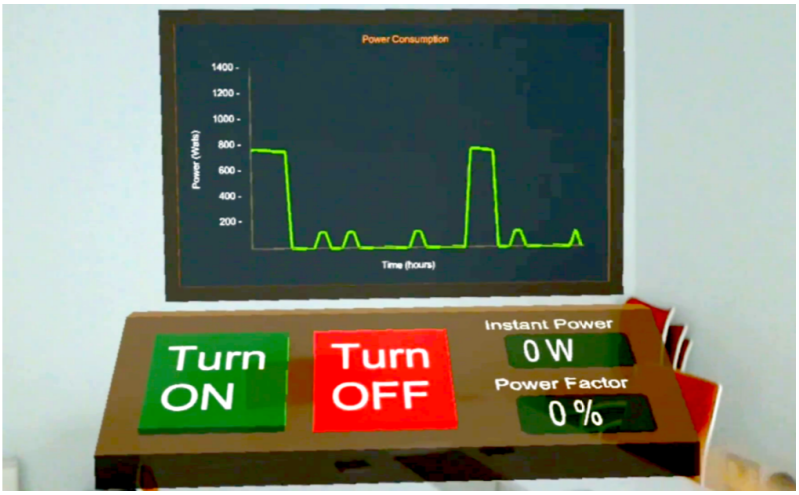


Abbildung 3.3: HoloLens-UI zur Darstellung des Stromverbrauchs an einer Smart-Steckdose und zur Steuerung dieser [BFAF20].

gespeichert und gibt diese an ein Endgerät in der Nähe weiter. Das Endgerät muss anschließend nur nach diesen Features suchen und nicht nach denen aller IoT-Geräte. Wird ein IoT-Gerät erkannt, kann der Nutzer die angebotenen Services nutzen. Somit ist das Framework skalierbar, da immer nur die IoT-Geräte in der Nähe getrackt werden müssen, was den Einfluss der Anzahl an IoT-Geräten im System für die Performance minimiert.

Koutitas et al. [KJG+18] demonstrieren eine AR-Anwendung zur Interaktion mit IoT-Geräten. Hierbei wird eine Microsoft HoloLens als AR-Gerät verwendet. Das Backend wird in der Edge ausgeführt und kommuniziert via HTTP mit dem AR-Gerät. Die IoT-Geräte verwenden ein ZigBee-Netzwerk zur Kommunikation mit einem Gateway, welches mit dem Backend verbunden ist. Die Demonstration der Anwendung beinhaltet dabei das Steuern einer LED an einem Arduino via HoloLens und das Anzeigen eines Temperaturwerts in einem 3D-Modell des Arduinos via Unity.

Sun et al. [SAR+19] entwickelten mit *MagicHand* ein System, welches das Steuern von smarten Geräten via Handgesten ermöglicht. Hierzu wird ein neuronales Netz zusammen mit einer Microsoft HoloLens und einer zusätzlichen Tiefenkamera für Echtzeit-Tracking der Hände verwendet. Das System ermöglicht auch das markerlose Tracking der smarten Geräte. Für die Interaktion werden eigens implementierte Gesten, wie ein Wischen nach rechts, um bei einem Lautsprecher zum nächsten Lied zu wechseln, verwendet. Um die Hände besser zu tracken trägt der Nutzer ein buntes Armband. Als Feedback wird dem Nutzer eine holographische UI via HoloLens über den Geräten angezeigt.

S. Park et al. [PCK+20] entwickelten eine AR-Anwendung zur Steuerung von Smart-Home-Geräten ohne Hände. Die AR-Anwendung verwendet dabei die Microsoft HoloLens als Anzeige. Mithilfe eines sogenannten Elektrookulogramms wird Blinzeln des Nutzers erkannt, was zur Interaktion verwendet wird. Die Anwendung ist dabei speziell für Ältere oder Menschen mit Behinderungen gedacht.

Ähnlich dazu entwarfen Mahroo et al. [MGS19] mit *HoloHome* einen AR-Prototyp zur Steuerung von Smart-Home-Geräten via Microsoft HoloLens. Die Rolle der Smart-Home-Geräte wird dabei durch Arduinos mit Wi-Fi eingenommen. Das Tracking wurde via Vuforia umgesetzt. Die AR-

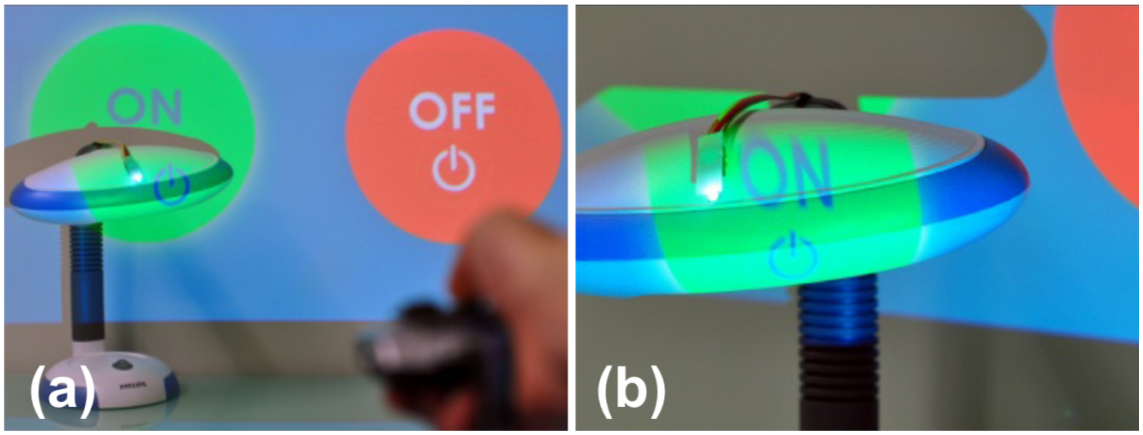


Abbildung 3.4: (a) Zeigt die projizierte UI einer Lampe. (b) Wie der „ON“-Button über dem Sensor positioniert wird, um die Lampe zu steuern [SMC12].

Anwendung kann damit Geräte erkennen und steuern. Gleichzeitig kann sie auch Ältere oder Menschen mit Behinderung dabei unterstützen, Objekte wiederzufinden. Die AR-Komponente wurde dabei in Unity auf Basis des MRTKs entwickelt.

Jang et al. [JB18] entwickelten mit *HoloSensor* eine Anwendung zur Visualisierung von Sensordaten unter Verwendung einer Microsoft HoloLens. Die Sensoren sind hierbei an Arduinos angeschlossen, welche mit dem Backend via Bluetooth kommunizieren. Die HoloLens kommuniziert mit dem Backend via REST. Für die AR-Visualisierung wird das Microsoft HoloToolkit (mittlerweile MRTK) in Unity verwendet. Dabei wird der Arduino nicht getrackt, was bedeutet, dass die UI immer vor dem Nutzer positioniert ist.

Ullah et al. [UIAH12] berichten über eine Anwendung, bei der ein Android Smartphone verwendet wird, um Smart-Home-Geräte zu steuern. Das Backend ist hier eigens für diesen Zweck entwickelt. Mithilfe eines abgescannten QR-Codes wird das zu steuernde Gerät eindeutig identifiziert. Dabei dient der QR-Code nicht als Marker, was auch bedeutet, dass die Anwendung kein Tracking verwendet und die Steuerung somit nicht im Raum registriert ist.

Schmidt et al. [SMC12] entwickelten mit *PICOntrol* eine projektorbasierte AR-Anwendung zum Steuern von Smart-Home-Geräten. Hierbei wird ein kleiner Pico-Projektor verwendet, welcher mit Tastern zur Auswahl von Inhalten erweitert wurde. Jedes zu steuernde Gerät benötigt einen Mikrocontroller und mindestens einen Sensor, bestehend aus einer Status-LED und einem Photosensor. Der Nutzer steuert das Gerät, indem er mit dem Projektor, welcher eine UI anzeigt, auf den Sensor am Gerät zeigt. Dabei wird wie in Abbildung 3.4 das UI-Element, was auf den Sensor projiziert wird, von diesem erkannt und ausgewählt. Ein Druck eines Tasters am Projektor bestätigt die Auswahl.

Mishra et al. [MKBD20] entwickelten eine auf Unity basierende Android-AR-App für Smart Homes. Die App soll hierbei die Objekte, mit denen interagiert werden kann, im Kamerabild erkennen, wie z. B. Lichtschalter, Ventilatoren etc. Das Tracking ist dabei markerlos, per Deep Learning auf Bildern der zu steuernden Geräte implementiert. Ein NodeMCU-Mikrocontroller wird als Middleware verwendet, um zwischen den verbundenen Smart-Home-Geräten und der AR-Anwendung Daten auszutauschen.

Purmaissur et al. [PTG+18] entwickelten ein AR-System, um Nutzern den Energieverbrauch ihrer Geräte in Echtzeit anzuzeigen. Hierbei wird Node-Red als Middleware auf einem Raspberry Pi verwendet, um mit smarten Steckdosen zu kommunizieren. Diese liefern Informationen über den Stromverbrauch der eingesteckten Geräte. Es werden auch historische Werte gespeichert. Ein Nutzer kann so die Geräte mit zu hohem Verbrauch identifizieren und, wenn gewünscht, abschalten. Die UI der Anwendung wird auf einem Smartphone in AR angezeigt. Dabei wird via Vuforia ein QR-Code auf dem jeweiligen Gerät gescannt.

Huo et al. [HCY+18] präsentieren mit *Scenariot* eine Methode, um mithilfe von Simultaneous Localization and Mapping (SLAM)[14] IoT-Geräte zu erkennen und zu lokalisieren. Nutzer können anschließend mit diesen Geräten via Smartphone AR interagieren. Alle IoT-Geräte und das Smartphone besitzen dazu Ultrabreitband (UWB)[15]-Sender und -Empfänger. Nach initialer Kalibrierung können so Abstände zwischen den Geräten berechnet werden. Das verwendete Smartphone verfügt zusätzlich zu UWB auch über Tango Hardware. So kann ein Nutzer durch die Umgebung laufen und es werden ihm verfügbare IoT-Geräte angezeigt. Wenn sich der Nutzer einem Gerät nähert, bekommt er potentielle Interaktionsmöglichkeiten angezeigt.

Marques et al. [MDA+20] demonstrieren einen Prototyp, um AR-Steuerungen für Smart-Home-Szenarien nutzerkonfigurierbar zu gestalten. Hierzu wird ein Tango-fähiges Smartphone als Anzeige, zusammen mit QR-Code-basiertem Marker Tracking, verwendet. Tango wird hier eingesetzt, um z. B. die UI perspektivisch korrekt auf Objekte wie Tische und Wände zu mappen. Der Prototyp erlaubt es Nutzern auch Modelle auszuwählen, die anschließend auf der Position eines QR-Codes angezeigt werden. Der Nutzer kann sich hiermit seine eigene Smart-Home-Anwendung zusammenstellen, da er die 2D-Interfaces zum Steuern der Geräte auf einen QR-Code mappen und frei positionieren kann.

Mayer et al. [MHS14] entwickelten ein System, um IoT-Geräte und deren Verbindungen via Smartphone oder Tablet zu erkennen und hervorzuheben. Dabei werden die Netzwerkpakete auf die HTTP-Header überprüft, um zu ermitteln, welches Gerät der Empfänger und welches der Sender des Pakets ist. Zusätzlich wurde eine „Magic Lense“-App entwickelt, mit der ein Nutzer in AR auf seinem Endgerät die verfügbaren IoT-Geräte und Kommunikationsverbindungen angezeigt bekommt. Hierzu wird die Kamera des Geräts und OpenCV für markerloses Tracking verwendet.

Hua et al. [HLRJ21] entwickelten mit *ArcIoT* ein weiteres Mobile-AR-System zur Steuerung von smarten Geräten und IoT. Das Erkennen der Geräte wurde, wie in Abbildung 3.5 zu sehen, via markerlosem Tracking mit Vuforia umgesetzt. Das Problem mehrerer gleich aussehender Geräte wurde mithilfe von BLE Beacons an den jeweiligen Geräten gelöst. So werden anhand der Signalstärken der Beacons und der IMU des Smartphones die Geräte kartographiert. Diese Karte wird anschließend verwendet, um zu erkennen, welches Gerät sich wirklich im Kamerabild des Smartphones befindet. Die Karte ist dabei in 2D-Koordinaten relativ zu dem Punkt, an dem der Nutzer die App das erste Mal geöffnet hat, aufgebaut.

Xia et al. [XWW+21] entwickelten mit *INSIGHT* eine Mobile-AR-Anwendung zur Interaktion mit IoT-Geräten. Auch hier wird Vuforia für markerloses Tracking eingesetzt. Dabei wird Vuforia nur verwendet, um Gerätetypen zu erkennen. Anhand der IMU des Smartphones und einer Datenbank, mit bereits einmal erkannten Geräten, wird anschließend versucht herauszufinden, welches Gerät

[14] SLAM: Das gleichzeitige Kartographieren der Umgebung und Lokalisieren innerhalb dieser Karte
[15] UWB: Funk mit niedriger Energie und Reichweite aber hoher Bandbreite

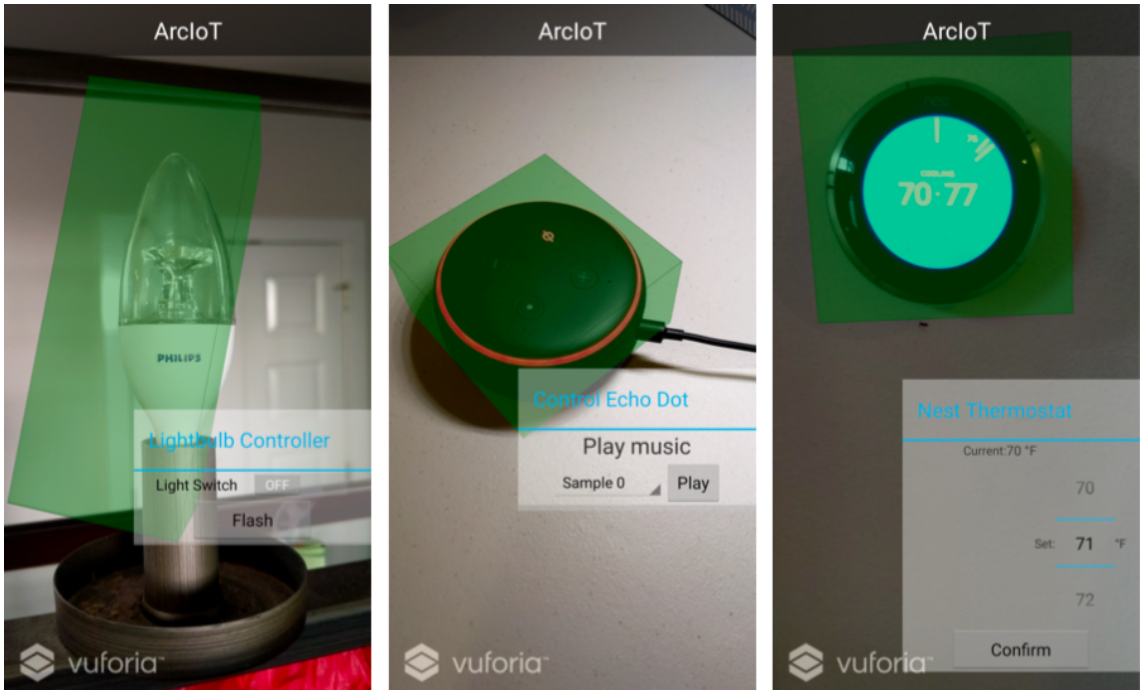


Abbildung 3.5: Verschiedene AR-UIs zur Steuerung von IoT-Geräten mit ArcIoT [HLRJ21].

sich vor dem Nutzer befindet. Wenn das Gerät nicht in der Datenbank gefunden wird, wird es manuell hinzugefügt. Ansonsten wird die dort gespeicherte IP-Adresse verwendet, um mit dem Gerät zu kommunizieren. So kann auch dieser Ansatz mit mehreren gleich aussehenden Geräten umgehen, ohne auf Marker zurückzugreifen.

Wirtz et al. [WRS+15] versuchen mit *STIF* das Problem zu lösen, dass viele IoT und smarte Geräte ihre eigenen Apps zur Interaktion benötigen. Die Kommunikation zu den smarten Geräten wird dabei von einer APP ausgehend via Visible Light Communication (VLC), audiobasierter Kommunikation, NFC, Wi-Fi oder BLE durchgeführt. So ist eine direkte Kommunikation zwischen STIF und dem jeweiligen Gerät möglich. Die UI kann in AR oder zweidimensional in der APP dargestellt werden. Für das Tracking der smarten Geräte in der AR UI wird Vuforia verwendet.

Auch Becker et al. [BRS19] versuchen das Problem zu lösen, dass viele IoT- und Smart-Home-Geräte auf die verschiedensten Arten gesteuert werden. Um dies zu vereinheitlichen, entwickelten sie eine AR-Anwendung auf Basis der Microsoft HoloLens zur Steuerung dieser Geräte. Dabei sollten drei Arten der Interaktion verglichen werden. Die erste war dabei das einfache Anzeigen von Widgets über dem zu interagierenden Objekt, gesteuert von den Standard HoloLens Pinch-Gesten. Abbildung 3.6 zeigt eines dieser Widgets zur Temperaturregulierung. Die Zweite war ein Armband, das verschiedene Gesten erkennen kann, wie ein Winken, Faust ballen, Finger spreizen etc. Das dritte Interaktionsparadigma war das Verwenden von realen, getrackten Objekten zur Steuerung, z. B. eine Teetasse die gedreht wird, um die Lautstärke eines Lautsprechers zu verändern wie ein Potentiometer an einem Radio. Eine Studie verglich diese Interaktionsparadigmen und zeigte, dass das erste Szenario mit den AR Widget über den jeweiligen Geräten am besten bei den Nutzern

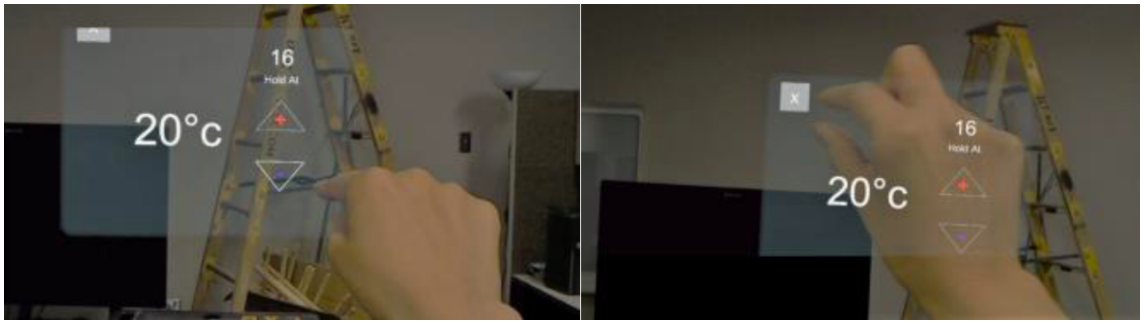


Abbildung 3.6: Beispiel-Widget zur Steuerung der Raumtemperatur via HoloLens-Gesten [BRS19]

ankam. Für die technische Umsetzung lief eine Middleware auf einem Smartphone und sendete Daten per Wi-Fi an die HoloLens. Das Tracking funktionierte via Markern und Vuforia. Für das Interface wurde Unity verwendet.

White et al. [WCPC19] präsentieren ein System, um Kontextinformationen für IoT-Geräte in AR zu geben. Hierfür werden Vuforia und Unity zusammen mit dem ARCore[16] AR SDK auf Android verwendet. Tracking wird anhand von eigenen Markern (Bilder von Kieselsteinen) durchgeführt. White et al. veranschaulichen, wie ein solches System von Technikern genutzt werden kann, um schnell Fehler zu finden, indem ihnen z. B. Quality-of-Service-Informationen oder die internen Komponenten eines Geräts anzeigt werden. Außerdem bemängeln sie, dass die bisherige Forschung nicht auf den Kontext des Nutzers, wie die Uhrzeit, Nutzerpräferenz, Wetter etc. für AR-Systeme eingehe.

Das *ERINOKS*-Küchensystem von Tanríseven et al. [TUKE19] verwendet Augmented Reality zum Steuern von smarten Herden und Anzeigen von Kochrezepten. Hierbei wurden Induktionsherde durch ein IoT-Gateway zu IoT-Herden. Diese senden Daten, wie die Essenstemperatur und -menge per MQTT an das System. Hier kann ein virtueller Assistent, der mithilfe einer HoloLens und Unity umgesetzt wurde, auf die Daten zugreifen. So kann der Assistent dem Nutzer das Kochrezept Schritt für Schritt erklären und gleichzeitig das Essen überwachen.

Ankireddy et al. [ARRA19] nehmen sich dem Problem der immer weiter steigenden Anzahl an IoT-Anwendungen an, indem sie IoT-Geräte via AR steuern. Die Arbeit ist dabei zweiteilig. Anfänglich wird eine AR-App unter Verwendung von Vuforia und Unity vorgestellt. Vuforia wird hierbei zum markerlosen Tracking verwendet. Als IoT-Gerät kommt ein Raspberry Pi mit einem Vier-Wege-Relais zum Einsatz. Dieses lässt sich so über die AR-App steuern. Der zweite Teil der Arbeit behandelt eine selbst entwickelte kostengünstige AR-Brille. Hierfür wird ein Raspberry Pi Zero W mit Touch-Sensoren verwendet. Die Interaktion wurde via einer Pi-Kamera, die auf einer Brille montiert ist, umgesetzt. Schaut der Nutzer in Richtung eines IoT-Geräts und tippt auf einen der Touch-Sensoren, wird eine Funktion, wie das Schalten der Relais, ausgeführt.

Iqbal et al. [IAAH16] stellen ein System zur Steuerung von Geräten im Haushalt über Gesten oder Sprache vor. Die Gesten und Sprachkommandos werden dabei von einer Microsoft Kinect interpretiert. Eine dieser Gesten ist beispielsweise das Zeigen auf ein Gerät, um dieses einzuschalten.

[16] ARCore: https://developers.google.com/ar

Gesteuert werden die Geräte über ein X10-Smart-Home-System. Dieses System soll speziell zur Bedienung durch Ältere oder Menschen mit Behinderung geeignet sein, was durch eine Studie mit Menschen dieser Zielgruppe gezeigt wurde.

Knierim et al. [KWAS19] erforschten das Potential von Augmented Reality im Zuhause. Es wurde eine Online-Umfrage mit 60 Teilnehmern durchgeführt. Diese Umfrage sollte einen Eindruck über die Meinung von Nutzern zu AR geben. Zusätzlich wurde eine 14-tägige Studie mit vier Haushalten, gefolgt von einem semistrukturierten Interview, durchgeführt. Dabei wurde den Teilnehmern zuerst eine Microsoft HoloLens demonstriert und anschließend über einen Zeitraum von zwei Wochen die Aufgabe gegeben, jede Interaktion in ihrem täglichen Leben, die sie mit AR verbessern würden, festzuhalten. Hierzu sollte in einer App das Szenario skizziert werden. Insgesamt zeigte sich, dass die Teilnehmer der Studien Augmented Reality im Zuhause positiv gegenüber stehen.

Prange et al. [PSP+21] entwickelten mit *PriView* einen Prototyp zur Visualisierung von potentiellen Privatsphärerisiken durch smarte Geräte in einem Raum. Hierzu wurden unter anderem die smarten Geräte mithilfe einer Wärmebildkamera erkannt. Als Feedback wurde entweder ein Smartphone oder ein VST-HMD (HTC Vive Pro) verwendet. Eine damit angelegte Studie zeigte, dass die Nutzer die Tatsache, dass das HMD die Hände frei hält, bevorzugten.

Haesler et al. [HKBW18] zeigen wie Augmented Reality auch rein als Feedback für eine Smart-Home-Steuerung verwendet werden kann. So wird unter anderem ein OST-HMD verwendet, um einen dreidimensionalen Avatar für den Amazon-Alexa-Sprachassistenten anzuzeigen. Dabei wurde untersucht, ob sich so das Vertrauen von Menschen in Sprachassistenten wie Amazon Alexa in Smart-Home-Umgebungen erhöhen lässt. Eine Studie mit Amazon Alexa ohne Avatar als Vergleich zeigte ein höheres, jedoch nicht signifikant höheres, Vertrauen in den Avatar.

Auch außerhalb des Augmented-Reality-Bereichs gibt es Ansätze Smart-Home-Geräte zu steuern. So kombinierten Eckstein et al. [EKEL19] Virtual Reality mit Smart Home. Die Vision dahinter ist es, dem VR-Erlebnis eine zusätzliche Ebene an Immersion zu geben. Diese wird durch das Steuern von physischen Objekten in der realen Welt erreicht. So kann z. B. ein Ventilator im virtuellen Raum angeschaltet werden, was der Nutzer anschließend real spüren kann. Die VR-Umgebung wurde hierbei in Unity erstellt und via HTC Vive VR-Brille angezeigt.

Ähnlich dazu entwickelten Simiscuka et al. [SM18] mit *VRITESS* ein System zur Steuerung von IoT Geräten in VR. Dabei wurde testweise ein Raspberry Pi mithilfe einer Oculus Rift VR-Brille gesteuert. Die VR-Umgebung wurde auch hier mit Unity erstellt.

3.6 Architektur von AR-IoT-Systemen

Im Folgenden wird auf Arbeiten eingegangen, welche die Architektur von AR-IoT-Mischanwendungen behandeln oder Einblicke in den Aufbau und die Funktionsweise solcher Systeme geben.

So analysierten Makolkina et al. [MPK+18] die Latenz zwischen Cloud-Plattform und AR-Gerät. Um die Latenz zu optimieren wird eine neue Architektur für AR-IoT-Mischanwendungen entwickelt. Die sogenannte Quality of Experience ist dabei ein Hauptfaktor für eine für den Nutzer angenehme AR-Anwendung und ist abhängig von den Latenzen zwischen AR und IoT. Oft gelangen Daten nicht direkt an die AR-Anwendung, sondern lagern in großen Datenbanken in der Cloud. Dieses Delay zwischen

der Anfrage an die Cloud und dem Erhalt der Daten wurde mit gängigen IoT-Cloud-Anbietern analysiert. Als Beispiel verwendeten Makolkina et al. hier einen Feuchtigkeitssensor an einer Pflanze, zusammen mit Temperatur- und Luftfeuchtigkeitssensoren, an einem NodeMCU-Mikrocontroller. Die dabei generierten Daten wurden anschließend an die verschiedenen Cloud-Anbieter gesendet. Das AR-Endgerät das die Daten abrufen sollte war hierbei ein Smartphone mit einer Unity-Anwendung, die Vuforia für das Tracking verwendet. Die betrachteten Cloud-Anbieter waren *ThingSpeak*, *thinger.io*[17], *Ubidots*[18] und *AWS*. ThingSpeak hatte hierbei das geringste Delay mit durchschnittlich einer Sekunde. Auch das Delay für das Vuforia-Tracking wurde gemessen. Hierbei zeigte sich, dass dieses größer wird, je mehr Objekte in der Datenbank zum Tracken gespeichert sind. Bereits bei zwei Objekten wurde ein Delay von ca. einer Sekunde gemessen. Für das Cloud-Delay wurde die Lösung einer mehrschichtigen Architektur vorgeschlagen. Das Endgerät fragt so zuerst die ihm am nächsten liegende Datenbank an, ähnlich zu Fog Computing. Ist die Information dort nicht vorhanden, wird bei der nächst entfernten Ebene angefragt. Je näher am Endgerät die Request bearbeitet werden kann, desto schneller ist somit die Antwort. So konnte die Antwortzeit der Anwendung halbiert werden.

Auch Tan et al. [TQZ+20] befassen sich mit dem Latenzproblem von AR-Anwendungen. Hier wird auch das Fog-Computing-Paradigma angewandt. Statt die Datenbanken näher zur Anwendung zu bringen, wird hier der Kommunikationskanal optimiert. Dazu werden sogenannte Unmanned Aerial Vehicles (UAVs) bzw. Drohnen für den kabellosen Datenaustausch verwendet. Der Vorteil dieser ist ihre Beweglichkeit. So kann die Position der Drohnen je nach Auslastung optimiert und damit die Latenz des Kommunikationskanals reduziert werden.

Sandhya Baskaran und Hari Kumar Nagabushanam [BN18] nehmen sich dem Problem an, dass das Training von AR-Anwendungen auf das Tracking der immer weiter steigenden Anzahl an IoT-Geräten nicht praktikabel sei. Die Lösung hierzu sei es, die IoT-Geräte anhand von Gateways zu detektieren. Diese Gateways sind dazu im gesamten Bereich des Netzwerks verteilt. Dabei speichert das Gateway die Position aller IoT-Geräte in seinem Bereich. Diese Position wurde zuvor über die Recieved Signal Strength (RSS) und Angle of Arrival (AoA) des Funksignals der Geräte bestimmt. Beim Eintreten des AR-Geräts in den Bereich wird auch dessen Position relativ zum Gateway auf dieselbe Art bestimmt. So wird kein optisches Tracking benötigt, um IoT-Geräte zu detektieren. Dadurch skaliert dieser Ansatz besser mit einer sehr hohen Anzahl an IoT-Geräten.

Ähnlich dazu entwickelten Y. Park et al. [PYK19] ein System namens *VisIoT*. Die Herausforderung bei AR-IoT-Mischanwendungen ist, laut Y. Parkt et al., einerseits die IoT-Geräte im Video-Stream zu lokalisieren und andererseits diese Position dem Datenstrom eines Geräts zuzuordnen. Die Herangehensweise von VisIoT dazu ist, nicht die Position des AR-Geräts in Welt-Koordinaten zu bestimmen und so das IoT-Gerät zu finden, sondern direkt das IoT-Gerät in den 2D-Kamera Koordinaten des AR-Geräts zu finden. Dazu wird ZigBee verwendet und die Distanz zu den IoT-Geräten, die Angle of Arrival und die IMU des AR-Geräts verwendet. Der Vorteil davon ist die Geschwindigkeit, da keine Bilderkennung verwendet wird.

[17] thinger.io: https://thinger.io
[18] Ubidots: https://ubidots.com

Campbell et al. [CGK+13] entwickelten mit *SIXTH* eine Java-basierte Middleware, um Sensoren mit AR-Geräten zu verbinden. Ein Sensoradapter sendet dabei Daten an die Middleware. Diese stellt eine REST-API zur Verfügung, um Sensordaten abzurufen. Gezeigt wurde die Funktionalität mit einem HMD ohne Sensoren, welches unter anderem Sensordaten von einem Smartphone erhielt, um so z. B. Headtracking umzusetzen.

Son et al. [SHL12] kritisieren, dass AR-Anwendungen nicht auf den Kontext, wie die Uhrzeit oder Geräte in der Nähe, eingehen würden. Hierzu wird ein Prototyp, um semantische Kommunikation umzusetzen entwickelt. Bilderkennung wird hier verwendet, um Geräte zu erkennen, die der Nutzer im Blick hat. Geräte erhalten vom System periodisch einen Broadcast, um andere Geräten in der Nähe zu erkennen. Sobald solch ein Gerät diesen Broadcast empfangen hat, sendet es Informationen über seine Funktionen und verfügbaren Services zurück. Ein sogenannter „Communication Core“ des Systems analysiert die Nachricht und versucht den Inhalt zu verstehen. Dazu wird eine Knowledgebase verwendet. Falls damit die Semantik der Nachricht noch nicht entschlüsselt werden kann, wird wieder mit dem Gerät kommuniziert, um weitere Informationen zu erhalten und der Vorgang wiederholt sich. Sollte dabei neue Information gesammelt werden, wird diese der Knowledgebase hinzugefügt. Anhand aller Antworten von den Geräten in der Nähe wird in einem „Context Manager“ der Kontext des Nutzers interpretiert und entschieden, was dem Nutzer in einer AR-UI angezeigt werden soll.

Fleck et al. [FSA20] entwickelten ein System zum Erstellen von AR-Applikationen nach Web-Prinzipien. So kann dabei eine AR-Applikation in JavaScript entwickelt werden und später trotzdem via Unity, was normalerweise C# verwendet, ausgeführt werden. Hierfür wird das Unity Framework *PowerUI* zum Interpretieren von JavaScript und Darstellen von HTML-basierten UIs verwendet. Die Verwendung von Web-Technologien soll hierbei das benötigte Knowhow zum Erstellen von AR-Anwendungen verringern, um diese für mehr Nutzer und Anwendungsfälle zugänglich zu machen.

Onoriode Uviase und Gerald Kotonya [UK18] betrachten die Architekturen von gängigen IoT-Frameworks und präsentieren anschließend eine verbesserte Architektur. Dabei sind Service Oriented Architectures (SOA) am verbreitetsten für das Verbinden von IoT Geräten und Anwendungen. Die betrachteten Frameworks waren das *Eclipse Smarthome*[19], *Calvin*[20], *SOCRADES*[21], *AllJoyn*[22], *FRASAD* [NTBG15], *ARIoT* [JK16] und *AVIoT* [JJH+15]. Oft wird dabei eine Architektur mit den Sensoren und Aktuatoren bzw. den Things auf einer Seite und der Anwendung auf der anderen, verbunden über eine Middleware, verwendet. Durch die steigende Anzahl an Things wird die fehlende Skalierbarkeit dabei zum Problem. Die Lösung ist laut Uviase und Kotonya eine Microservice-Architektur. Dabei sollen die IoT-Systeme in kleine Services aufgeteilt werden, die unabhängig voneinander laufen und dabei jeweils nur eine Funktionalität bereitstellen. Die Services werden unterteilt in funktionale und nichtfunktionale Services. Funktionale Services sind dabei solche, die IoT-Daten abrufen während nichtfunktionale Services die von dem System selbst verwendeten Services sind, wie z. B. die Authentifizierung. Ein sogenannter „Service Assembler“ behandelt die Requests der Nutzer und kombiniert je nach Anfrage Services, um dem Nutzer die gewünschten Ergebnisse zu liefern. Dies geschieht jedoch so, dass die Services sich nach dem

[19] Eclipse Smarthome: https://projects.eclipse.org/projects/iot.smarthome
[20] Calvin: https://github.com/EricssonResearch/calvin-base
[21] SOCRADES: http://socrades.net/Home/default.html
[22] AllJoyn: https://openconnectivity.org/technology/reference-implementation/alljoyn/

initialen Aufruf des Service Assemblers gegenseitig aufrufen und nicht wie bei SOA alle über einen Mediator in der Middleware zentral aufgerufen werden. Damit kann die Antwortzeit reduziert werden. Der Service Assembler wird außerdem über eine API aufgerufen, was bedeutet, dass für den Kunden die Nutzung des Services gleich bleibt, selbst wenn sich intern etwas ändert.

3.7 Weitere Anwendungsgebiete

Hier folgen weitere Beispiele für AR-IoT-Mischanwendungen aus Bereichen, die bisher noch nicht behandelt wurden.

Steiger et al. [SGA19; SKKA21] präsentieren mit *HoloFlows* eine Modellierungsanwendung, um IoT-Abläufe in AR zu modellieren statt via Desktop-Programm. Da IoT nicht nur in großen Firmen von trainiertem Personal, sondern auch bei Heimanwendern in Form von Smart Homes Verbreitung findet, soll das Erstellen von Workflows auch für diese Nutzergruppe einfacher gestaltet werden. Die Idee ist dabei, IoT-Geräte zu verbinden, indem sie in AR mit virtuellen Kabeln verbunden werden. Als AR-Gerät wird die Microsoft HoloLens mit den eingebauten Gesten verwendet. Dabei wird der jeweilige Status und verfügbare Funktionen direkt über dem IoT-Gerät als Hologramm angezeigt. Es gibt auch die Möglichkeit, Workflows direkt zwischen zwei IoT-Geräten zu erstellen. So kann z. B. ein Farbsensor mit einer Philips Hue Lampe verbunden werden, um die erfasste Farbe anzuzeigen. Als Middleware, um mit IoT-Geräten verschiedener Hersteller zu kommunizieren, wird *OpenHAB*[23] verwendet.

Karadeniz et al. [KAKE19] betrachteten die Verwendung von Digital Twins für „eGastronomic Things“. Ein Digital Twin ist ein digitales Gegenstück des realen Objekts. Hierbei wurde eine Fallstudie mit einem Digital Twin einer Eiscrememaschine durchgeführt. Es werden sowohl AR als auch VR verwendet, um den Digital Twin zu visualisieren. Dabei wurde die Unreal Spiele-Engine[24] für VR verwendet, wegen besserer Graphik und Unity wegen besserer Effizienz für AR mit einer Microsoft HoloLens.

Cao et al. [CXL+19] zeigen in ihrer Arbeit mit *V.Ra* ein System, um Roboter in AR zu programmieren. Dabei wird ein Smartphone einerseits zur Erstellung der Aufgaben für den Roboter verwendet, andererseits werden auch dessen Sensoren für SLAM verwendet. Das verwendet Smartphone muss Tango-fähig sein. Um den Roboter zu programmieren, muss ein Nutzer die Aktionen, die er ausführen soll, auf dem Smartphone in einer Unity-basierten Anwendung aufzeichnen.

Suzuki et al. [SBPI14] präsentieren ein AR-System für smartes Gebäudemanagement. Dabei sammeln IoT-Geräte verschiedenste Informationen des Gebäudes und senden diese an ein zentrales Management-System via MQTT. Dort werden diese ausgewertet und bei überschrittenen Schwellwerten eine Warnung an einen Techniker gesendet, der sich des Problems annehmen sollte. Dabei wurde eine Smartphone-App entwickelt, die dem Techniker in AR die Aufträge und benötigten Dokumente wie z. B. Pläne dazu anzeigt.

[23] OpenHAB: https://www.openhab.org
[24] Unreal Spiele-Engine: https://unrealengine.com/

Huang et al. [HLL12] entwickelten eine AR-Anwendung, um in einer Bücherei einfacher Bücher zu finden. Dazu werden die Buchrücken gescannt und mit Daten einer Cloud verglichen. Dort wird der Buchrücken mit einer Datenbank aus Büchern abgeglichen, um das Buch zu finden, das der Nutzer sucht. Diese Information wird anschließend verwendet, um dieses Buch für den Nutzer im Kamera-Livebild zu finden und mit Zusatzinformationen zu augmentieren.

Nestor Lobo [Lob16] entwickelte mit *Intelli-Mirror* einen Spiegel der mittels Bilderkennung eine Person vor ihm erkennt und ausgewählte Kleidung auf deren Körper anzeigt. Für die Detektion wird OpenCV verwendet. Implementiert wurde der Spiegel mit einem Raspberry Pi, welcher mit dem Internet verbunden ist, um z. B. neue Kleider im Inventar zu laden.

Dongsik Jo und Gerard Jounghyun Kim [JK19] zeigen mit *Digi-Log* eine weitere Demonstration eines Smart-Shopping-Systems das AR und IoT verbindet. Dabei verwendet der Nutzer eine Mobile-AR-Anwendung auf seinem Smartphone in einem Laden. So kann z. B. beim Lampenkauf mit dem Smartphone über jeder Lampe ein Schalter angezeigt werden, um diese zu testen. Bei der Auswahl von Lautsprechern kann so auch ein Interface für das Abspielen von Musik angezeigt werden um diese zu testen.

Mueller et al. [MAK13] entwickelten mit *GuideMe* eine Android-Anwendung um Anleitungen zu Hausgeräten in AR anzuzeigen. Dabei wird ein Hausgerät via Vuforia und Markern erkannt und die passende Anleitung heruntergeladen und angezeigt. Eine Studie mit Anleitungen in Papierform zeigte, dass Nutzer die AR-Variante dem Papier vorzogen.

3.8 Fazit

Zusammenfassend zeigt sich, dass AR-IoT-Mischanwendungen Einsatz in vielen Bereichen finden. Oft wird für die Anbindung von IoT- zu AR-Geräten eine Middleware, wie ThingSpeak oder die MBP in dieser Arbeit, verwendet. Eine Arbeit verwendete ähnlich zur MBP, MQTT zwischen den IoT-Geräten und der Middleware und REST zwischen der Middleware und den AR-Geräten [BFAF20]. Das IoT besteht speziell bei Prototypen oft, wie auch in dieser Arbeit, aus Raspberry Pis. Für die AR-Geräte sind sowohl Handheld Displays als auch HMDs beliebt. Projektorbasierte Lösungen sind wenig vertreten. Unter den HMDs scheint die Microsoft HoloLens am verbreitetsten zu sein. Die AR-Elemente werden fast immer in Unity implementiert, was laut Karadeniz et al. [KAKE19] an der besseren Effizienz von Unity, im Vergleich zu z. B. Unreal, liegt. Einige Male wurde auch der Einsatz des MRTK mit der HoloLens erwähnt. Die meisten Arbeiten, die Tracking verwenden, setzten dies visionsbasiert um. Marker Tracking und markerloses Tracking sind hier jedoch gleich beliebt. Für beides scheint der Standard Vuforia zu sein, vereinzelt wird das Tracking auch selbst implementiert z. B. via OpenCV. Bei Marker Tracking werden meist QR-Codes getrackt. So deckt sich der Ansatz dieser Arbeit mit dem State of the Art auf diesem Gebiet.

4 Smart-Home-Prototyp

Im Folgenden wird der Aufbau des prototypischen Smart Homes, welches in dieser Arbeit aufgebaut wurde, beschrieben. Auf das dem zugrunde liegende Smart-Home-Szenario wird hier anfangs näher eingegangen. Anschließend wird der Aufbau der verwendeten Hardware beschrieben, gefolgt von der Einrichtung dieser via der MBP. Final werden Hürden, die während des Erstellens dieses Prototyps auftraten und überwunden wurden, behandelt.

4.1 Smart-Home-Szene

Die hier betrachtete Smart-Home-Szene bezieht sich auf eine 2-Zimmer-Wohnung mit Schlaf- und Wohnzimmer. Hier befindet sich ein Raspberry Pi im Schlafzimmer und ein weiterer im Wohnzimmer. In beiden Räumen wird so mithilfe von Licht- und Temperatursensoren die Raumtemperatur und Helligkeit überwacht. Außerdem soll anhand eines Schallsensors erkannt werden, ob sich eine Person im jeweiligen Raum aufhält. RGB-LEDs, welche mit den Pis verbunden sind, stellen dabei Ambientebeleuchtung in beiden Räumen dar. Ein Bosch XDK110[1] überwacht zusätzlich die Temperatur am Küchenfenster. Das Magentometer des XDKs gibt dabei Informationen darüber, ob das Fenster geöffnet ist. Ein Xiaomi MiFlora-Pflanzensensor[2] befindet sich an einer Topfpflanze auf dem Balkon und überwacht hier die Feuchtigkeit, Temperatur, Helligkeit und den Nährstoffgehalt der Erde.

4.2 Hardware-Aufbau

Die Hardware des Smart-Home-Prototyps umfasst einen Raspberry Pi 4 Model B, zwei Raspberry Pi 3 Model B, ein Bosch XDK110, einen Xiaomi MiFlora-Pflanzensensor und eine FRITZ!Box 7490. Um den Hardware-Aufbau für Tests portabel zu halten, wurde eine FRITZ!Box 7490 als Router für das „Heimnetzwerk“ verwendet. Der Router erhält über einen Ethernet-Anschluss Internetzugang und baut damit ein separates Heimnetzwerk auf. Diese Herangehensweise hat dabei den Vorteil, dass über den Router statische IP-Adressen an die IoT-Geräte vergeben werden können, die persistent bleiben, egal an welchem Ort der Router betrieben wird. Außerdem muss so kein separates Notebook oder Smartphone als Accesspoint konfiguriert und durchgehend betrieben werden. Sowohl die statischen IP-Adressen als auch die Nutzeroberfläche der FRITZ!Box vereinfachen so die Konfiguration und Wartung des Smart Homes. Abbildung 4.2 zeigt hierbei den Aufbau der Geräte im Netzwerk.

[1] Bosch XDK110: IoT-Plattform mit acht verschiedenen Sensoren und WiFi-Kommunikation [DG]

[2] Xiaomi MiFlora: Sensorplattform zur Überwachung der Helligkeit, Temperatur, Wasser- und Nährstoffversorgung einer Pflanze [Xia]

Abbildung 4.1: Aufbau des Smart-Homes bestehend aus zwei Raspberry Pis [Fou](a), zwei Temperatorsensoren [Lin16c](b), zwei Geräuschsensoren [Lin16b](c), zwei Lichtsensoren [Lin16a](d), ambienter Beleuchtung [Ind](e), einem Bosch XDK [Bos](f) und einem Pflanzensensor [Xia]. Architekturmodell aus SweetHome3D[Puy]

Der Raspberry Pi 4, im Folgenden *MBP-Pi* genannt, wird dabei als Middleware verwendet, da auf ihm die MBP installiert ist. Hierzu wurde auf einer microSD-Karte Ubuntu 20.04.2 LTS installiert. Gleichzeitig wurden die benötigten Services auf diesem System als Auto-Start konfiguriert, was die MBP nach dem Boot direkt startet. So muss der MBP-Pi lediglich mit Strom versorgt werden und die MBP ist nach kurzer Zeit unter einer gleichbleibenden IP-Adresse erreichbar. Da bei längerem Ausführen der MBP der Raspberry Pi 4 warm wird, wurde ein Case mit Lüfter und ein Kühlkörper auf dessen Prozessor angebracht.

Auf den beiden Raspberry Pi 3, im Folgenden *IoT-Pi-1* und *IoT-Pi-2* genannt, wurde Raspberry Pi OS Lite in Version 10 Buster installiert. Dies ist eine 32-Bit-Linux-Distribution ohne Desktop-Umgebung basierend auf Debian 10 Buster. Beide IoT-Pis wurden mit einem LK-RB-Shield[3] ausgestattet. Dieses Shield wird auf die *General Purpose Input/Output (GPIO)*[4] Pins des Raspberry Pi gesteckt und bietet so Steckverbinder für serielle Kommunikation (UART) und I2C[5] als

[3]LK-RB-Shield: https://www.linkerkit.de/images/7/77/LK-RB-Shield_Anleitung.pdf
[4]GPIO: Anschluss dessen Zweck nicht vorbestimmt ist, sondern in Software definiert wird
[5]I2C: Serieller Bus zur Kommunikation zwischen integrierten Schaltungen

auch digitales GPIO und analoge Eingänge. Für die analogen Eingänge wird ein MCP3008 8-Kanal Analog-Digital-Wandler verwendet. In dieses Shield wurden jeweils drei Sensoren und ein Aktuator eingesteckt. Bei den Sensoren handelte es sich um den LK-Temp-Sensor[6] auf Anschluss *Analog A0*, den LK-Light-Sensor[7] auf *Analog A2* und den LK-Geräusche-Sensor[8] auf *Analog A4*. Der LK-Temp-Sensor verwendet dabei zum Messen der Temperatur einen analogen TMP36 Temperatursensor. Der LK-Light-Sensor verwendet einen lichtabhängigen Widerstand und einen LM358 Operationsverstärker für die Helligkeitsmessung. Um die Schallstärke zu messen, besteht der Schallsensor (LK-Geräusche-Sensor) hauptsächlich aus einem Mikrofon und dem LM386-Audioverstärker. Als Aktuator wurde jeweils ein WS2812B-basierter[9] LED-Strip mit sieben LEDs auf GPIO-12 verbunden. WS2812B bezeichnet hier integrierte Schaltungen bei denen RGB-LEDs und Treiber Chips in einem Paket vereint sind. Diese LEDs sind hintereinanderschaltbar und einzeln adressierbar. Somit kann in einem LED-Strip jede LED einzeln angesteuert werden. Gleichzeitig wird nur eine Datenleitung verwendet. Die Verkabelung dieser Sensoren und Aktuatoren mit dem Shield ist auf dem Diagramm 4.3 zu sehen.

Der IoT-Pi-2 wird zusätzlich als Gateway für das Bosch XDK und den MiFlora-Pflanzensensor verwendet, da beide nicht direkt mit der MBP kommunizieren können. Das Bosch XDK wird per Wi-Fi mit dem Router verbunden und kommuniziert so mit einem Skript auf dem Pi, welches wiederum mit der MBP kommuniziert. Der MiFlora Sensor ist via Bluetooth an den Pi verbunden und kommuniziert so mit der MBP.

Für das Management der Geräte wurde aufgrund der Portabilität ein Netbook mit Lubuntu Linux verwendet, womit per SSH auf die Pis zugegriffen werden konnte. Mit dem Netbook wurden auch periodisch Abbilder der microSD-Karten erstellt und gespeichert, um die Konfiguration der Geräte zu versionieren.

4.3 Installation der MBP

Die Installation der MBP ist auf verschiedene Wege möglich. Via Installer-Skripten für verschiedene Plattformen, in Docker oder indem zuerst der Mosquitto MQTT Broker, der MongoDB Server und Tomcat installiert werden und die MBP anschließend auf Tomcat deployt wird. Informationen über alle Installationsmöglichkeiten finden sich auf dem MBP-GitHub-Repository[10]. Wie zu Beginn dieses Kapitels bereits erwähnt, wurde die MBP für diese Arbeit auf einem Raspberry Pi mit Ubuntu 20.04.2 LTS installiert. Dabei wurde die Installer-Skript-Methode für MBP Version 1.0 angewandt. Dieses Shell-Skript installiert mit Java, Maven, Mosquitto, dem MongoDB Server und Tomcat zuerst die Abhängigkeiten. Nach der jeweiligen Installation werden Mosquitto, MongoDB und Tomcat gestartet und der Maven-Build der MBP gestartet. Ist dieser erfolgreich beendet, wird die resultierende *WAR*-Datei zu Tomcat deployt. Nach kurzer Zeit ist die MBP auf *http://[IP-DES-*

[6] LK-Temp-Sensor: https://www.linkerkit.de/index.php?title=LK-Temp
[7] LK-Light-Sensor: https://www.linkerkit.de/index.php?title=LK-Light-Sen
[8] LK-Geräusche-Sensor: https://www.linkerkit.de/index.php?title=LK-GeräuscheSen
[9] WS2812B Datenblatt: https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf
[10] MBP Installation: https://github.com/IPVS-AS/MBP/wiki/Installation

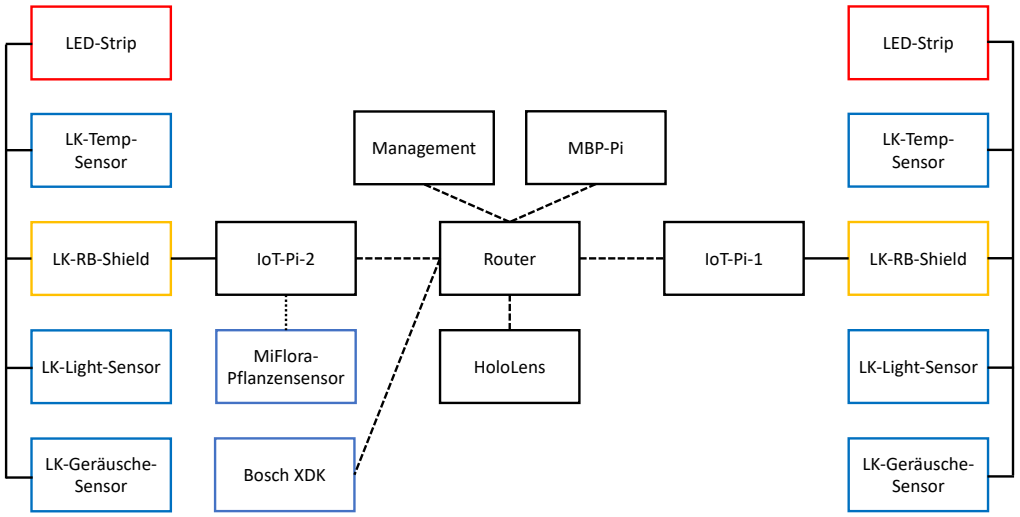


Abbildung 4.2: Aufbau des Smart-Home-Prototyps. Blau: Sensoren, Rot: Aktuatoren, Gelb: LK-Shield, Schwarz: sonstige Hardware, Linie: Kabelverbindung, Gestrichelt: Wi-Fi-Verbindung, Gepunktet: Bluetooth-Verbindung

GERÄTS]:8080/MBP erreichbar. Im Falle des aktuellen Install-Skripts werden die Services von Mosquitto, MongoDB und Tomcat auf „enabled“ gesetzt, was die MBP nach dem Boot automatisch starten lässt. Zusätzlich existieren auch Skripte für das Aktualisieren und Deinstallieren der MBP.

4.4 Anlegen der Geräte

Nach dem Einloggen in die MBP ist der erste Schritt zum Erstellen einer IoT-Umgebung das Registrieren der IoT-Geräte. Im Folgenden wird dieser Prozess für die beiden, in dieser Arbeit verwendeten Raspberry Pis, erläutert. Die grundlegende Vorgehensweise ist jedoch bei jedem Gerät dieselbe. Da im nächsten Schritt SSH verwendet wird, muss dies bei den Raspberry Pis zuvor aktiviert werden. Dies geschieht, indem eine Datei namens „ssh“ in das Stammverzeichnis der Boot-Partition der SD-Karte angelegt wird. Dabei sind die Standard-Zugangsdaten „pi“ als Nutzername und „raspberry“ als Passwort. Zur Sicherheit wurden diese Zugangsdaten hier geändert. Außerdem müssen die Pis mit dem Router per Wi-Fi kommunizieren können. Hierzu wird in der Boot-Partition eine weitere Datei angelegt, mit Namen „wpa_supplicant.conf“. Wie im Folgenden zu sehen, wird hier die SSID und das Passwort des Netzwerks eingetragen.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=DE
network={
        ssid="FRITZ!Box IoT"
        psk="PASSWORT"
        key_mgmt=WPA-PSK
}
```

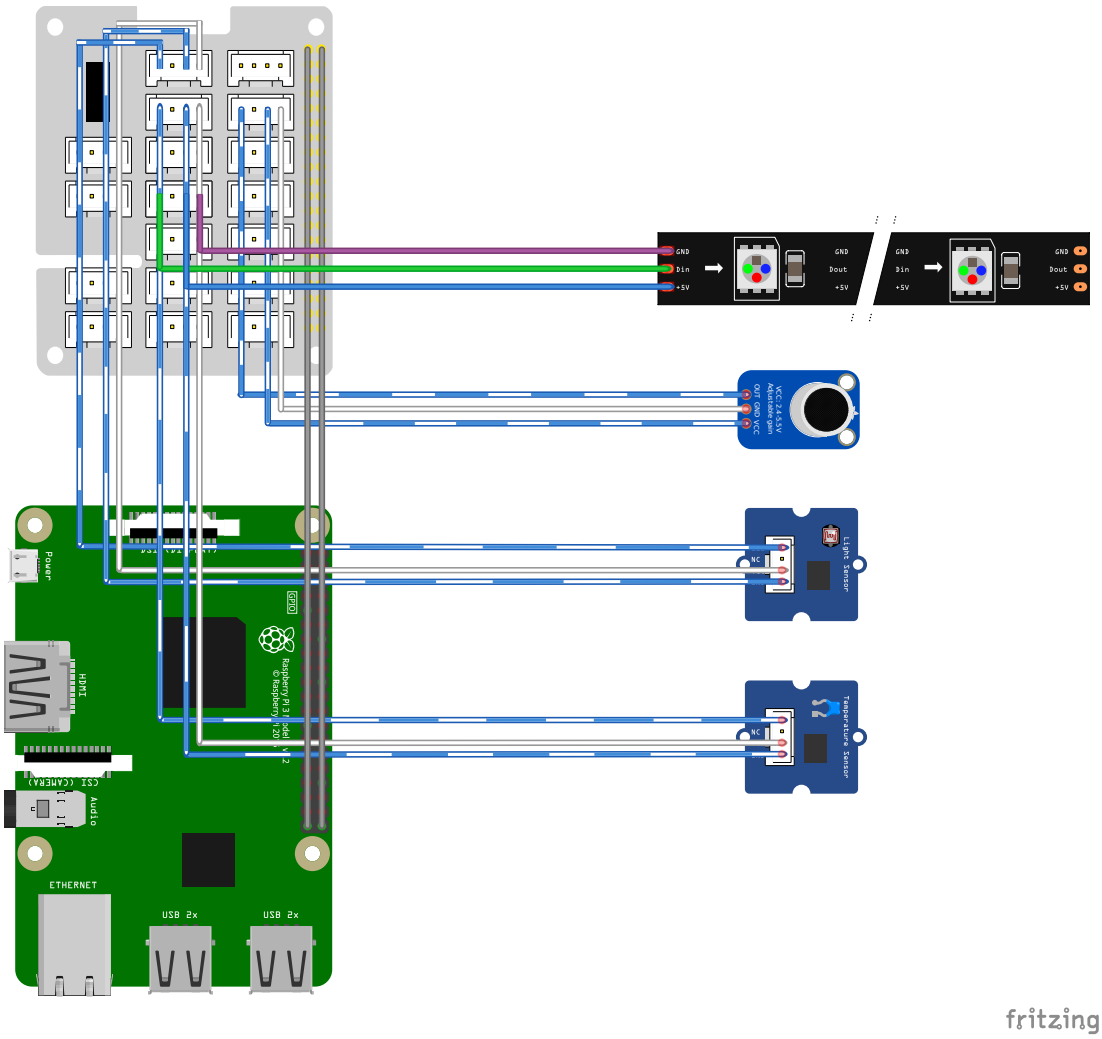


Abbildung 4.3: Verkabelung der Sensoren und Aktuatoren mit dem LK-Shield und Raspberry Pi. Abbildung basierend auf Fritzing-Diagramm [Fri]

Wird die microSD-Karte nun in den Pi eingesetzt und dieser mit Strom versorgt, so verbindet sich Raspberry Pi OS beim Boot mit dem Wi-Fi-Netzwerk und aktiviert SSH mit den zuvor erwähnten Standard-Zugangsdaten.

Anschließend kann auf der MBP-Webseite, im Untermenü „Devices“ des Menüs „IoT Hardware“, ein neues Gerät hinzugefügt werden. Nach dem Eintragen der benötigten Daten des Geräts, wie des Namens, der Geräteart, der IP-Adresse und der SSH-Zugangsdaten kann das Gerät registriert werden. Zu sehen ist dieser Vorgang in Abbildung 4.4. Die MBP verbindet sich anschließend per SSH an das angelegte Gerät, um den Status zu überprüfen. Ist der Status auf der sich anschließend öffnenden Seite, wie in Abbildung 4.5 grün, kann mit dem nächsten Schritt fortgefahren werden.

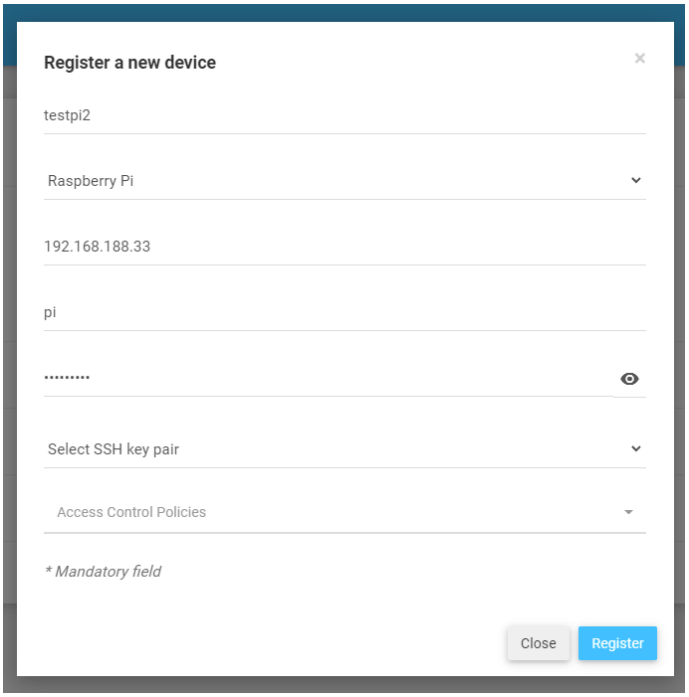


Abbildung 4.4: Dateneingabe zur Verbindung eines neuen Raspberry Pi

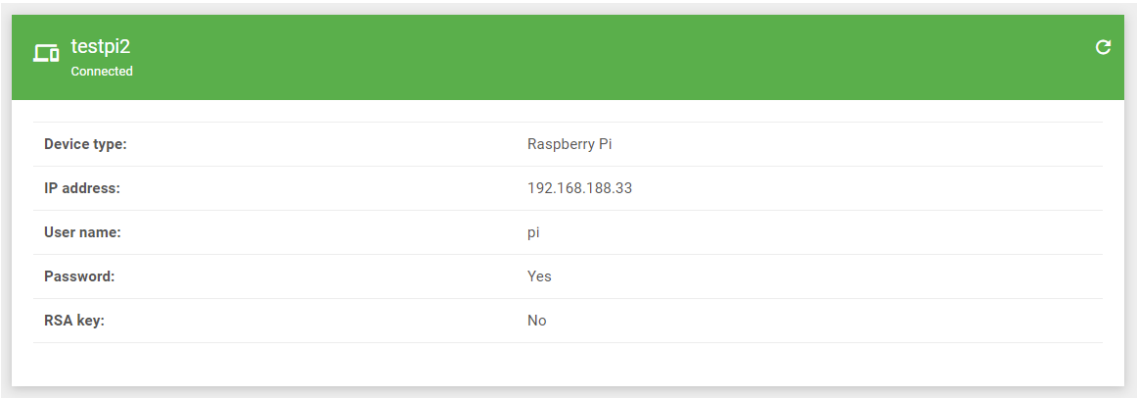


Abbildung 4.5: Status des verbundenen Raspberry Pis auf MBP-Frontend

4.5 Anlegen der Operatoren

Um einen Sensor oder Aktuator mit der MBP zu verbinden, muss zuerst ein sogenannter Operator angelegt werden. Die Einrichtung findet hier über das MBP Webinterface, über den Menüpunkt „Operators“ statt. Für das Erstellen eines Operators müssen primär sogenannte Operator-Skripte vorhanden sein. Diese können für Sensoren und Aktuatoren, die bereits von der MBP unterstützt werden, wie den MiFlora-Sensor oder das Bosch XDK, direkt aus dem GitHub Repository geladen oder selbst implementiert werden. Diese Skripte werden dazu verwendet, um Daten von den

Sensoren auszulesen und an die MBP via MQTT zu senden oder um Aktuatoren via MQTT von der MBP aus zu steuern. Zusätzlich existieren auch Skripte zum Überprüfen des Status und Installieren aller Abhängigkeiten.

Abbildung 4.6: Aufbau des Operators für einen LK-Schallsensor

Am Beispiel des Schallsensors, zu sehen in Abbildung 4.6, wird dies im Folgenden näher erklärt. Dieser Operator wurde im Laufe dieser Arbeit implementiert und ist auf GitHub[11] zu finden. Die *README.md* enthält Instruktionen darüber, wie der Operator zu verwenden ist. Hier z. B. an welchen Anschluss der Schallsensor eingesteckt werden muss.

Die *install.sh* ist ein Shell-Skript welches die Abhängigkeiten des Operators beim Deployment aus der MBP auf dem Zielgerät installiert. In diesem Fall Python 3 mit dem Paket-Manager *pip*, um den MQTT-Client *paho-mqtt* zu installieren.

Das *LK-sound_raspberry-pi.py*-Skript ist das Hauptskript, welches Daten des Schallsensors sammelt und per MQTT an die MBP sendet. Hier werden auch Parameter, die in der MBP für den Sensor gesetzt werden können, wie der analoge Kanal des Pis, an dem der Sensor angeschlossen ist, übertragen. Dabei werden auch die *mbp_client.py*- und *mbp_analog_reader.py*-Skripte verwendet. Das *mbp_client.py*-Skript stellt die Verbindung zur MBP via MQTT her und wird benötigt, um Daten zu senden oder zu lesen. Der *mbp_analog_reader.py* liest analoge Werte, wie die des Schallsensors und liefert diese an das Hauptskript zurück.

Die *start.sh* und *stop.sh* sind Shell-Skripte, die das Hauptskript starten und stoppen. Das *running.sh*-Skript liefert zurück, ob der Prozess des Hauptskripts noch ausgeführt wird.

Die *entry-file-name*-Datei enthält den Namen des Hauptskripts, in diesem Fall „LK-sound_raspberry-pi.py“. So weiß das Start-Skript welches Hauptskript ausgeführt werden soll. Zusammen mit dem install.sh-Skript wird so die Möglichkeit geboten, das Hauptskript und somit den Operator in nahezu jeder Programmiersprache zu entwickeln.

Sind die Skripte heruntergeladen oder angelegt, kann der Operator angelegt werden. Zusätzlich zu den Skripten wird hierbei ein Name, mögliche Parameter, eine Beschreibung und die Einheit, in der gemessen wird, falls es sich um einen Sensor handelt, angegeben. Nach dem Anlegen ist dieser Operator inklusive der Skripte auf dem Gerät, auf dem die MBP ausgeführt wird, hier also der Raspberry Pi 4, abgespeichert und kann anschließend auf Geräte deployt werden. Hierbei kann ein Operator auf beliebig viele Geräte deployt werden.

[11] Operator für Schallsensor: https://github.com/IPVS-AS/MBP/tree/master/resources/operators/extraction/LK-sound_raspberry-pi

Operatoren existieren nicht nur für Sensoren und Aktuatoren sondern auch für das Monitoring der Geräte. Zu finden sind diese im Menüpunkt „Operators“ unter „Monitoring Operators“. Ein Monitoring-Operator kann so z. B. die Temperatur oder Auslastung des Prozessors überwachen.

4.6 Anlegen von Sensoren und Aktuatoren

Um einen Sensor oder Aktuator anzulegen, muss auf der MBP-Oberfläche unter „IoT Hardware“ das Menü „Sensors“ oder „Actuators“ geöffnet werden. Im Folgenden wird der Vorgang wieder anhand des Schallsensors erläutert. Beim Anlegen wird nach Angabe des Namens, der Typ des Operators, der Operator und das gewünschte Gerät, auf das der Operator installiert werden soll, ausgewählt. Als Typ des Sensors würde hier „Sound“ gewählt werden, als Operator der zuvor angelegte und als Gerät einer der beiden Raspberry Pi 3. Wird auf der sich anschließend öffnenden Übersichtsseite auf „install Operator“ geklickt, deployt die MBP den Operator, inklusive Skripts, auf den Pi. Gleichzeitig wird auch das „install“-Skript ausgeführt und die Abhängigkeiten installiert. Bis zu diesem Zeitpunkt wurden noch keine Daten von der MBP auf den Pi geladen, es wurde lediglich die SSH-Verbindung geprüft. Nun wird ein neuer Ordner namens „scripts“ im „Home“-Verzeichnis des angegebenen SSH-Nutzers, hier „pi“ angelegt. In diesem Ordner befindet sich nun ein weiterer, der mit dem Präfix „mbp“ gefolgt von der ID des Sensors benannt ist. Dieser Ordner enthält die vorher beschriebenen Skripte des Operators. Zusätzlich enthält er auch eine *connections.txt*-Datei. Diese enthält die IP-Adresse des MBP-Pis für die MQTT-Kommunikation.

Die Vorgehensweise ist bei den Aktuatoren an sich dieselbe, außer dass mit ihnen nicht direkt interagiert werden kann. Die Sensoren zeigen, auf der zuvor beschrieben Übersichtsseite die derzeitigen und historischen Sensorwerte. Der Aktuator kann über diese Seite jedoch nicht gesteuert werden. Die Aktuatoren sollen eigentlich über die Rule-Engine gesteuert zu werden. Hier könnte eine Regel angelegt werden, bei der ein Aktuator angesteuert wird, sobald die Werte des Schallsensors einen bestimmten Schwellenwert überschreiten. Um einen Aktuator direkt zu steuern, existiert die Möglichkeit, eine sogenannte „Rule-Action“ anzulegen und diese zu testen. Diese befinden sich im „Rules“-Menü unter „Actions“. Beim Erstellen muss als Typ „Actuator action“ angegeben und der gewünschte Aktuator ausgewählt werden. Nach dem Anlegen einer solchen Rule-Action wird die Möglichkeit die Action zu testen, durch „Test Action“ gegeben. Dies steuert den Aktuator an und kann so z. B. eine LED aufleuchten lassen.

4.7 Hürden während der Einrichtung der Geräte

Durch die Hardwarenähe dieses Projekts traten im Laufe der Arbeit einige Hürden auf. Im Folgenden werden diese beschrieben und die jeweiligen Lösungen aufgezeigt.

Das erste Problem trat bereits bei der Installation der MBP auf. Die in *Abschnitt 4.3* bereits erwähnten Skripte zur Installation, Aktualisierung und Deinstallation der MBP waren nicht mehr mit aktuelleren Versionen von Ubuntu, wie das hier verwendete 20.04.2 LTS, kompatibel. Die Skripte wurden ursprünglich auf Ubuntu 16.04 LTS ausgelegt. So wurden die Skripte dementsprechend angepasst und im Zuge dessen neue Versionen von MongoDB und Tomcat hinzugefügt. Gleichzeitig wurde der MongoDB-Service auf „enabled“ gesetzt, was den Autostart der MBP ermöglichte, da

die anderen benötigten Services bisher bereits auf „enabled“ gesetzt wurden. Nach diesen in Pull Request 577[12] und 623[13] resultierenden Änderung konnte die MBP ohne weiteres auf Ubuntu 20.04.2 LTS installiert werden.

Direkt nach der Installation der MBP und Inbetriebnahme der Raspberry Pis trat ein weiteres Problem auf. Die Operatoren für die LinkerKit-Sensoren konnten nicht ausgeführt werden. Hier war das Problem, dass der Analog-Digital-Wandler auf dem LK-RB-Shield das Serial Peripheral Interface (SPI)[14] zur Kommunikation verwendet. SPI ist unter Raspbian bzw. Raspberry Pi OS jedoch standardmäßig deaktiviert. Nachdem SPI in der *config.txt* in der Boot-Partition aktiviert wurde, war das Problem gelöst. Die nächste Problematik zeigte sich bereits beim Konfigurieren der übrigen Sensoren an dem Shield. Das LK-RB-Shield wird in zwei Ausführungen angeboten. Eine basierend auf dem MCP3004 4-Kanal Analog-Digital-Wandler, eine weitere auf Basis des MCP3008 8-Kanal Analog-Digital-Wandlers. Die in diesem Prototyp eingesetzten Sensoren benötigen zwar nur einen analogen Kanal, jedoch blockiert der Stecker der Sensoren Pins für zwei Kanäle. So können bei dem 4-Kanal Shield nur zwei Sensoren verbunden werden. Der Prototyp verwendete zwar die 8-Kanal Version, der Code auf Seiten der MBP war jedoch nur für die 4-Kanal Version entwickelt. So wurde beim Verwenden weiterer Kanäle immer eine *-1* als Resultat auf Anfragen nach dem Sensorwert zurückgegeben. Behoben wurde dies durch Pull Request 608[15]. Dabei wurde im Code lediglich die Abfrage auf die Kanal-Adresse abgeändert. Anschließend konnten alle Sensoren verbunden werden.

Auch nach dem alle Sensoren verbunden waren, zeigten sich weitere Probleme beim Auslesen der Werte. So lieferten nach dem Aufsetzen der Pis und Verbinden aller Sensoren und Aktuatoren mit dem LK-RB-Shield zwei Sensoren, spezifisch ein LK-Temp-Sensor und ein LK-Light-Sensor, stark fluktuierende Werte, außerhalb des plausiblen Wertebereichs, für die Situation in der getestet wurde. Hier kam durch Mischen der einzelnen Komponenten, wie Pis, Shields, Sensoren und Kabel, bereits der Verdacht auf, dass die Sensoren defekt sein könnten. Um diesen Verdacht zu überprüfen, wurden die Sensoren mit einem Arduino-Mikrocontroller getestet. Dieser ist einfacher aufgebaut als der Raspberry Pi, was beim Testen die Stellen, an denen Probleme auftreten könnten, deutlich reduziert. Auch hier wurden dieselben fluktuierenden Werte gemessen. Der finale Test wurde anschließend mit einem Multimeter[16] durchgeführt. Hier wurde für den LK-Light-Sensor vor und nach dem Operationsverstärker gemessen. Vor dem Operationsverstärker verhält sich der hier gemessene Widerstand wie erwartet, die Spannung nach dem Operationsverstärker jedoch nicht. Bei dem LK-Temp-Sensor wurden am Sensor selbst schon kein nachvollziehbarer Spannungswert gemessen. Die Schlussfolgerung war, dass beide Sensoren defekt waren. Nach dem Austausch durch neue Sensoren traten bei gleicher Konfiguration keine Auffälligkeiten mehr auf. Zur Sicherheit wurde die Prozedur mit dem Arduino und dem Multimeter, anschließend mit allen Sensoren und Aktuatoren, durchgeführt, zeigte jedoch keine weiteren Anomalien.

Als Aktuator wurde, wie bereits erwähnt, pro Raspberry Pi jeweils ein WS2812B-basierter RGB-LED-Strip verwendet. Der Operator für diese Art Aktuator verwendete Python2, alle anderen Skripte bisher jedoch Python3. Um das Skript zu aktualisieren, musste eine andere Bibliothek für die

[12] Pull Request 577: https://github.com/IPVS-AS/MBP/pull/577
[13] Pull Request 623: https://github.com/IPVS-AS/MBP/pull/623
[14] SPI: Standard für einen synchronen seriellen Bus für Master-Slave-Verbindungen
[15] Pull Request 608: https://github.com/IPVS-AS/MBP/pull/608
[16] Multimeter: Messgerät für verschiedene elektronische Messgrößen

Kommunikation mit den WS2812B LEDs verwendet werden. Zuvor wurde die Neopixel-Bibliothek von Adafruit verwendet. So wurde auf die mittlerweile für Python3 von Adafruit empfohlene rpi-ws281x-Bibliothek aktualisiert [Ada14]. Dementsprechend konnte das Install-Skript angepasst werden, um die neue Bibliothek zu installieren und Python3 zu verwenden. Außerdem wurde das Start-Skript angepasst, um den Operator als Root auszuführen, da die rpi-ws281x-Bibliothek Root-Rechte für den Zugriff auf Pulsweitenmodulation (PWM)[17] benötigt. Diese Anpassungen resulutierten in Pull Request 589[18]

Der Operator konnte nun problemlos ausgeführt werden. Die LEDs flackerten jedoch noch. Das noch verbleibende Problem war, dass analoges Audio des Raspberry Pis auf PWM basiert. Die LEDs werden jedoch auch via PWM angesteuert, was einen Konflikt darstellte. Durch das Deaktivieren des Audio-Kernel-Moduls und Erzwingen von HDMI-Audio[19] konnte PWM für die rpi-ws281x-Bibliothek verwendet werden. Somit leuchteten die LEDs anschließend wie gewünscht.

Das Verbinden des MiFlora-Pflanzensensors bereitete ebenfalls Komplikationen. Hier konnte keine Bluetooth-Verbindung zwischen beiden Raspberry Pis und dem Sensor hergestellt werden. Mithilfe des Kommandozeilenprogramms *bluetoothctl* (Teil des bluez-utils-Pakets) wurde hier erfolglos versucht, manuell eine Verbindung herzustellen. Auch ein Test mit einem anderen Raspberry Pi (Pi Zero W) lieferte dasselbe Resultat. Nach einem Batteriewechsel und Update der Firmware über die Android-App des Herstellers konnte via bluetoothctl eine Verbindung hergestellt werden. Anschließend funktionierte die Kommunikation auch über das Operator-Skript.

Auch die Verbindung des Bosch XDK gestaltete sich kompliziert. Da sich das XDK nicht direkt mit der MBP verbindet, sondern per MQTT mit einem der Raspberry Pis kommuniziert, muss ein Projekt mit dem dafür nötigen Code auf das XDK geladen werden. Ein Bosch-Account muss hierfür erstellt werden, um die auf Eclipse basierende IDE namens *XDK-Workbench*[20] herunterzuladen. Anschließend muss das Projekt aus dem MBP-GitHub-Repository in die XDK-Workbench importieren werden. Daraufhin wird im Code die *AppController.h*-Datei angepasst. Unter anderem müssen hier die Zugangsdaten des Wi-Fi-Netzwerks, die IP-Adresse des MQTT Brokers und der Port unter, dem dieser erreichbar ist, angegeben werden. Eine genauere Übersicht der Parameter findet sich in der dazu angelegten Readme-Datei aus Pull Request 609[21]. Nach dem Kompilieren des Projekts kann dieses per USB-Kabel auf das XDK übertragen werden. Anschließend konnte der Operator auf dem Raspberry Pi mit dem XDK via MQTT kommunizieren. Die Auswahl des verwendeten Sensors wird bei diesem Operator über die bereits erwähnten Parameter aus *Abschnitt 4.5* aufseiten der MBP gelöst.

[17] PWM: Modulation zwischen zwei Werten (hier: an und aus)
[18] Pull Request 589: https://github.com/IPVS-AS/MBP/pull/589
[19] PWM-Audio deaktivieren: https://pypi.org/project/rpi-ws281x/
[20] XDK-Workbench: https://developer.bosch.com/products-and-services/sdks/xdk/downloads
[21] Pull Request 609: https://github.com/IPVS-AS/MBP/pull/609

5 Augmented-Reality-Prototyp

Zur Entwicklung des AR-Prototyps für die Steuerung und Visualisierung von Geräten, Sensoren und Aktuatoren (im folgenden IoT-Komponenten genannt) via MBP wurde, wie in *Abschnitt 3.8* bereits angedeutet, die Microsoft HoloLens 2 gewählt. Aufgrund der dort beschriebenen Vielzahl an erfolgreichen Projekten wurde die HoloLens mit der Unity-Engine für die Entwicklung des Prototyps verwendet. Im Folgenden wird deshalb nach einer Einführung zur HoloLens, die Einrichtung von sowohl der HoloLens als auch Unity beschrieben. Außerdem wird dargelegt, wie das Tracking der IoT-Komponenten umgesetzt wurde. Anschließend folgt eine Beschreibung der Funktionen, durch die mit der Anwendung interagiert werden kann. Zum besseren Verständnis des Unity-Projekts wird auch ein Überblick über die technische Umsetzung der Funktionen gegeben. Final wird auf Probleme während der Entwicklung des Prototyps und die gewählten Lösungen eingegangen.

5.1 HoloLens-Hardware

Folgende Übersicht über die Hardware der HoloLens 2 und einen kurzen Vergleich mit der ersten Generation basiert auf der Hardware-Dokumentation von Microsoft [Mic21a; Mic21b]. Die HoloLens 2 ist wie ihr Vorgänger ein Optical See-Through Mixed-Reality-Headset von Microsoft.

Abbildung 5.1: Seitenansicht der HoloLens 2 [Mic]

Abbildung 5.1 zeigt die Seitenansicht der HoloLens 2. Als Anzeige werden zwei 3:2 Displays mit 2K Auflösung verwendet. Im Gegensatz zur Intel x86 basierten HoloLens 1 verwendet die HoloLens 2 einen Snapdragon 850 ARM64 Prozessor. Als Betriebssystem wird *Windows Holographic*, basierend auf Windows 10, eingesetzt. Die zweite Generation verdoppelt auch den Arbeitsspeicher von 2 GB auf 4 GB LPDDR4x. Auch die unterstützte Bluetooth-Version wurde verbessert, von Bluetooth 4.1 LE auf Bluetooth 5.0. Zusätzlich steht Wi-Fi 802.11ac mit zweifachem MIMO zur Verfügung. Beide Versionen der HoloLens werden mit 64 GB Flash-Speicher ausgeliefert, die HoloLens 2 setzt jedoch auf den performanteren UFS 2.1-Standard. Die Akkulaufzeit beträgt weiterhin zwei bis drei Stunden. Eine Neuerung der HoloLens 2 ist das, auch in dieser Arbeit verwendete, Hand-Tracking. Hierbei werden mithilfe der vier Kopf-Tracking-Kameras, der IMU, einer Tiefenkamera und einer 8 Megapixel Hauptkamera die einzelnen Finger beider Hände voll beweglich getrackt. Außerdem stehen Eye-Tracking und Sprachsteuerung als Eingabe zur Verfügung. Die Sensoren und Kameras können auch verwendet werden, um einen Raum abzutasten und so ein Mesh des Raumes zu erstellen, welches als Referenz für die Platzierung von Objekten in AR verwendet werden kann.

5.2 HoloLens-Setup

Bevor mit dem Entwickeln des Prototyps begonnen werden konnte, musste die HoloLens eingerichtet werden. Im Folgenden wird darüber ein kurzer Überblick gegeben.

Nach dem ersten Einschalten der HoloLens 2 muss diese zuerst auf die Augen des Nutzers kalibriert werden, unter anderem, für die spätere Verwendung des Eye-Trackings. Dabei tauchen verschiedene Objekte an unterschiedlichen Stellen im Gesichtsfeld des Nutzers auf, die dieser mit den Augen verfolgen muss. Der nächste Schritt war das Verbinden der HoloLens mit einem Wi-Fi-Netzwerk. Zur Eingabe des WLAN-Passworts wird eine holographische virtuelle Tastatur eingeblendet, die auf Touch-Eingaben reagiert. Dabei muss der Nutzer mit seinem Finger die Projektion an der Stelle der gewünschten Taste oder generell an der Stelle von UI-Elementen berühren. Daraufhin muss ein Microsoft-Account angegeben werden. Hier wurde für die HoloLens ein neuer Account angelegt. Das Eingeben der Daten geschieht auch hier wieder über die holographische virtuelle Tastatur. Anschließend folgen Fragen zum Lizenzvertrag, Produktverbesserung und Standortdaten, wie sie auch bei regulären Windows 10 Installationen vorkommen. Abschließend wird nach dem Einrichten einer Pin oder von Windows Hello mit Iris-Scanner verlangt. Hier wurde aufgrund der einfacheren Bedienung das Iris-Scanning eingerichtet. Dazu wird wieder ein Prozess, wie beim initialen Kalibrieren des Eye-Trackers durchlaufen und am Ende noch eine Pin zum Entsperren ohne Iris-Scanner verlangt.

Nach dem Einrichten taucht ein Menü auf, das dem Windows 10 Startmenü ähnelt, worin eine App mit weiteren Tipps zur Bedienung der HoloLens empfohlen wird. Das Gerät ist an dieser Stelle jedoch fertig eingerichtet. Um später mit der HoloLens zu kommunizieren, den Video-Stream aufzuzeichnen, Status-Informationen abzufragen, Code zu deployen, etc. muss noch das Geräteportal aktiviert werden. Dieses befindet sich in den Einstellungen im Startmenü unter „Update und Sicherheit“, „Für Entwickler“. Hier muss der „Entwicklermodus“ aktiviert werden. Danach können die „Gerätesuche“ und das „Geräteportal“ aktiviert werden. Bei verbundenem Internet steht hier auch die IP-Adresse der HoloLens. Wird nun an einem Computer *https://[IP-DER-HOLOLENS]*

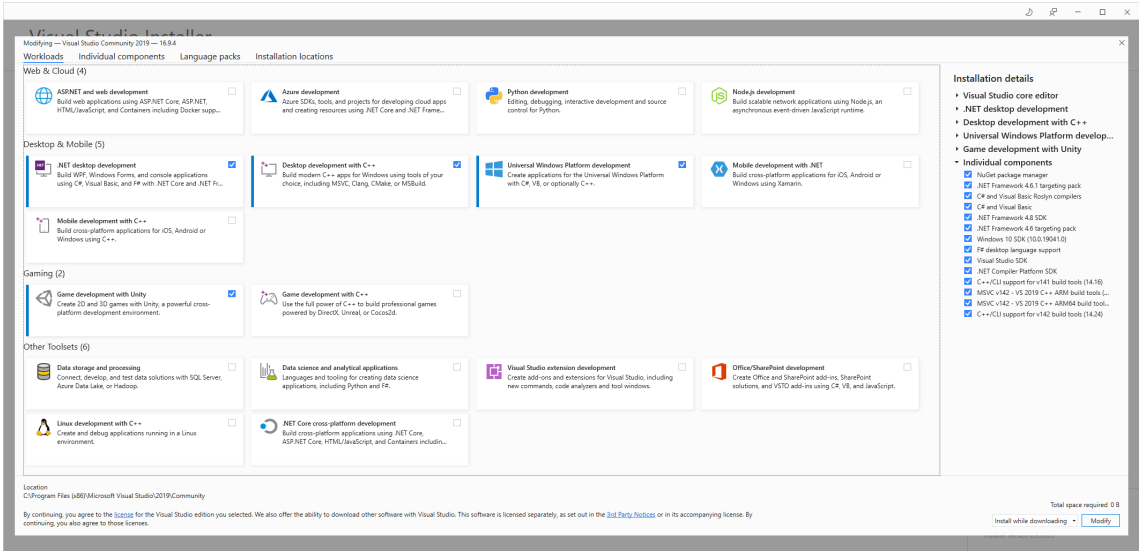


Abbildung 5.2: Modulauswahl bei der Visual-Studio-Installation

im Browser aufgerufen, leitet der Browser auf das Geräteportal der HoloLens weiter. Beim ersten Aufruf dieser Seite muss ein Benutzername und Passwort angelegt werden. Danach ist die HoloLens bereit für das Deployment von Anwendungen.

5.3 Unity-Entwickler-Setup

Wie bereits erwähnt, wurde für diese Arbeit die Microsoft HoloLens 2 mit Unity verwendet. Zur Entwicklung für die HoloLens wird ein Computer mit Windows 10 Pro unter Version *21H1* und aktiviertem Entwicklermodus, eine HoloLens oder ein Emulator, Visual Studio und Unity mit verschiedenen Plugins und Paketen benötigt. Durch die Komplexität der Einrichtung all dieser Komponenten wird im Folgenden ein Überblick über die nötigen Schritte gegeben, um für die HoloLens entwickeln zu können. Der erste Schritt ist es den Entwicklermodus in Windows 10 zu aktivieren. Dieser befindet sich in den Windows-Einstellungen unter „Update und Sicherheit“, „Für Entwickler“. Als nächstes wird die Microsoft-Visual-Studio-Entwicklungsumgebung installiert[1]. Diese wird benötigt, um den C#-Code, und somit die Logik für die Anwendung, zu entwickeln. Im Zuge dieser Arbeit wurde die Version *2019 16.9.4* in der Community-Edition verwendet. Bei der Installation müssen die folgenden Module, auch *Workloads* genannt, ausgewählt werden: „Spieleentwicklung mit Unity“ und „Entwicklung für die universelle Windows-Plattform“. Abbildung 5.2 zeigt eine Übersicht der Module im Installer. Dabei wird auch das benötigte Windows SDK installiert, hier in Version *10.0.19041.685*. Das Windows SDK wird zum Entwickeln benötigt, da das Betriebssystem der HoloLens auf Windows 10 basiert. Sobald Visual Studio installiert ist und sich in einen Microsoft-Account eingeloggt wurde, kann Visual Studio geschlossen und Unity installiert werden.

[1] Visual Studio Download: `https://visualstudio.microsoft.com/de/downloads/`

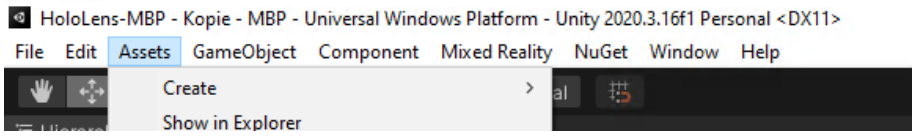


Abbildung 5.3: Reiter in Unity nach dem Installieren von NuGet

Die Installation von Unity wird über den „Unity Hub“[2] durchgeführt. Nachdem der Unity Hub installiert ist, muss sich auch hier eingeloggt werden, um eine Lizenz hinzuzufügen. Es existieren hierbei kostenlose und kostenpflichtige Lizenzen mit unterschiedlichem Funktionsumfang[3]. Für diese Arbeit wurde die kostenfreie Lizenz für Studierende verwendet. Im Unity Hub unter „Installs“ kann anschließend „Unity 2020.3.16f1 LTS“ bzw. die aktuellste 2020 LTS-Version ausgewählt werden. Während der Installation, im Schritt „add modules to your install“ muss hier auch das Modul für den „Universal Windows Platform Build Support“ hinzugefügt werden.

Nach Abschluss der Installation, wird im Unity Hub unter „Projects“ ein neues Unity-Projekt, unter Verwendung des „3D-Templates“, erstellt. Sobald dieses Projekt geöffnet ist, müssen die Mixed-Reality-Toolkit-Bibliotheken installiert werden. Hierfür kann das Microsoft Mixed Reality Feature Tool[4] verwendet werden. Nach dem Öffnen des Feature-Tools muss dort der Pfad zum vorher erstellten Unity-Projekt angeben werden. Nach einem Klick auf „Discover Features“, können unter „Mixed Reality Toolkit“ die „Mixed Reality Toolkit Foundation“ und die „Mixed Reality Toolkit Extensions“ gewählt werden. Dabei werden auch weitere benötigte Komponenten wie die „Mixed Reality Toolkit Standard Assets“ mitinstalliert. Zusätzlich empfiehlt es sich auch die „Mixed Reality Toolkit Examples“ zu installieren, um den Umgang mit den Elementen des MRTK an Beispielszenen besser zu veranschaulichen. Alle MRTK-Pakete wurden im Zuge dieser Arbeit in Version *2.7.0* verwendet. Für die Detektion der QR-Codes wird zusätzlich noch OpenXR[5] benötigt. OpenXR ist ein Open-Source-XR-Standard der Khronos Group [Khra]. Es findet sich im Feature Tool unter „Platform Support“ als „Mixed Reality OpenXR Plugin“ und wurde in Version *1.0.2* installiert.

Die Installation via Feature Tool bereitete unter bestimmten Windows Versionen, im Laufe dieser Arbeit, Probleme. Die Bibliotheken können jedoch auch einzelnen aus dem GitHub-Repository[6] geladen und in Unity importiert werden, sollte dieser Fall eintreten. Um diese manuell zu importieren, muss über den „Assets“-Reiter unter „Import Package“, „Custom“ ausgewählt und anschließend die entsprechende *.unitypackage*-Datei ausgewählt werden. In Abbildung 5.3 sind die Reiter aus Unity zu sehen. In dieser Arbeit wurde das MRTK in Version *2.7.0* via Mixed Reality Feature Tool in Version *1.0.2104.0-Beta* installiert. Abbildung 5.4 zeigt das Feature-Tool-Fenster mit den ausgewählten Paketen.

Nach der Installation via Feature Tool muss zurück in das Unity-Fenster gewechselt werden. Nach dem manuellen Import müsste das Unity-Fenster bereits im Fokus sein. Dort sollte sich ein Konfigurationsfenster öffnen, bei dem das „Unity OpenXR Plugin“, wie in Abbildung 5.5a

[2] Unity Hub Download: https://unity3d.com/de/get-unity/download
[3] Übersicht über Unity-Lizenzen: https://store.unity.com/de/compare-plans
[4] Download Mixed Reality Feature Tool: https://www.microsoft.com/en-us/download/details.aspx?id=102778
[5] XR ist ein Sammelbegriff für AR, VR und MR
[6] MRTK GitHub: https://github.com/microsoft/MixedRealityToolkit-Unity

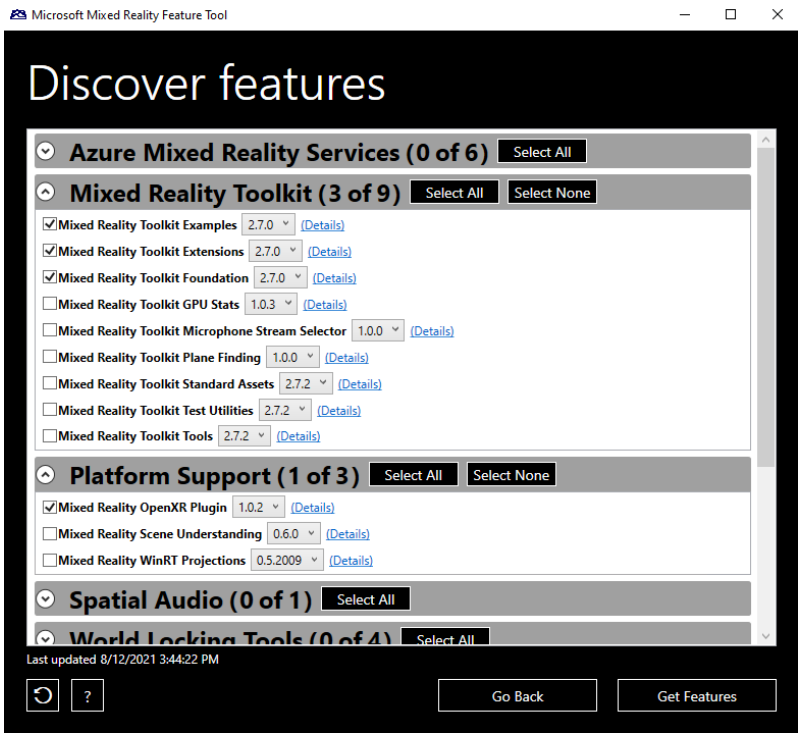


Abbildung 5.4: Übersicht über die benötigten Pakete des MRTK Feature Tools

zu sehen, aktiviert werden muss. Sollte sich ein weiteres Fenster zu den Einstellungen der „XR Pipeline“ öffnen, kann hier wie in Abbildung 5.5b auf „Skip This Step“ geklickt werden, da diese Anpassungen, nach dem Installieren der übrigen Pakete in einem Schritt durchgeführt werden. Auf einem weiteren Fenster, über das die Projekteinstellungen konfiguriert werden können, genügt es, durch Klick auf „Apply“ die Standardeinstellungen zu übernehmen. Während dieser Installation und Konfiguration ist es möglich, dass Unity mehrfach neu gestartet wird. So auch nach dem Anwenden der Standard-MRTK-Einstellungen wie in Abbildung 5.5c zu sehen. Nach einem dieser Neustarts sollte sich das in Abbildung 5.5d zu sehende Konfigurationsfenster des TextMeshPro-Pakets öffnen. Hier kann dieses durch Klick auf „Import TMP Essentials“ installiert werden. TextMeshPro wird benötigt, um geglätteten Text in der AR-UI darzustellen. Bei manueller Installation können in Unity im „Window“-Reiter unter „TextMeshPro“ die „TMP Essential Resources“ importiert werden. Nun sollte abgesehen von Informationen zu den MRTK-Beispielen der MRTK-Installationsassistent beendet sein und das MRTK somit installiert. Nach dem MRTK-Konfigurationsassistent öffnet sich ein weiteres Fenster, welches die Option gibt, das Backend des neuen Unity-Interaktionssystems zu aktivieren. Dieses sollte aktiviert werden.

Für den nächsten Schritt muss eine für Unity adaptierte Version des Paket-Managers „NuGet“ installiert werden. Verwendet wurde hier NuGet in Version *3.0.2*. Dazu muss die *.unitypackage*-Datei aus GitHub[7] heruntergeladen und über den „Assets“-Reiter unter „import package“, „custom“ ausgewählt und importiert werden. Nach dem Import sollte sich in Unity ein weiterer Reiter namens „NuGet“ befinden. Zu sehen ist das in Abbildung 5.3. NuGet wird hier verwendet, um

[7] NuGetForUnity: https://github.com/GlitchEnzo/NuGetForUnity/releases/tag/v3.0.2

(a) Konfiguration des Open XR Plugins

(b) Konfiguration der XR Pipeline

(c) Standard-MRTK-Einstellungen

(d) Konfiguration von TextMeshPro

Abbildung 5.5: Screenshots des MRTK-Konfigurationsassistenten

wie in Abbildung 5.6 zu sehen, die Microsoft Mixed Reality QR-Bibliothek[8] für das Tracking von QR-Codes zu installieren. Hierzu muss unter dem „NuGet“-Reiter in Unity „Manage NuGet Package“ ausgewählt und „Microsoft.MixedReality.QR“ in die Suche eingegeben werden. Die Bibliothek wurde hier in Version *0.5.3011* installiert.

Zuletzt wird für die QR-Bibliothek noch das „Windows XR Plugin“ installiert. Hierzu muss in Unity unter dem „Window“-Reiter der „Package Manager“ geöffnet, dort nach „Windows XR Plugin“ gesucht und dieses installiert werden. Im Zuge dieser Arbeit wurde die Version *4.5.0* installiert. Nun sollten alle benötigten Pakete vorhanden sein. Abbildung 5.7 zeigt hierzu eine Übersicht der bisher installierten Pakete.

[8] QR-Bibliothek: https://www.nuget.org/packages/Microsoft.MixedReality.QR/

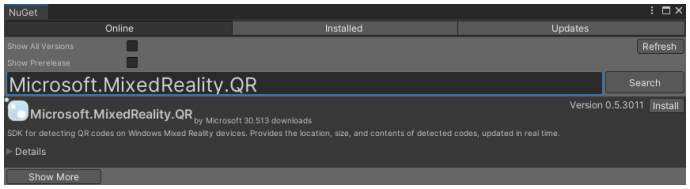


Abbildung 5.6: Screenshot Microsoft Mixed Reality QR-Bibliothek im NuGet-Paketmanager

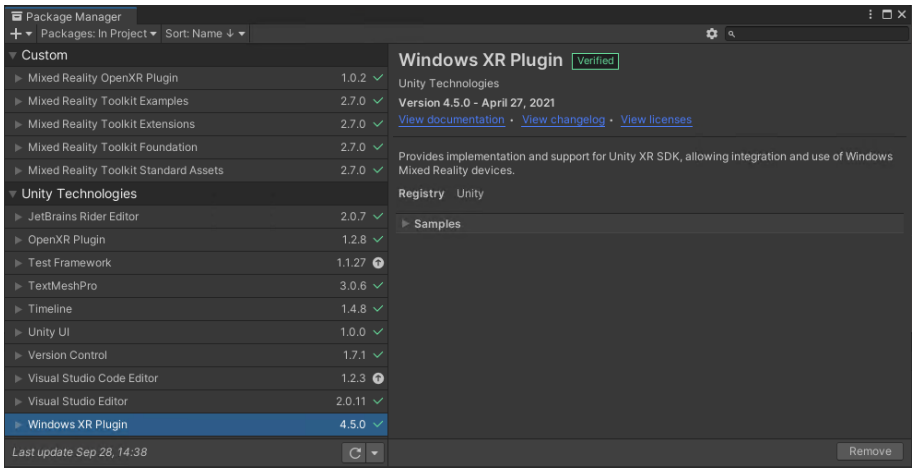


Abbildung 5.7: Screenshot des Unity-Paketmanagers mit allen bisher installierten Paketen

Anschließend muss die Unity-Szene der Anwendung erstellt oder eine Demo-Szene ausgewählt werden. Wird eine neue Szene erstellt, muss diese für das MRTK vorbereitet werden. Hierzu kann unter dem Reiter „Mixed Reality“, „Toolkit“ „Add to Scene and Configure...“ ausgewählt werden. Dies fügt verschiedene GameObjects, welche die Kamera und Interaktion via Mixed Reality Headsets umsetzten, hinzu und entfernt die standardmäßig in der Szene befindliche, zweidimensionale Kamera.

Sollen auch QR-Codes gelesen werden, kann die Referenz-Implementierung der QR-Code-Funktionalität aus der Beispielszene aus dem GitHub-Repository[9] der QR-Code-Bibliothek als Basis importiert werden. Dabei muss der Inhalt des „Script“-Ordners in das neue Projekt übernommen werden, da sich hier die C#-Skripte zum Lesen von QR-Codes befinden. Aus dem „Prefabs“-Ordner muss mindestens das „QRCodesManager“-Prefab übernommen werden. Testweise können hier auch alle Prefabs und Skripte übernommen oder das komplette Projekt in Unity geöffnet werden, um das QR-Code-Lesen zu testen.

[9] QR-Code Beispielszene: `https://github.com/yl-msft/QRTracking`

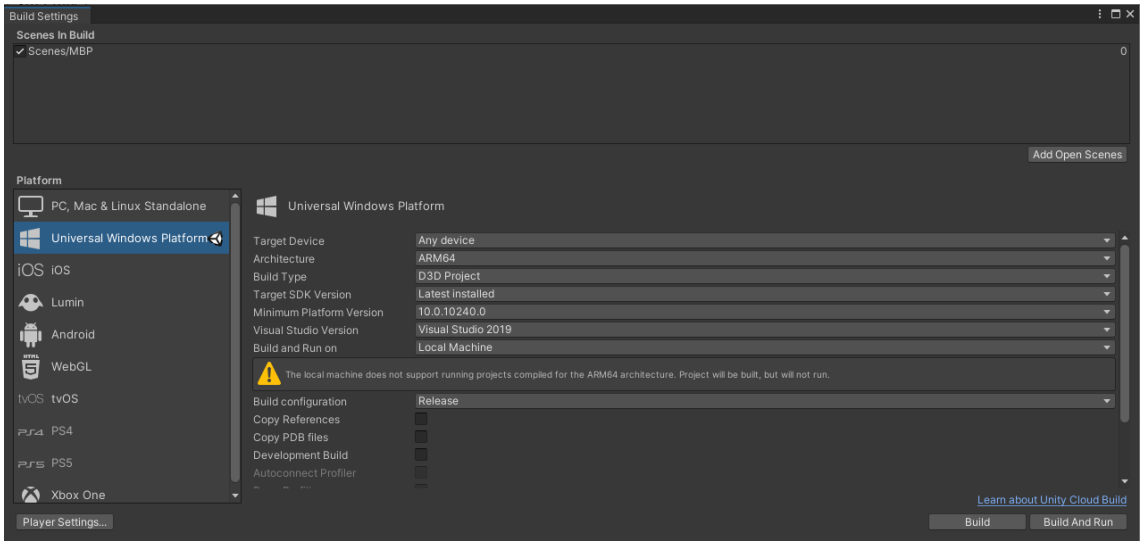


Abbildung 5.8: Fertig konfigurierte Unity Build Settings

Nun folgt folgt die Anpassung des Projekts für XR und speziell für die HoloLens. Unter dem „File“-Reiter muss dazu das „Build Settings“-Menü geöffnet werden. Zu sehen ist dies in Abbildung 5.8. In diesem Menü muss durch „Add Open Scenes“ die gewünschte offene Szene zum Build hinzugefügt werden. Anschließend wird die Plattform auf „Universal Windows Platform“ und die Architektur auf „ARM64“ gestellt. Hier kann auch die verwendete Visual-Studio-Version gesetzt werden, falls mehrere Versionen installiert sind.

Außerdem muss mit Klick auf „Player Settings“ das „Player Settings“-Fenster geöffnet werden. Unter „XR Plug-in Management“ kann nun „OpenXR“ und die „Microsoft HoloLens feature group“ aktiviert werden. Die dabei angezeigten Warnungen können ignoriert werden. Speziell um die Hand-Tracking-Funktionen der HoloLens 2 zu verwenden, muss im „OpenXR“-Menü unter „Interaction Profiles“ das „Microsoft Hand Interaction Profile“ hinzugefügt werden. Zu sehen ist dies in Abbildung 5.9.

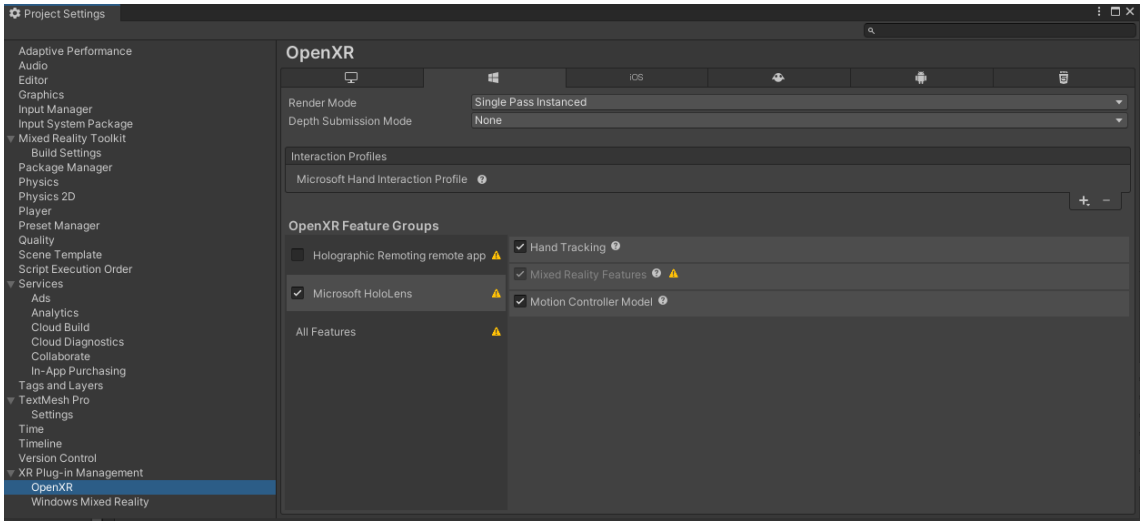


Abbildung 5.9: Player Settings mit OpenXR-Einstellungen

Da das Lesen der QR-Codes Zugriff auf die Kamera der HoloLens benötigt, muss dieser aktiviert werden. Hierzu muss unter „Player“ das „Publishing Settings“-Menü aufgeklappt werden und unter „Capabilities“ der Menüpunkt „WebCam“ wie in Abbildung 5.10 aktiviert werden.

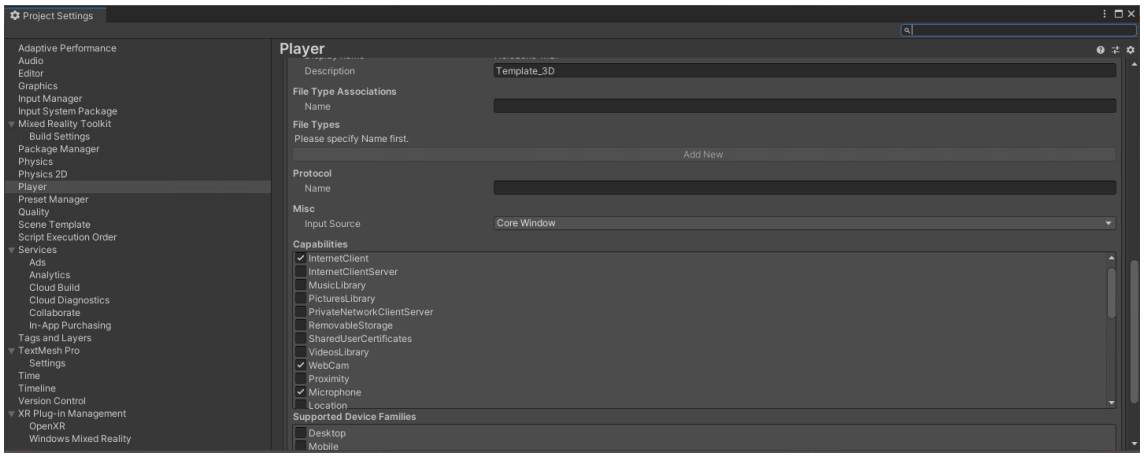


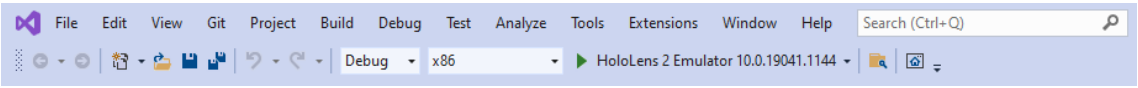
Abbildung 5.10: Player Settings mit aktivierter WebCam Capability

Nun kann der Build in Unity im „Build Settings“-Fenster gestartet werden. Dabei wird auch ein Zielverzeichnis für die gebaute Anwendung ausgewählt. Dieser Build baut keine fertige Windows-Anwendung, sondern ein Visual-Studio-Projekt, mit dem Visual Studio eine Windows-Anwendung erstellen kann. Nachdem der Build erfolgreich durchgelaufen ist, kann in diesem Ordner eine *.sln*-Datei vorgefunden werden. Dies ist eine sogenannte „Visual Studio Solution“, sprich eine Datei die ein oder mehrere Projekte inklusive Konfiguration enthalten kann. Diese kann anschließend in Visual Studio geöffnet werden.

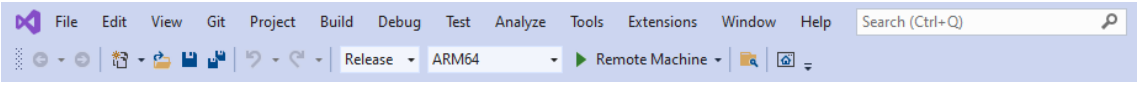
Die Anwendung kann entweder auf der HoloLens direkt ausgeführt oder mithilfe des HoloLens-Emulators[10] getestet werden.

Da der Emulator, spezifisch der HoloLens-2-Emulator, in dieser Arbeit oft Verwendung fand, wird im Folgenden an den relevanten Stellen auch erklärt, wie der entsprechende Vorgang mit dem Emulator umzusetzen wäre.

Nachdem die *Solution* in Visual Studio geöffnet wurde kann es passieren, dass das MBP-Projekt nicht als „Startprojekt“ festgelegt ist. Hier kann im Projektmappen-Explorer das Projekt durch einen Rechtsklick als Startprojekt festgelegt werden. Anschließend wird dieses Projekt automatisch beim Start der Solution ausgewählt. Daraufhin muss die Plattform für den Emulator auf „x86“ und für die HoloLens 2 auf „ARM64“ gestellt werden. Das Zielgerät muss entweder auf den HoloLens-Emulator oder „Remote Machine“ für die reale HoloLens gestellt werden. Um auf die HoloLens zu deployen, muss unter den „Debug Properties“ der „Machine Name“ auf die IP-Adresse der HoloLens oder deren Gerätename gesetzt werden. Dazu muss sich diese im selben Netzwerk befinden, als der Computer, auf dem Visual Studio ausgeführt wird und wie in *Abschnitt 5.2* erklärt, konfiguriert sein. Einstellungen für beide Deployment-Arten sind auf den Abbildungen 5.11a, 5.11b zu sehen.



(a) Einstellungen für das Test-Deployment auf den HoloLens-2-Emulator



(b) Einstellungen für das Release-Deployment auf die HoloLens 2

Abbildung 5.11: Screenshots der Visual-Studio-Konfiguration für das Deployment der Anwendung

Nun kann das „Start“-Symbol ausgewählt werden, um den finalen Build zu starten. Beim ersten Ausführen auf der HoloLens wird zusätzlich noch nach einer Pin gefragt. Diese kann in den Einstellungen der HoloLens unter „Update und Sicherheit“, „Für Entwickler“, nach einem Klick auf den „Koppeln“-Button angezeigt werden. Die Anwendung wird nach dem Kompiliervorgang nun auf der HoloLens gestartet. Im Falle des Emulators startet sich dieser gegen Ende des Builds selbst. Zur Steuerung des Emulators kann entweder die Maus und Tastatur verwendet werden oder ein Xbox Controller. Für Tests in AR, speziell auf der HoloLens, sollte der Build auch von „Debug“ auf „Release“ gestellt werden, da so die Performance deutlich erhöht wird.

[10] HoloLens-Emulator: https://docs.microsoft.com/de-de/windows/mixed-reality/develop/platform-capabilities-and-apis/using-the-hololens-emulator

5.4 Tracking

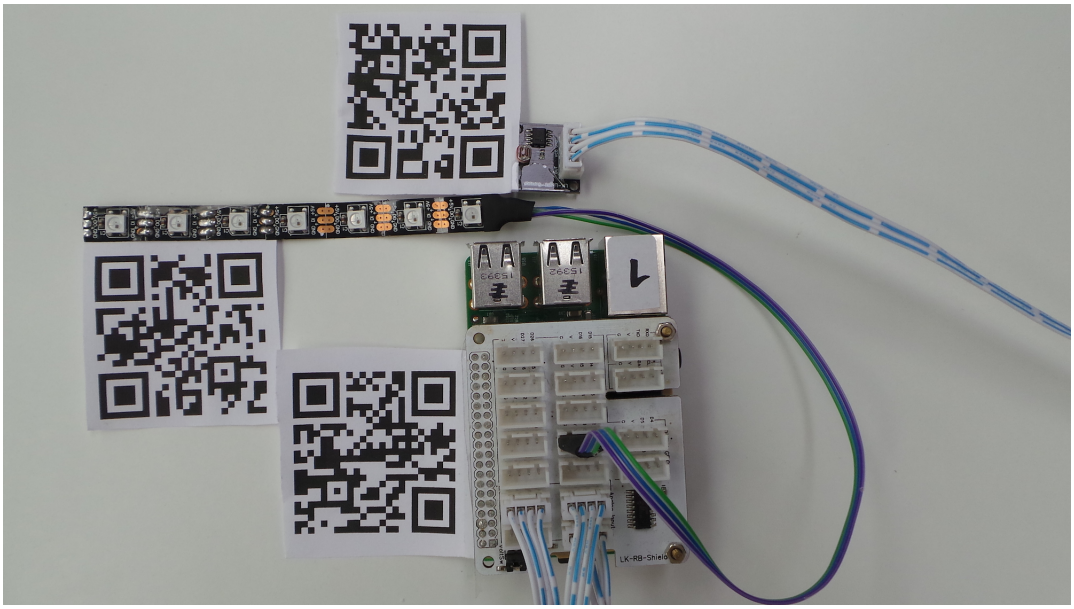


Abbildung 5.12: QR-Codes für einen Raspberry Pi („60802e175889cd2f6828fc4b, DEV“), Lichtsensor („608d52fb0c013d5bcfea9f2b, SEN“) und RGB-LEDs („60c0e4012f60c17b7eae1f9b, ACT“)

Zum Tracking der IoT-Komponenten wurde nach Evaluation der verschiedenen Möglichkeiten aus *Kapitel 3* Marker Tracking mit QR-Codes verwendet. Der Hauptgrund hierfür war das einfachere Unterscheiden von identisch aussehenden Geräten, im Gegensatz zu markerlosem Tracking. Da das prototypische Smart Home in dieser Arbeit aus mehreren Raspberry Pis, Sensoren und Aktuatoren mit gleichem Aussehen besteht, war dies besonders wichtig. Es wurde auch früh die Entscheidung für QR-Codes als Marker getroffen, da so direkt die Informationen, die verwendet werden, um mit dem dazugehörigen IoT-Komponenten zu kommunizieren, aus dem QR-Code geladen werden können. Hierfür wurde ein QR-Code in Version 2 verwendet, da die Daten zu umfangreich für Version 1 waren. Die Versionen definieren bei QR-Codes die Anzahl schwarz-weißer Kästchen (auch Module genannt) [WAV]. QR-Codes der Version 2 beinhalten dabei 25 Kästchen. Die Fehlerkorrektur wurde auch auf das niedrigste Level gesetzt, um Platz im QR-Code zu sparen und so eine möglichst niedrige Version und große Module zu erreichen. Pro QR-Code wird die aus der MBP stammende ID gefolgt von „SEN“, „ACT“ oder „DEV“ für Sensor, Aktuator und Gerät kodiert. Zu sehen ist dies in Abbildung 5.12.

Um das Erstellen der QR-Codes so einfach wie möglich zu gestalten, wurde der *MBP QR-Codes Creator* entwickelt. Dieses Python-Skript tätigt jeweils eine GET Request um alle Aktuatoren, Sensoren und Geräte, die mit der MBP verbunden sind, zu erhalten. Aus jeder so erhaltenen IoT-Komponente werden anschließend der Name und die ID extrahiert und zum erstellen eines QR-Codes mithilfe der *qrcode*-Bibliothek[11] verwendet. Dabei wird die ID und Art der IoT-Komponente,

[11] Python QR-Code-Bibliothek (qrcode): https://pypi.org/project/qrcode/

wie oben beschrieben, in die Daten des QR-Codes geschrieben und der Name und die ID bilden den Dateinamen. Nach dem Ausführen des Skripts befinden sich alle QR-Codes als *PNG*-Datei zum ausdrucken im QR-Codes-Creator-Verzeichnis.

Im Gegensatz zu Marker Tracking mit Vuforia wird für QR-Code-Tracking keine Internetverbindung oder ein vorher erstelltes offline Modell benötigt. Außerdem ist die aktuelle Preisgestaltung von Vuforia, im Gegensatz zum lokalen Tracken von QR-Codes, unter anderem an die Anzahl zu trackender Objekte oder Marker geknüpft und je nach Lizenz auf verschiedene Weisen eingeschränkt[12]. Im Vergleich zu proprietären Markern lassen sich QR-Codes außerdem auf unterschiedlichste Arten generieren, von Python-Bibliotheken, wie hier im Prototyp verwendet, bis hin zu Online-Werkzeugen und Smartphone-Apps. Zum Erfassen der QR-Codes via HoloLens wurde, wie bereits erwähnt, die Microsofts QR-Bibliothek für Mixed-Reality-Geräte eingesetzt. Dies ist die von Microsoft empfohlene Art, QR-Codes mit der HoloLens zu tracken [Mic21c]. Es wurde dabei die minimal unterstützte QR-Code-Größe mit 5 cm Seitenlänge verwendet. Größere QR-Codes könnten aus weiterer Entfernung erkannt werden, wären für den Nutzer jedoch potentiell störend, da sie somit größer als die zu trackenden Geräte wären. Da die Position der Geräte jedoch nur initial oder beim Verschieben dieser getrackt werden muss, wurde sich dagegen entschieden, die QR-Codes zu vergrößern.

5.5 Beschreibung der Interaktion

Der folgende Abschnitt setzt eine eingerichtete MBP mit IoT-Komponenten und HoloLens voraus. Nach dem Start der AR-Anwendung wird der Zustand aller verbundenen IoT-Komponenten aus der MBP abgefragt. Einige der dafür verwendeten Requests benötigen bis zu mehreren Minuten. Trotzdem kann hier bereits mit dem Einrichten der Anwendung begonnen werden. Alle Menüs und Interaktionsmöglichkeiten sind so aufgebaut, dass sie auch ohne eine Verbindung zur MBP ausgeführt werden können und die benötigten Daten beim Wiederherstellen der Verbindung nachladen. Die Anwendung cacht dazu alle von der MBP geladenen Daten und versucht diese periodisch zu aktualisieren. So zeigen die Menüs immer den Stand der Daten zur Zeit der letzten erfolgreichen Aktualisierung an.

Möchte ein Nutzer mit einer, mit der MBP verbundenen, IoT-Komponente kommunizieren, muss er diese zuerst mit der AR-Anwendung tracken. Dies setzt voraus, dass für die zu trackende IoT-Komponente ein QR-Code im richtigen Format existiert. Das Verwenden des bereits erwähnten MBP-QR-Codes-Creator-Skripts garantiert dies. Sind die korrekten QR-Codes an der IoT-Komponente angebracht, muss sich der Nutzer mit aufgesetzter HoloLens und laufender AR-Anwendung in Richtung QR-Code bewegen, bis dieser erkannt wird. Bei den, während der Entwicklung verwendeten, QR-Codes mit 5 cm Kantenlänge, musste sich auf bis zu 10 cm an den QR-Code angenähert werden. Wird ein gültiger QR-Code erkannt, erscheint auf dessen Position ein farbiger Pfeil auf einem weißen Quader, wie in Abbildung 5.13 zu sehen. Dieser Pfeil ist rot gefärbt für einen Aktuator, blau für einen Sensor und gelb für ein Gerät, wie einen Raspberry Pi. Die Position und Rotation des Quaders wird bis zum nächsten Neustart der HoloLens gespeichert und in der Umgebung verankert.

[12] Vuforia Lizenzen: https://www.ptc.com/de/products/vuforia/vuforia-engine/pricing

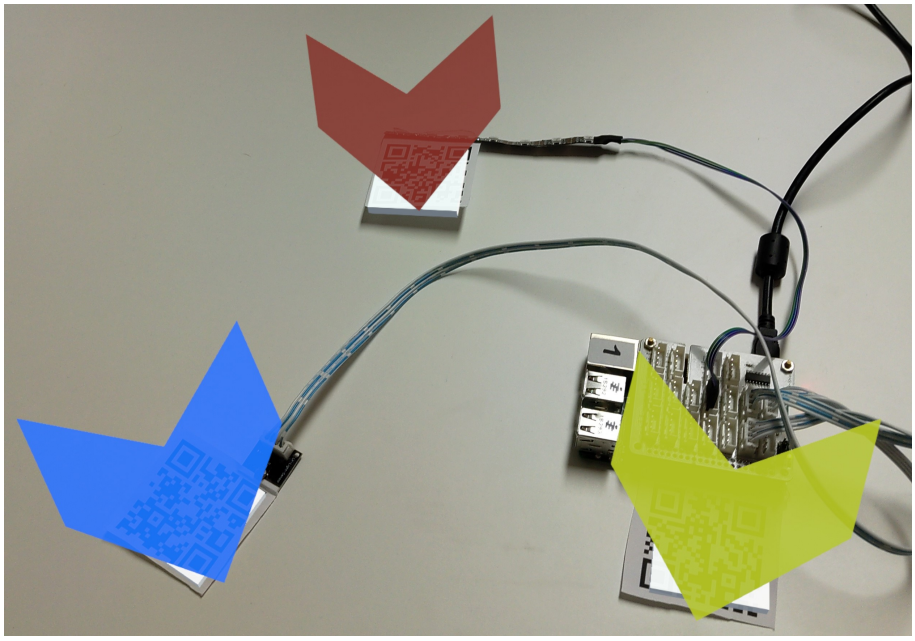


Abbildung 5.13: QR-Codes für einen Raspberry Pi, Lichtsensor und RGB-LEDs mithilfe der HoloLens getrackt

Hat der Nutzer, wie in Abbildung 5.13, alle gewünschten Geräte getrackt, kann er via der sogenannten „Far Interaction“ der HoloLens 2 mit den Geräten interagieren. Dabei entspringt dem ausgestreckten Zeigefinger ein Strahl, der als Zeiger verwendet wird. Schneidet dieser ein dafür konfiguriertes GameObject in der Szene, wie die Pfeile, so kann durch eine Pinch-Geste mit derselben Hand oder indem der Sprachbefehl „Select“ ausgesprochen wird, die Auswahl bestätigt werden. Abbildung 5.14 zeigt die Far Interaction auf einen Sensor.

Um diese Interaktion zu vereinfachen, verfügen die Pfeile der getrackten Komponenten über sogenanntes *Billboarding*. Billboarding ist eine Funktion des MRTKs, die GameObjects auf angegebenen Achsen in Richtung der Kamera, sprich des Nutzers, rotiert. Wendet der Nutzer diese Far Interaction auf einen der getrackten Komponenten an, öffnet sich in Griffreichweite ein Menü, welches sich je nach IoT-Komponente unterscheidet. Geräte, wie der Raspberry Pi, zeigen alle verbundenen Monitoring-Operatoren, Sensoren und Aktuatoren in einer Liste, inklusive des Status der jeweiligen Operatoren an. Jeder Menüeintrag zeigt auch das von der MBP geladene Icon der dazugehörigen Komponente an. Zusätzlich ermöglichen diese Menüs durch das Auswählen eines Eintrages zusätzliche Informationen über die ausgewählte Komponente zu erhalten oder sie zu steuern. Wird ein Aktuator in der Liste ausgewählt, steuert diese Aktion direkt den Aktuator. Wird der Pfeil über dem QR-Code eines Aktuators ausgewählt, öffnet sich ein Menü mit derselben Funktion. Bei Sensoren öffnet sich sowohl aus dem Gerätemenü, als auch über die Pfeil-Interaktion, dasselbe Menü. Dieses zeigt neben dem Namen des Sensors auch den letzten erhaltenen Wert und den Durchschnittswert, jeweils in der von der MBP geladenen Einheit, an. Zusätzlich wird ein Liniendiagramm über die letzten 50 Werte gezeichnet. Zur einfacheren Interaktion verfügen alle Menüs über Billboarding, um sich immer in Richtung des Nutzers zu drehen. Zu sehen sind all diese Menüs in Abbildung 5.15.

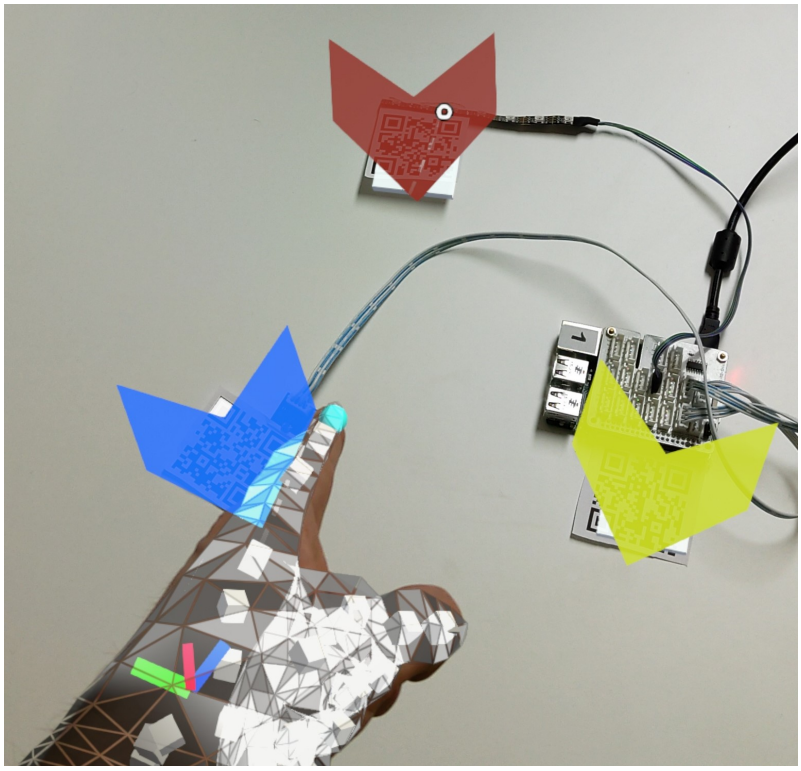


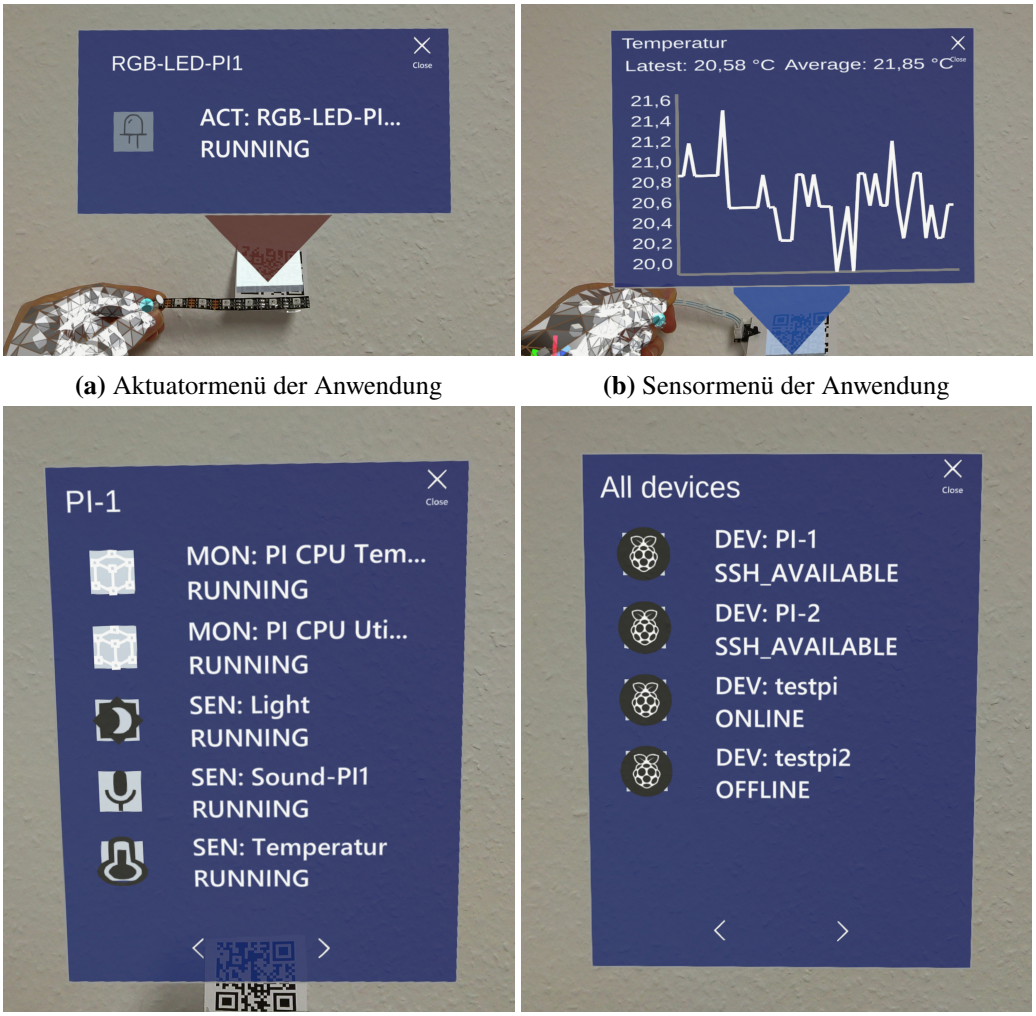
Abbildung 5.14: HoloLens Far Interaction am Beispiel eines Aktuators

Abgesehen von den besprochenen Menüs existiert noch ein sogenanntes Handmenü. Dieses öffnet sich, wenn der Nutzer seine Hand mit gespreizten Fingern und der Handfläche in Richtung HoloLens hält. Dabei erscheint wie in Abbildung 5.16 zu sehen, ein Menü mit drei Unterpunkten an der Innenseite der Hand. Diese können mit der anderen Hand bedient werden. Hier gibt es die Möglichkeit alle Operatoren der MBP zu starten, die keinen Parameter benötigen, um Zeit bei der Interaktion zu sparen, da dies sonst während des Auswählens eines Sensors oder Aktuators geschehen würde. Außerdem kann der Nutzer alle Geräte der MBP anzeigen, was für größere Räumlichkeiten hilfreich sein kann, bei denen die HoloLens die Szenen nicht weit genug in die Entfernung anzeigen kann. Der letzte Menüpunkt schließt das Handmenü.

Um die Anwendung zu schließen bzw. zu minimieren muss die Hand wie bei der Handmenügeste vor die HoloLens gehalten werden und auf das auf die Handwurzel projizierte Windows Symbol, gedrückt werden. Dieses Symbol ist auch in Abbildung 5.16 am unteren Bildrand zu sehen. Beim Antippen öffnet sich das HoloLens-Start-Menü, über das die Anwendung geschlossen oder minimiert und die HoloLens ausgeschaltet werden kann.

5.6 Technische Umsetzung des Prototyps

Im Folgenden wird ein Überblick über den technischen Aufbau der MBP-AR-Anwendung gegeben.



(a) Aktuatormenü der Anwendung

(b) Sensormenü der Anwendung

(c) Gerätemenü der Anwendung

(d) Menü zum Anzeigen aller Geräte

Abbildung 5.15: Screenshots der verschiedenen Menüs des Prototyps

Erbt eine Klasse in Unity von *MonoBehaviour* und ist einem GameObject angehängt, so erhält sie Zugriff auf den Lebenszyklus dieses GameObjects. Wird dieses instanziiert, so wird eine *Awake*-Methode aufgerufen, anschließend eine *Start*-Methode und bei jedem weiteren Frame-Update eine *Update*-Methode. Dieser Lebenszyklus wird beispielsweise von der *MBPConnectionManager*-Klasse verwendet. Der MBPConnectionManager ist dabei ein konstant ausgeführtes Skript, angehängt an ein gleichnamiges GameObject in der Unity-Szene, das Dictionaries mit Daten der MBP anderen Klassen zur Verfügung stellt. Dictionaries sind in C# Collections zur Speicherung von Key-Value-Paaren. Diese Dictionaries werden via REST Requests in der Awake-Methode gefüllt und in der Update-Methode nach Ablauf eines Timers periodisch aktualisiert und machen den MBPConnectionManager so zum Cache der MBP-Daten.

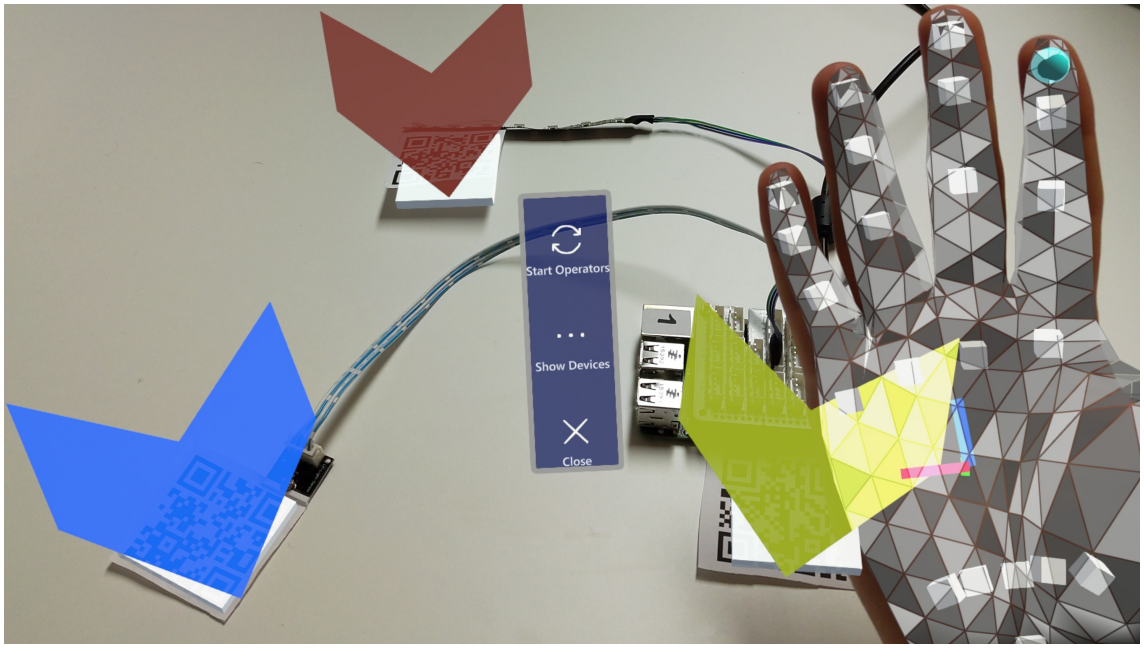


Abbildung 5.16: Handmenü der Anwendung

Zuerst werden hier die Sensoren und Aktuatoren in Dictionaries mit der jeweiligen ID als Schlüssel und dem entsprechenden Datentransferobjekt (DTO)[13] als Wert geladen. Da diese GET Requests jedoch nicht den Status des Sensors oder Aktuators zurückgeben, werden anschließend weitere GET Requests für den Status durchgeführt. Folgendes zeigt eine solche GET Request für den Status der Sensoren:

```
GET /MBP/api/sensors/state HTTP/1.1
Host: 192.168.188.24:8080
X-MBP-Access-Request: requesting-entity-firstname=admin;;requesting-entity-lastname=admin;;req
uesting-entity-username=admin
Authorization: Basic YWRtaW46MTIzNDU=
[...]

HTTP/1.1 200
Content-Type: application/hal+json;charset=UTF-8
[...]

{60c0e90d2f60c17b7eae1f9e: "DEPLOYED", 60e072e19392c6502aabcd72: "RUNNING", 6113838fe25fb12ad4
ccf355: "NOT_READY"}
```

Die Liste aus den dabei erhaltenen Antworten wird den Einträgen der Sensor- und Aktuator-Dictionaries anhand der IDs zugeordnet. Selbiges wird auch für ein Dictionary für *Devices* und eines für *MonitoringOperators* mit deren jeweiligen Zuständen angelegt. Außerdem wird aus den Sensor- und Aktuator-Dictionaries ein sogenanntes *FieldDevices* Dictionary aufgebaut. Dieses verknüpft die Devices mit den jeweils an diese angeschlossenen Sensoren und Aktuatoren. Die Geräte-ID bildet den Schlüssel und eine Liste aus FieldDevices den Wert. MonitoringOperators werden analog

[13] Datentransferobjekt: Repräsentation der Datenstrukturen, die via REST empfangen und gesendet werden

zu den FieldDevices in einem Dictionary aus den Geräte-IDs und Listen von MonitoringOperators gespeichert. In beiden Fällen vereinfacht dies den Zugriff des *DeviceMenu*, welches alle Sensoren, Aktuatoren und MonitoringOperators des ausgewählten Geräts anzeigt. Zusätzlich wird eine Liste der Icons der IoT-Komponenten von der MBP geladen, um diese in den Menüs anzuzeigen.

Die GET Requests zum Füllen der Dictionaries funktionieren mehrstufig. Am Beispiel der GET Request der Sensoren, zu sehen in Abbildung 5.17 wird zuerst eine asynchrone *GetSensors*-Methode aufgerufen. Diese und alle anderen hier aufgerufenen Methoden sind als *async* markiert und geben einen *Task* zurück, der wiederum den Rückgabewert enthält. Async bedeutet in C#, dass das *await* Keyword in der Methode verwendet werden kann und so auf die Ausführung anderer asynchroner Methoden gewartet werden kann. Dabei ist await kein *busy waiting*, sondern setzt die laufende Methode in den Hintergrund, sodass andere Methoden während des Wartens ausgeführt werden können. Die GetSensors-Methode ruft wiederum die generische *Get*-Methode auf und übergibt hierbei den gewünschten API-Endpoint. Get erstellt eine UnityWebRequest[14], konkateniert den API-Endpoint mit dem Endpoint der MBP-API und ruft hiermit die *DoGet*-Methode auf. DoGet setzt die Header für die Authentifizierung, sendet die Request und wartet auf das Ergebnis. Der HTTP Response Body der UnityWebRequest wird hierbei von der DownloadHandler-Klasse verarbeitet und als String-Repräsentation schlussendlich zurückgegeben. Im Falle eines Fehlers wird dieser zurückgegeben und geloggt. Am Ende erhält die GetSensors-Methode einen JSON String der Sensoren und erstellt daraus ein Array aus *SensorDTO*-Objekten.

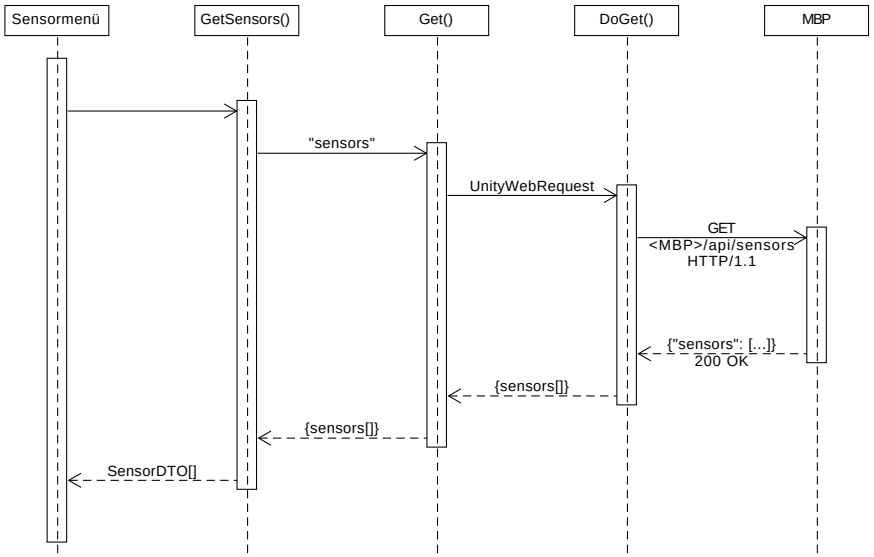


Abbildung 5.17: Vereinfachter Ablauf einer GET Request für Sensoren aus dem Sensormenü

Da der für die GET Requests verwendete DownloadHandler in der DoGet-Methode immer einen JSON-String als Response erwartet und diesen so direkt zurückgibt, ist diese Methode für die Icons der IoT-Komponenten ungeeignet. Hier wurde eine separate *DoGetMedia*-Methode angelegt, die

[14] UnityWebRequest: Unity-Klasse, die GET-, PUT- und POST-Requests ermöglicht

ein DownloadHandler-Objekt zurückgibt, um die Einfachheit der regulären DoGet-Methode nicht zu beeinträchtigen. Aus diesem wird in einer *GetImage*-Methode die Bilddatei extrahiert und direkt dem Dictionary der Icons angefügt.

Abgesehen von den GET Requests der Dictionaries und vereinzelter anderer Methoden, die von anderen Klassen im Projekt benötigt werden, führt der MBPConnectionManager auch POST Requests aus. Diese sind analog zu den GET Requests implementiert. Statt der Get-Methode existieren hier zwei Methoden, eine *PostWithPayload*-Methode und eine *PostWithoutPayload*-Methode, je nachdem ob die POST Request einen Payload benötigt. Beide Methoden verwenden jedoch dieselbe *DoPost*-Methode. Diese ist analog zur DoGet-Methode aufgebaut und setzt die Header, kodiert den Payload, falls nötig, sendet die Request und wartet auf das Ergebnis.

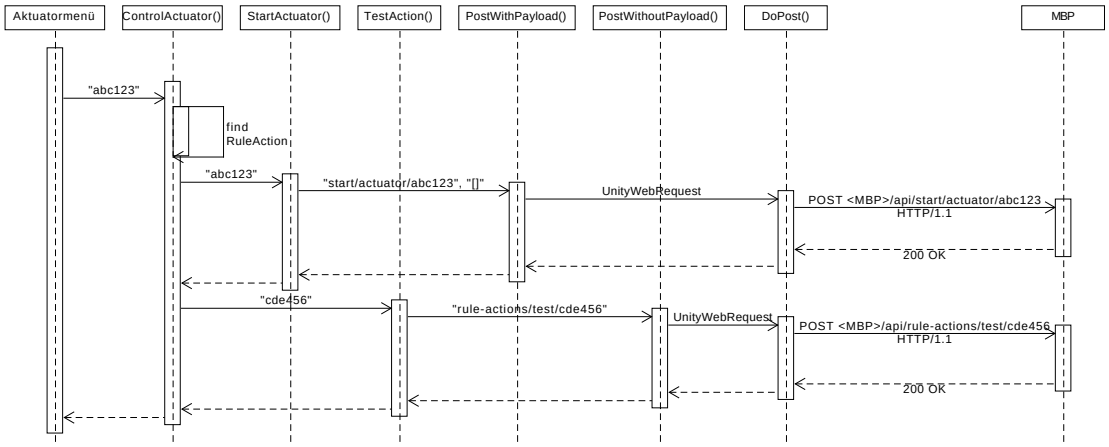


Abbildung 5.18: Vereinfachter Ablauf zweier POST Requests, um einen Aktuator aus dem Aktuatormenü zu steuern

Am Beispiel des Steuerns eines fiktiven Aktuators mit ID „abc123“ über ein Aktuatormenü zeigt Abbildung 5.18 den Ablauf einer solchen POST Request. Um einen Aktuator zu steuern, muss, wie in *Abschnitt 4.6* bereits erwähnt, zuvor eine RuleAction dazu existieren, die anschließend ausgeführt wird. Die Herangehensweise ist hier, beim Start der Anwendung und beim Neuladen der Dictionaries für alle Aktuatoren zu überprüfen, ob eine RuleAction existiert und falls nicht, eine neue anzulegen. Durch den Aufruf der *ControlActuator*-Methode wird die passende RuleAction mithilfe der Aktuator-ID gesucht, um diese im Anschluss auszuführen. Im Beispiel wird hier die RuleAction mit der ID „cde456“ gefunden. Anschließend wird eine POST Request mit Payload durchgeführt, um den Operator des Aktuators zu starten. Da diese Request optionale Parameter als Array erwartet, wird ein Payload mitgesendet. Selbst ohne Parameter, wie im Beispiel hier, muss so ein leeres Array mitgesendet werden. Nachdem der Operator ausgeführt wird, kann die *TestAction*-Methode eine POST Request ohne Payload durchführen, um die zuvor geladene RuleAction auszuführen.

Da das Starten jedes einzelnen Operators viel Zeit während dem Ausführen der Requests kostet, existiert auch eine *StartOperators*-Methode, die vom in *Abschnitt 5.5* beschriebenen Hand-Menü aus aufgerufen werden kann und alle Operatoren der MBP startet, die keine speziellen Parameter benötigen.

Alle in *Abschnitt 5.5* erwähnten Menüs erhalten ihre Daten aus den hier beschriebenen Dictionaries des MBPConnectionManagers. Durch das regelmäßige Aktualisieren der Daten wird der Inhalt dieser Menüs aktuell gehalten. Im Prototyp werden die Listen alle 30 Sekunden aktualisiert. Das Laden der Bilder und Überprüfen bzw. Erstellen der RuleAction geschieht alle fünf Minuten. Zusätzlich wird so auch ermöglicht, dass die AR-Anwendung unabhängig von der MBP gestartet werden kann. Ein Menü bleibt so lange leer und zeigt eine Ladeanimation, bis die MBP online ist und der MBPConnectionManager die Dictionaries aktualisiert hat. Auch ein Ausfall der Verbindung zur MBP kann durch diesen Cache ohne kompletten Funktionsverlust überstanden werden. Der Stand der Daten bleibt erhalten, bis die Verbindung wieder hergestellt ist.

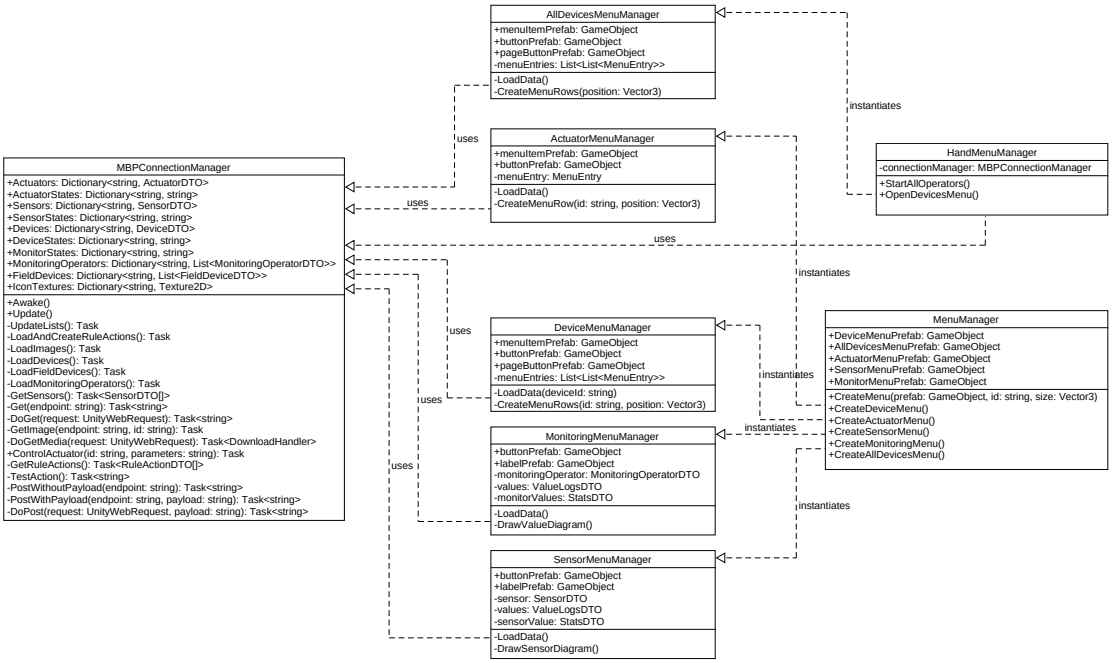


Abbildung 5.19: Vereinfachter Aufbau des Zusammenspiels aus MBPConnectionManager und den jeweiligen Menüs

Eine weitere wichtige Klasse des Prototyps ist der *MenuManager* mit gleichnamigem GameObject. Dieser ist für das Instanziieren der verschiedenen Menüs zuständig. Die Menüs sind dabei generisch aufgebaut und basieren jeweils auf einem Menü-Prefab, das beim Instanziieren in Griffreichweite vor dem Nutzer den Menühintergrund platziert. Buttons, Text und weitere Inhalte werden dabei automatisch anhand der Daten aus der MBP platziert. Hierfür werden an den Menü-Prefabs angehängte Manager-Skripte verwendet, die ausgeführt werden, sobald das Prefab instanziiert wird, sprich in der Awake-Methode.

Eines dieser Manager-Skripte ist der *HandMenuManager*, der die Funktionen des Handmenüs verwaltet. So können aus dem Handmenü, wie bereits beschrieben, alle Operatoren gestartet oder das Menü, um alle Geräte anzuzeigen, geöffnet werden. Für die Verwaltung dieses sogenannten „AllDevicesMenus“ existiert der *AllDevicesMenuManager*. Hier wird das Devices Dictionary aus dem MBPConnectionManager verwendet, um die verbundenen Geräte aufzulisten. Jeder Listeneintrag besteht, wie der Menühintergrund selbst, wiederum aus einem instanziierten Prefab,

das aus der MRTK-Bibliothek abgeleitet wurde. Der AllDevicesMenuManager enthält hier auch Methoden, um auf Inputs für Menüeinträge zu reagieren. Wird z. B. der Schließen-Button betätigt, wird das Menü-GameObject aus der Szene entfernt. Wird auf einen Menüeintrag für ein Gerät getippt, wird das Gerätemenü geöffnet.

Das Manager-Skript des Gerätemenüs ist der *DeviceMenuManager*. Dieses verwendet so z. B. den FieldDevices Dictionary aus dem MBPConnectionManager, um für jeden Sensor und Aktuator darin einen Listeneintrag im Menü zu erzeugen oder das MonitoringOperators Dictionary, um die MonitoringOperators aufzulisten. Wird beispielsweise auf einen Menüeintrag für einen Aktuator getippt, so wird via MBPConnectionManager die oben beschriebene POST Request zum Steuern dieses Aktuators gesendet. Dieselbe Funktionalität wird auch vom Aktuatormenü bzw. dem *ActuatorMenuManager* umgesetzt, falls direkt ein Aktuator via QR-Code ausgewählt wird.

Wird ein Sensor ausgewählt, so öffnet sich das Sensormenü. Ähnlich zum Geräte- und Aktuatormenü existiert hier ein Manager-Skript in Form des *SensorMenuManagers*. Dieses lädt die letzten 50 Sensorwerte per GET Request via MBPConnectionManager und speichert sie in einem lokalen Dictionary. Alle 15 Sekunden wird das Dictionary aktualisiert und dazu verwendet, ein Diagramm der Werte zu zeichnen. Wird im Gerätemenü auf einen Monitoring-Operator getippt, öffnet sich analog zum Sensormenü das Monitoring-Menü. Auch hier werden die letzten 50 Werte von der MBP geladen und in einem Diagramm dargestellt. Abbildung 5.19 zeigt eine vereinfachte Version des Zusammenspiels aus MBPConnectionManager und den hier behandelten Menüs.

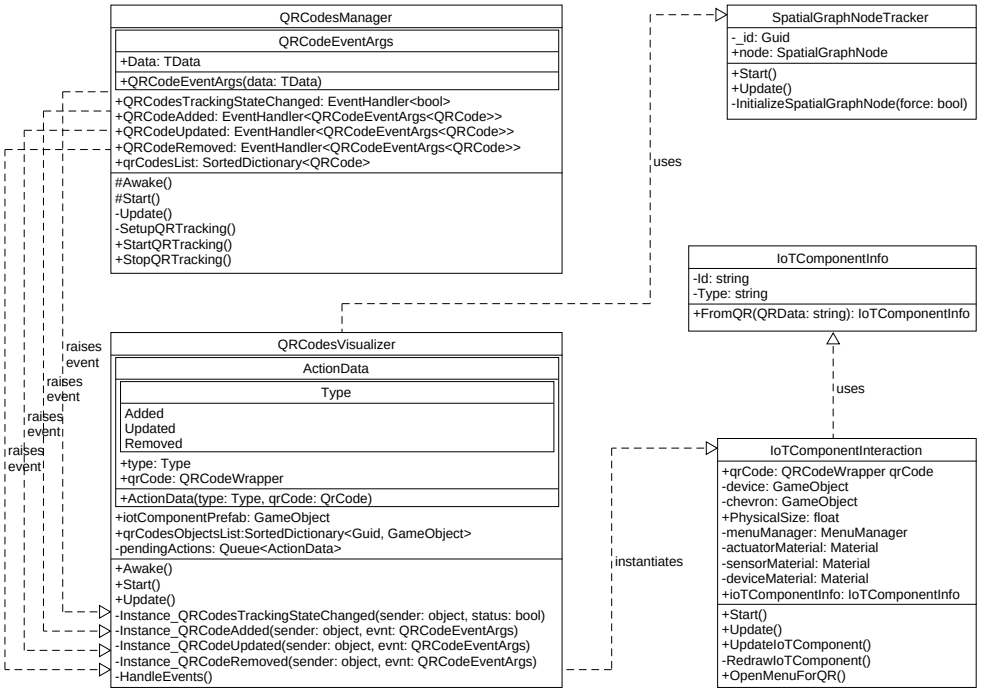


Abbildung 5.20: Vereinfachter Aufbau des QR-Tracking-Codes

Der ebenfalls beim Start der Anwendung aufgerufene *QRCodesManager*, welcher dem gleichnamigen GameObject in der Szene zugeordnet ist, ist Teil des QR-Code-Trackings. Dieser stammt, wie bereits in *Abschnitt 5.3* erwähnt, aus der Beispielszene der QR-Code-Bibliothek, startet das QR-Tracking und

sendet Events, beispielsweise sobald ein QR-Code erkannt wurde. Die Events des QRCodesManagers behandelt der *QRCodesVisualizer*, der ebenfalls dem QRCodesManager GameObject zugeordnet ist und auch auf dem Code der Beispielszene basiert. Wurde ein neuer QR-Code detektiert, wird hier ein vordefiniertes GameObject, in Abbildung 5.20 als „iotComponentPrefab“ zu sehen, an der Stelle des QR-Codes positioniert. Hierzu wird, mithilfe der QR-Code-Bibliothek, die Größe und Position des QR-Codes bestimmt. Das instanziierte GameObject namens „IoTComponent“ besteht, wie in Abbildung 5.13 zu sehen ist, aus einem Quader, der auf den QR-Code gelegt wird und einem Pfeil, der orthogonal zu dem QR-Code steht und die Interaktion ermöglicht. Das IoTComponent GameObject hat das *SpatialGraphNodeTracker*- und das *IoTComponentInteraction*-Skript angehängt. Der SpatialGraphNodeTracker ist für das Verwalten der *SpatialGraphNode*-Objekte der entsprechenden QR-Codes in der Szene verantwortlich. Die SpatialGraphNode-Objekte dienen hierbei als Speicher der Position und Rotation des QR-Codes [Khrb].

Das IoTComponentInteraction-Skript verwaltet das IoTComponent GameObject in der Szene. Einerseits wird hier mithilfe des *DeviceInfo*-Objekts die Art der IoT-Komponente aus dem QR-Code extrahiert, um das GameObject entsprechend einzufärben, sodass sich die Pfeile für Geräte, Sensoren und Aktuatoren visuell unterscheiden. Andererseits wird auch für das Verschieben und Skalieren des Quaders und des Pfeils über dem QR-Code anhand der extrahierten Informationen aus dem QR-Code gesorgt. Die *OpenMenuForQR*-Methode des IoTComponentInteraction-Skripts ist außerdem die Methode, die bei einer Far Interaction auf den Pfeil ausgeführt wird. Dazu enthält das Skript eine Referenz auf den MenuManager. So wird je nach Inhalt der DeviceInfo eines der bereits behandelten Menüs instanziiert. Diese Methode stellt so den Zusammenhang zwischen dem QR-Tracking und den Menüs bzw. dem MBPConnectionManager her. Ein vereinfachter Überblick über die Klassen des QR-Trackings befindet sich in Abbildung 5.20.

5.7 Hürden während der Entwicklung der Anwendung

Dadurch, dass die Entwicklung von HoloLens-AR-Anwendungen in Unity bisher kein etabliertes Gebiet der Software-Entwicklung ist, kann es unter Umständen zeitaufwendig sein, während der Entwicklung auftretende Probleme zu lösen. Im Folgenden sollen deshalb einige Lösungen, zu häufig während der Entwicklung der AR-Anwendung aufgetretenen Problemen, aufgezeigt werden.

Während des Aufsetzens des Projekts und Einrichtens des MRTKs war, wie in *Abschnitt 5.3* bereits erwähnt, das zum Zeitpunkt des Verfassens dieser Arbeit aktuellste Mixed Reality Feature Tool nur auf einem System mit einer älteren Windows-10-Version ausführbar und stürzte auf der neuesten Windows-Version ohne Fehlermeldung ab. Dies wurde durch ein Kopieren des Projekts auf ein anderes System mit einer älteren Windows-Version behoben. Dort konnte das Feature Tool ausgeführt und das Projekt anschließend wieder zurückkopiert werden. Ein manuelles Importieren der Bibliotheken ist auch möglich, wenn auch potentiell zeitaufwendiger.

Ein weiteres Problem, das bereits beim Anlegen des Unity-Projekts auftreten kann, ist abhängig vom Speicherort des Projekts. Da einige der Pfade innerhalb des MRTKs sehr lang sind, sollte das Unity-Projekt nicht in Unterordnern mit langem Pfade liegen. Ansonsten löst die Begrenzung von Windows-Dateipfaden auf 256 Zeichen Probleme aus.

Aus organisatorischen Gründen war die HoloLens erst gegen Ende der Bearbeitungszeit dieser Arbeit zum Testen verfügbar. So wurde anfänglich mit dem HoloLens-Emulator entwickelt. Da dieser nicht über die Möglichkeit verfügt, auf eine Kamera zuzugreifen, wurde die endgültige Entscheidung zum Tracking verschoben. Als zu einem entsprechend späteren Zeitpunkt die QR-Code-Bibliothek festgelegt wurde, musste auf eine neuere Unity-Version upgedated werden. Dieses Update bereitete wiederum Probleme mit der damals verwendeten Version des MRTK. Daraufhin wurde das Projekt mit der neuesten Versionen von Unity und dem MRTK neu angelegt. Der Code und die Prefabs konnten dabei in das neue Projekt importiert werden. Das Zusammenspiel aus der Menge an Zusätzen für Unity in einem AR-Projekt (MRTK, OpenXR, QR-Bibliothek, NuGet, etc.) erzeugt bei Updates Inkompatibilitäten. Aufgrund dessen wurde während des restlichen Verlaufs des Projekts kein weiteres Update für Unity oder eine dieser Komponenten installiert.

Nach dem Neuanlegen des Projekts wurde, wie oben erwähnt, versucht, die Skripte und Prefabs wieder aus dem alten Projekt in das neue zu kopieren. Dies wurde außerhalb von Unity durchgeführt, was Kompilierfehler zur Folge hatte. In Unity können Probleme auftreten, sofern Dateien extern abgeändert oder hinzugefügt werden. Dies ist mehrere Male während des Projekts in verschiedenen Situationen aufgetreten. Die Lösung des Problems war es, die Dateien wie in *Abschnitt 5.3* via Unity Import und nicht manuell zu importieren.

Nach dem Verschieben von Skripten in Unity ist es weiterhin mehrmals vorgekommen, dass diese nach einem Neustart von Unity doppelt existierten, einmal in der neuen Position und einmal in der vorherigen. Nach einer Umstrukturierung eines Unity-Projekts empfiehlt es sich, in Windows Explorer die Dateipfade zu überprüfen.

Es ist in Unity möglich, einen Build zum Debugging der HoloLens-Anwendung zu erstellen. Dieser ermöglicht es beispielsweise, Konsolen-Output in Visual Studio zu erhalten. Während des Entwickelns des Prototyps war dies ein hilfreiches Werkzeug. Sollte das Deployment eines solchen Builds jedoch fehlschlagen, muss die zuvor als Release kompilierte Anwendung von der HoloLens deinstalliert werden. Hierzu kann in der Anwendungsübersicht der HoloLens auf den Eintrag der installierten Anwendung lange gedrückt werden. Es empfiehlt sich, bei Deployment-Problemen die HoloLens neu zu starten.

Eine weitere Schwierigkeit, die während des Builds in Unity oft auftrat ist, dass eine Fehlermeldung der Form „The process cannot access the file because it is being used by another process“ angezeigt wurde. Dies kann dadurch gelöst werden, Visual Studio während des Builds zu schließen.

Da der Build und das Deployment auf die HoloLens oder den Emulator lange dauern kann, können Funktionen, die keine spezielle Hardware der HoloLens benötigen, auch direkt in Unity getestet werden. Hierzu verfügt Unity über den sogenannten *Play Mode*. Dieser kann durch das Play-Symbol in der Mitte der Unity Toolbar gestartet werden. Dabei wird die Anwendung im Unity-Fenster ausgeführt. Diese Funktion wurde beispielsweise für das Testen der REST-API verwendet, da hier weder Kameras noch Sensoren der HoloLens benötigt wurden.

6 Fazit und Ausblick

In dieser Arbeit wurde untersucht, ob sich eine auf der Microsoft HoloLens basierende Augmented-Reality-Anwendung für die Interaktion mit der MBP-IoT-Plattform eignet. Dabei wurde nach ausführlicher Literaturrecherche ein Prototyp mithilfe der Unity Spiele-Engine entwickelt. Im Laufe der Entwicklung und während zahlreicher Tests wurde das folgende Fazit gesammelt.

Fazit

Wie in *Kapitel 5* dargestellt, eignete sich die Microsoft HoloLens zur Entwicklung einer AR-Anwendung, um mit der MBP zu kommunizieren. Die Grundfunktionalitäten des Auslesens von Sensorwerten und Monitoring-Daten und des Steuerns von Aktuatoren funktionierte dabei problemlos. Während verschiedener Tests der Anwendung fiel jedoch auf, dass die Rechenleistung der HoloLens vereinzelt nicht ausreichend war. Sobald viele sich bewegende Objekte dargestellt werden, wie z. B. die Pfeile der IoT-Komponenten und Menüs mit Billboarding, sinkt die Framerate weit unter 60 Bilder pro Sekunde und es kann zu Verzögerungen kommen.

Während der Entwicklung fiel außerdem oft auf, dass die Dokumentation zur Entwicklung von AR-Anwendungen in Unity veraltet und deshalb nicht mehr uneingeschränkt anwendbar war. Dies ist der Geschwindigkeit von neuen Entwicklungen auf dem Gebiet geschuldet.

Der zweistufige Kompiliervorgang aus Unity und Visual Studio war während häufig ausgeführten Tests störend. Im Falle des hier verwendeten Entwicklersystems dauerte es oft über 5 Minuten, bis die Anwendung auf der HoloLens ausgeführt wurde. Debugging Builds benötigten manchmal sogar 10 Minuten.

Während längerer Phasen der Entwicklung, stellte auch die Akkulaufzeit gelegentlich ein Problem dar. Oft musste die HoloLens nach drei Stunden wieder aufgeladen werden. Diese kurze Akkulaufzeit, speziell im Vergleich zu Smartphones oder Tablets, ist für ein reales Anwendungsszenario problematisch. Positiv ist jedoch, dass die HoloLens einen standardisierten USB-C-Anschluss verwendet und das Schnellladen mit 18 Watt unterstützt. Potentiell könnte so auch die Akkulaufzeit mit einer Powerbank für Smartphones verlängert werden. Selbst ohne das Gewicht einer zusätzlichen Powerbank fühlte sich die HoloLens nach längerer Verwendung oft schwer an und kann im Stirnbereich zu Druckstellen führen. Die Tatsache, dass die HoloLens jedoch komplett kabellos einsetzbar ist, ist speziell im Vergleich zu VR Headsets von Vorteil und erleichtert die Handhabung. Auch während des Entwickelns war dies vorteilhaft, da so auf dem Entwicklersystem, z. B. Logs betrachtet werden oder ein neuer Build deployt werden konnte, ohne die HoloLens abzunehmen und an das System anzuschließen.

Die Displays der HoloLens sind bei normalen Lichtverhältnissen in Innenräumen ausreichend hell und auch in dunkleren Umgebungen, aufgrund der einstellbaren Helligkeit, noch nutzbar. Hier lässt jedoch die Genauigkeit des Trackings, speziell von QR-Codes, stark nach. Bei direkter Sonneneinstrahlung durch ein Fenster kann das Bild möglicherweise als zu dunkel empfunden werden.

Neben den Problemen bei hoher Helligkeit ist auch die Umgebungsverdeckung ein Problem. Wie bei allen Optical See-Through Head Mounted Displays kann auch die HoloLens keine Objekte in der realen Welt überdecken. So kann ein mit der HoloLens dargestelltes Menü, beispielsweise auf stark texturierten Hintergründen, schwer zu erkennen sein.

Auch die Begrenzung des Sichtfeld kann zu unerwünschten Effekten führen, speziell wenn der Nutzer durch den Raum blickt und Objekte bereits vor Ende des Gesichtsfeldes verschwinden. Nähert sich der Nutzer einem Objekt so weit an, dass dieses die komplette Größe des Sichtfelds einnimmt, ist der Effekt am auffälligsten. Trotzdem ist das Sichtfeld der HoloLens 2 spürbar besser als bei der ersten Generation.

Ein weiterer Nachteil der HoloLens im Vergleich zu Smartphone-AR ist, dass zum Zeitpunkt des Schreibens dieser Arbeit nahezu jede Person bereits ein Smartphone besitzt und auch jederzeit mit sich führt. Die HoloLens ist hierbei durch ihren Preis und ihre Größe noch nicht für Consumer-Anwendungen geeignet.

Nach kurzen Testvorführungen der Anwendung vor Personen, die noch nie eine AR-Brille verwendet hatten, kam auch das Feedback, wie „futuristisch“ die HoloLens sei. Ähnlich wie bei dem Aufkommen der Smartphones vor einigen Jahren könnte auch diese Begeisterung von etwas Neuem ein Vorteil für die Akzeptanz von AR-Brillen sein.

Die Gestensteuerung, die wie ein virtueller Touchscreen funktioniert, wurde auch als sehr intuitiv empfunden. Bei der Verwendung der HoloLens sollte jedem Nutzer eines Touchscreen-Geräts klar sein, wie die Interaktion funktioniert. Speziell die Far-Interaction-Geste zeigt diese Intuitivität, da hier lediglich auf ein Element gezeigt werden muss, mit dem interagiert werden soll. Die Bestätigungsgeste mit Zeigefinger und Daumen ist jedoch weniger intuitiv und funktioniert je nach Blickwinkel schlechter. Hier verschafft jedoch die Spracheingabe Abhilfe, mit der Interaktionen ebenfalls bestätigt werden können. Die Sprachsteuerung ist für Texteingaben auch besser geeignet als die virtuelle Tastatur. Beides eignet sich jedoch nicht für komplexere Texte.

Ausblick

Wie oben beschrieben, eignet sich die HoloLens 2 bereits heute für die Interaktion mit einer IoT-Plattform wie der MBP. Viele der hier aufgezählten Probleme, wie die Akkulaufzeit, das Gewicht, die Rechenleistung oder Probleme mit der Entwicklungsumgebung, werden höchstwahrscheinlich in wenigen Gerätegenerationen nicht mehr existieren, insbesondere wenn der Unterschied zwischen der ersten und zweiten Generation HoloLens betrachtet wird.

Technisch könnte bei der beschriebenen Anwendung versucht werden, die Hologramme auch nach einem Neustart der HoloLens persistent zu halten. Hier war das Problem, dass das für die Positionierung verwendete Koordinatensystem seinen Ursprung an dem Punkt hat, an dem die Anwendung gestartet wird. Testweise wurde eine solche Funktion auch implementiert und

ermöglichte persistente Hologramme, solange die Startposition dieselbe war. Um die Interaktion mit Geräten wie dem Bosch XDK, das zusätzliche Parameter zum Starten benötigt, zu vereinfachen, könnte auch versucht werden, eine Liste an möglichen Parametern über die MBP-API zur Verfügung zu stellen. Anhand dieser könnte ein Dropdown in den Menüs erstellt werden. So müsste der Nutzer dies nicht per Tastatur oder Sprache eingeben.

Aus Zeitgründen und der anhaltenden COVID-19-Pandemie konnte während des Bearbeitungszeitraums dieser Arbeit keine Nutzerstudie durchgeführt werden. In einer künftigen Arbeit könnte so die AR-Anwendung beispielsweise mit einer regulären Smartphone-App verglichen werden. Weiterhin kann versucht werden, die Anwendung auf Android und iOS zu portieren, um so Smartphone-AR mit HMD-AR zu vergleichen.

Literaturverzeichnis

[AA17] M. Akçayır, G. Akçayır. „Advantages and challenges associated with augmented reality for education: A systematic review of the literature“. In: *Educational Research Review* 20 (2017), S. 1–11 (zitiert auf S. 9, 18).

[Ada14] Adafruit. *Python Usage | NeoPixels on Raspberry Pi | Adafruit Learning System*. https://learn.adafruit.com/neopixels-on-raspberry-pi/python-usage. Sep. 2014 (zitiert auf S. 52).

[AKBH17] M. F. Alam, S. Katsikas, O. Beltramello, S. Hadjiefthymiades. „Augmented and virtual reality based monitoring and safety system: A prototype IoT platform“. In: *Journal of Network and Computer Applications* 89 (2017). Emerging Services for Internet of Things (IoT), S. 109–119. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2017.03.022. URL: https://www.sciencedirect.com/science/article/pii/S1084804517301315 (zitiert auf S. 31).

[Ald03] F. K. Aldrich. „Smart homes: past, present and future“. In: *Inside the smart home*. Springer, 2003, S. 17–39 (zitiert auf S. 14).

[Ama06] Amazon. *Announcing Amazon Elastic Compute Cloud (Amazon EC2) - beta*. https://aws.amazon.com/de/about-aws/whats-new/2006/08/24/announcing-amazon-elastic-compute-cloud-amazon-ec2---beta/. Aug. 2006 (zitiert auf S. 13).

[AMPT18] D. Agrawal, S. B. Mane, A. Pacharne, S. Tiwari. „IoT Based Augmented Reality System of Human Heart: An Android Application“. In: *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*. 2018, S. 899–902. DOI: 10.1109/ICOEI.2018.8553807 (zitiert auf S. 27).

[ARRA19] M. A. Ankireddy, A. Rajath, M. Ruthwik Ganesh, M. Anuradha. „Augmented Reality Rendered for IoT applications“. In: *2019 IEEE 16th India Council International Conference (INDICON)*. 2019, S. 1–4. DOI: 10.1109/INDICON47234.2019.9029045 (zitiert auf S. 37).

[Ash+09] K. Ashton et al. „That 'internet of things' thing“. In: *RFID journal* 22.7 (2009), S. 97–114 (zitiert auf S. 11).

[Azu97] R. T. Azuma. „A survey of augmented reality“. In: *Presence: Teleoperators & Virtual Environments* 6.4 (1997), S. 355–385 (zitiert auf S. 16, 19).

[AZZ+17] M. Alaa, A. A. Zaidan, B. B. Zaidan, M. Talal, M. L. M. Kiah. „A review of smart home applications based on Internet of Things“. In: *Journal of Network and Computer Applications* 97 (2017), S. 48–65 (zitiert auf S. 14).

[BFAF20] Ó. Blanco-Novoa, P. Fraga-Lamas, M. A. Vilar-Montesinos, T. M. Fernández-Caramés. „Creating the Internet of Augmented Things: An Open-Source Framework to Make IoT Devices and Augmented and Mixed Reality Systems Talk to Each Other“. In: *Sensors* 20.11 (2020). ISSN: 1424-8220. DOI: 10.3390/s20113328. URL: https://www.mdpi.com/1424-8220/20/11/3328 (zitiert auf S. 32, 33, 42).

[BFC+14] M. Bordegoni, F. Ferrise, E. Carrabba, M. Di Donato, M. Fiorentino, A. E. Uva. „An application based on Augmented Reality and mobile technology to support remote maintenance“. In: *Conference and Exhibition of the European Association of Virtual and Augmented Reality*. Bd. 1. 1. The Eurographics Association Geneve. 2014, S. 131–135 (zitiert auf S. 31).

[BFFV18] Ó. Blanco-Novoa, T. M. Fernández-Caramés, P. Fraga-Lamas, M. A. Vilar-Montesinos. „A Practical Evaluation of Commercial Industrial Augmented Reality Systems in an Industry 4.0 Shipyard“. In: *IEEE Access* 6 (2018), S. 8201–8218. DOI: 10.1109/ACCESS.2018.2802699 (zitiert auf S. 29, 30).

[BFVF20] Ó. Blanco-Novoa, P. Fraga-Lamas, M. A. Vilar-Montesinos, T. M. Fernández-Caramés. „Towards the Internet of Augmented Things: An Open-source Framework to Interconnect IoT Devices and Augmented Reality Systems“. In: *Proceedings* 42.1 (2020). ISSN: 2504-3900. DOI: 10.3390/ecsa-6-06563. URL: https://www.mdpi.com/2504-3900/42/1/50 (zitiert auf S. 32).

[BMZA12] F. Bonomi, R. Milito, J. Zhu, S. Addepalli. „Fog computing and its role in the internet of things“. In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM. 2012, S. 13–16 (zitiert auf S. 13).

[BN18] S. Baskaran, H. K. Nagabushanam. „Relational localization based Augmented reality Interface for IOT applications“. In: *2018 International Conference on Information and Communication Technology Convergence (ICTC)*. 2018, S. 103–106. DOI: 10.1109/ICTC.2018.8539469 (zitiert auf S. 39).

[Bos] Bosch. *Developer Portal*. https://developer.bosch.com/products-and-services/sdks/xdk (zitiert auf S. 44).

[BPG10] I. Barandiaran, C. Paloc, M. Graña. „Real-time optical markerless tracking for augmented reality applications“. In: *Journal of Real-Time Image Processing* 5.2 (2010), S. 129–138 (zitiert auf S. 20).

[BRS19] V. Becker, F. Rauchenstein, G. Sörös. „Investigating Universal Appliance Control through Wearable Augmented Reality“. In: *Proceedings of the 10th Augmented Human International Conference 2019*. AH2019. Reims, France: Association for Computing Machinery, 2019. ISBN: 9781450365475. DOI: 10.1145/3311823.3311853 (zitiert auf S. 9, 36, 37).

[CGK+13] A. G. Campbell, L. Görgü, B. Kroon, D. Lillis, D. Carr, G. M. O'Hare. „Giving mobile devices a SIXTH sense: Introducing the SIXTH middleware for Augmented Reality applications“. In: *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2013, S. 245–246. DOI: 10.1109/ISMAR.2013.6671787 (zitiert auf S. 40).

[CM92] T. P. Caudell, D. W. Mizell. „Augmented reality: An application of heads-up display technology to manual manufacturing processes“. In: *Hawaii international conference on system sciences*. 1992, S. 659–669 (zitiert auf S. 18).

[Com15] F. Computing. „the Internet of Things: Extend the Cloud to Where the Things are“. In: *Cisco White Paper* (2015) (zitiert auf S. 13).

[CPG+15] D. Chaves-Diéguez, A. Pellitero-Rivero, D. García-Coego, F. J. González-Castaño, P. S. Rodríguez-Hernández, Ó. Piñeiro-Gómez, F. Gil-Castiñeira, E. Costa-Montenegro. „Providing IoT Services in Smart Cities through Dynamic Augmented Reality Markers“. In: *Sensors* 15.7 (2015), S. 16083–16104. ISSN: 1424-8220. DOI: 10.3390/s150716083. URL: https://www.mdpi.com/1424-8220/15/7/16083 (zitiert auf S. 20).

[CPTT18] M. Collotta, G. Pau, T. Talty, O. K. Tonguz. „Bluetooth 5: A concrete step forward toward the IoT“. In: *IEEE Communications Magazine* 56.7 (2018), S. 125–131 (zitiert auf S. 12).

[CSD93] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti. „Surround-screen projection-based virtual reality: the design and implementation of the CAVE“. In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. 1993, S. 135–142 (zitiert auf S. 18).

[CXL+19] Y. Cao, Z. Xu, F. Li, W. Zhong, K. Huo, K. Ramani. „V.Ra: An In-Situ Visual Authoring System for Robot-IoT Task Planning with Augmented Reality“. In: *Proceedings of the 2019 on Designing Interactive Systems Conference*. DIS '19. San Diego, CA, USA: Association for Computing Machinery, 2019, S. 1059–1070. ISBN: 9781450358507. DOI: 10.1145/3322276.3322278 (zitiert auf S. 41).

[DG] B. C. Devices, S. GmbH. *xdk_node_110_combined_datasheet*. https://www.bosch-connectivity.com/media/downloads/xdk/xdk_node_110_combined_datasheet.pdf (zitiert auf S. 43).

[Did05] J.-Y. Didier. „Contributions á la dextérité d'un système de réalité augmentée mobile appliqué á la maintenance industrielle“. Diss. Université d'Evry-Val d'Essonne, 2005 (zitiert auf S. 20).

[DM91] D. Drascic, P. Milgram. „Positioning accuracy of a virtual stereographic pointer in a real stereoscopic video world“. In: *Stereoscopic Displays and Applications II*. Bd. 1457. International Society for Optics und Photonics. 1991, S. 302–313 (zitiert auf S. 9, 16).

[DMP+02] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, B. Weihl. „Globally distributed content delivery“. In: *IEEE Internet Computing* 6.5 (2002), S. 50–58. DOI: 10.1109/MIC.2002.1036038 (zitiert auf S. 13).

[Dri] E. Driscoll. *The history of X10*. http://home.kpn.nl/lhendrix/x10_history.htm (zitiert auf S. 14).

[DSTB17] A. Doshi, R. T. Smith, B. H. Thomas, C. Bouras. „Use of projector based augmented reality to improve manual spot-welding precision and accuracy for automotive manufacturing“. In: *The International Journal of Advanced Manufacturing Technology* 89.5-8 (2017), S. 1279–1293 (zitiert auf S. 18).

[DTD19] M. De Donno, K. Tange, N. Dragoni. „Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog“. In: *IEEE Access* 7 (2019), S. 150936–150948. DOI: 10.1109/ACCESS.2019.2947652 (zitiert auf S. 11).

[DV19] S. J. Danbatta, A. Varol. „Comparison of Zigbee, Z-Wave, Wi-Fi, and bluetooth wireless technologies used in home automation“. In: *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*. IEEE. 2019, S. 1–5 (zitiert auf S. 12).

[EKEL19] B. Eckstein, E. Krapp, A. Elsässer, B. Lugrin. „Smart substitutional reality: Integrating the smart home into virtual reality“. In: *Entertainment Computing* 31 (2019), S. 100306 (zitiert auf S. 38).

[EMP+17] G. Evans, J. Miller, M. I. Pena, A. MacAllister, E. Winer. „Evaluating the Microsoft HoloLens through an augmented reality assembly application“. In: *Degraded environments: sensing, processing, and display 2017*. Bd. 10197. International Society for Optics und Photonics. 2017, S. 101970V (zitiert auf S. 31).

[Erg04] S. C. Ergen. „ZigBee/IEEE 802.15. 4 Summary“. In: *UC Berkeley, September* 10.17 (2004), S. 11 (zitiert auf S. 12).

[FFSV18] T. M. Fernández-Caramés, P. Fraga-Lamas, M. Suárez-Albela, M. Vilar-Montesinos. „A Fog Computing and Cloudlet Based Augmented Reality System for the Industry 4.0 Shipyard“. In: *Sensors* 18.6 (2018). ISSN: 1424-8220. DOI: 10.3390/s18061798. URL: https://www.mdpi.com/1424-8220/18/6/1798 (zitiert auf S. 29).

[FMHW97] S. Feiner, B. MacIntyre, T. Höllerer, A. Webster. „A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment“. In: *Personal Technologies* 1.4 (1997), S. 208–217 (zitiert auf S. 18).

[Fou] R. P. Foundation. *Buy a Raspberry Pi 3 Model A+ – Raspberry Pi*. https://www.raspberrypi.org/products/raspberry-pi-3-model-a-plus/ (zitiert auf S. 44).

[Fri] Fritzing. *Fritzing*. https://fritzing.org/ (zitiert auf S. 47).

[FSA20] P. Fleck, D. Schmalstieg, C. Arth. „Creating IoT-ready XR-WebApps with Unity3D“. In: *The 25th International Conference on 3D Web Technology*. 2020, S. 1–7 (zitiert auf S. 40).

[GAEA11] J. A. Garcia Macias, J. Alvarez-Lozano, P. Estrada, E. Aviles Lopez. „Browsing the Internet of Things with Sentient Visors“. In: *Computer* 44.5 (2011), S. 46–52. DOI: 10.1109/MC.2011.128 (zitiert auf S. 28).

[GIMA10] D. Giusto, A. Iera, G. Morabito, L. Atzori. *The internet of things: 20th Tyrrhenian workshop on digital communications*. Springer Science & Business Media, 2010 (zitiert auf S. 11).

[GMMM14] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, M. J. Marín-Jiménez. „Automatic generation and detection of highly reliable fiducial markers under occlusion“. In: *Pattern Recognition* 47.6 (2014), S. 2280–2292 (zitiert auf S. 19).

[GSG18] A. Gabbiadini, C. Sagioglou, T. Greitemeyer. „Does Pokémon Go lead to a more physically active life style?“ In: *Computers in Human Behavior* 84 (2018), S. 258–263 (zitiert auf S. 18).

[Hal19] J. Halpern. *Developing 2D Games with Unity*. Springer, 2019 (zitiert auf S. 22).

[Har03] R. Harper. „Inside the smart home: Ideas, possibilities and methods“. In: *Inside the smart home*. Springer, 2003, S. 1–13 (zitiert auf S. 14).

[HCY+18] K. Huo, Y. Cao, S. H. Yoon, Z. Xu, G. Chen, K. Ramani. „Scenariot: Spatially Mapping Smart Things Within Augmented Reality Scenes“. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18. Montreal QC, Canada: Association for Computing Machinery, 2018, S. 1–13. ISBN: 9781450356206. DOI: 10.1145/3173574.3173793 (zitiert auf S. 35).

[HF11] S. Henderson, S. Feiner. „Exploring the Benefits of Augmented Reality Documentation for Maintenance and Repair“. In: *IEEE Transactions on Visualization and Computer Graphics* 17.10 (2011), S. 1355–1368. DOI: 10.1109/TVCG.2010.245 (zitiert auf S. 31).

[HKBW18] S. Haesler, K. Kim, G. Bruder, G. Welch. „Seeing is Believing: Improving the Perceived Trust in Visually Embodied Alexa in Augmented Reality“. In: *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. 2018, S. 204–205. DOI: 10.1109/ISMAR-Adjunct.2018.00067 (zitiert auf S. 38).

[HLL12] B.-R. Huang, C. H. Lin, C.-H. Lee. „Mobile augmented reality based on cloud computing“. In: *Anti-counterfeiting, Security, and Identification*. 2012, S. 1–5. DOI: 10.1109/ICASID.2012.6325354 (zitiert auf S. 42).

[HLRJ21] J. Hua, S. Lee, G.-C. Roman, C. Julien. „ArcIoT: Enabling Intuitive Device Control in the Internet of Things through Augmented Reality“. In: *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE. 2021, S. 558–564 (zitiert auf S. 22, 35, 36).

[HO18] J. Huuskonen, T. Oksanen. „Soil sampling with drones and augmented reality in precision agriculture“. In: *Computers and Electronics in Agriculture* 154 (2018), S. 25–35 (zitiert auf S. 26).

[HSK+19] B. Hoppenstedt, M. Schmid, K. Kammerer, J. Scholta, M. Reichert, R. Pryss. „Analysis of Fuel Cells Utilizing Mixed Reality and IoT Achievements“. In: *Augmented Reality, Virtual Reality, and Computer Graphics*. Hrsg. von L. T. De Paolis, P. Bourdot. Cham: Springer International Publishing, 2019, S. 371–378. ISBN: 978-3-030-25999-0 (zitiert auf S. 15, 29).

[IAAH16] M. A. Iqbal, S. Asrafuzzaman, M. M. Arifin, S. A. Hossain. „Smart home appliance control system for physically disabled people using kinect and X10“. In: *2016 5th International Conference on Informatics, Electronics and Vision (ICIEV)*. IEEE. 2016, S. 891–896 (zitiert auf S. 14, 37).

[IGF+17] J. E. Ibarra-Esquer, F. F. González-Navarro, B. L. Flores-Rios, L. Burtseva, M. A. Astorga-Vargas. „Tracking the Evolution of the Internet of Things Concept Across Different Application Domains“. In: *Sensors* 17.6 (2017). ISSN: 1424-8220. DOI: 10.3390/s17061379. URL: https://www.mdpi.com/1424-8220/17/6/1379 (zitiert auf S. 11).

[Ind] A. Industries. *NeoPixel Ring - 24 x 5050 RGB LED with Integrated Drivers : ID 1586 : $16.95 : Adafruit Industries, Unique & fun DIY electronics and kits*. https://www.adafruit.com/product/1586 (zitiert auf S. 44).

[JB18] J. Jang, T. Bednarz. „HoloSensor for Smart Home, Health, Entertainment“. In: *ACM SIGGRAPH 2018 Appy Hour*. SIGGRAPH '18. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2018. ISBN: 9781450358071. DOI: 10.1145/3213779.3213786 (zitiert auf S. 34).

[JCKL19] K. M. Jeon, C. J. Chun, H. K. Kim, M. J. Lee. „User-Aware Audio Marker Using Low Frequency Ultrasonic Object Detection and Communication for Augmented Reality“. In: *Applied Sciences* 9.10 (2019), S. 2004 (zitiert auf S. 22).

[JJH+15] Y. Jeong, H. Joo, G. Hong, D. Shin, S. Lee. „AVIoT: web-based interactive authoring and visualization of indoor internet of things“. In: *IEEE Transactions on Consumer Electronics* 61.3 (2015), S. 295–301. DOI: 10.1109/TCE.2015.7298088 (zitiert auf S. 40).

[JK16] D. Jo, G. J. Kim. „ARIoT: scalable augmented reality framework for interacting with Internet of Things appliances everywhere“. In: *IEEE Transactions on Consumer Electronics* 62.3 (2016), S. 334–340. DOI: 10.1109/TCE.2016.7613201 (zitiert auf S. 32, 40).

[JK19] D. Jo, G. J. Kim. „IoT+ AR: pervasive and augmented environments for “Digi-log” shopping experience“. In: *Human-centric Computing and Information Sciences* 9.1 (2019), S. 1–17 (zitiert auf S. 42).

[KAKE19] A. M. Karadeniz, Í. Arif, A. Kanak, S. Ergün. „Digital Twin of eGastronomic Things: A Case Study for Ice Cream Machines“. In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2019, S. 1–4. DOI: 10.1109/ISCAS.2019.8702679 (zitiert auf S. 41, 42).

[Kau09] L. M. Kaufman. „Data security in the world of cloud computing“. In: *IEEE Security & Privacy* 7.4 (2009), S. 61–64 (zitiert auf S. 13).

[KBB+18] K. Kim, M. Billinghurst, G. Bruder, H. B.-L. Duh, G. F. Welch. „Revisiting trends in augmented reality research: A review of the 2nd decade of ISMAR (2008–2017)“. In: *IEEE transactions on visualization and computer graphics* 24.11 (2018), S. 2947–2962 (zitiert auf S. 19).

[KHLI12] S. Kasahara, V. Heun, A. S. Lee, H. Ishii. „Second Surface: Multi-User Spatial Collaboration System Based on Augmented Reality“. In: *SIGGRAPH Asia 2012 Emerging Technologies*. SA '12. Singapore, Singapore: Association for Computing Machinery, 2012, S. 1–4. ISBN: 9781450319126. DOI: 10.1145/2407707.2407727 (zitiert auf S. 17).

[Khra] Khronos. *OpenXR Overview - The Khronos Group Inc*. https://www.khronos.org/openxr/ (zitiert auf S. 56).

[Khrb] Khronos. *The OpenXR Specification*. https://www.khronos.org/registry/OpenXR/specs/1.0/html/xrspec.html#xrCreateSpatialGraphNodeSpaceMSFT (zitiert auf S. 73).

[KJG+18] G. Koutitas, J. Jabez, C. Grohman, C. Radhakrishna, V. Siddaraju, S. Jadon. „Demo/poster abstract: XReality research lab — Augmented reality meets Internet of Things“. In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2018, S. 1–2. DOI: 10.1109/INFCOMW.2018.8406848 (zitiert auf S. 33).

[KK17] A. Katsaros, E. Keramopoulos. „FarmAR, a farmer's augmented reality application based on semantic web". In: *2017 South Eastern European Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. 2017, S. 1–6. DOI: 10.23919/SEEDA-CECNSM.2017.8088230 (zitiert auf S. 26).

[KKS17] A. Katsaros, E. Keramopoulos, M. Salampasis. „A prototype Application for cultivation optimization using augmented reality". In: *CEUR Workshop. In Proceedings of the 8th International Conference on Information and Communication Technologies in Agriculture, Chania, Crete Island, Greece*. 2017, S. 21–24 (zitiert auf S. 26).

[KLA20] E. Khorov, I. Levitsky, I. F. Akyildiz. „Current status and directions of IEEE 802.11 be, the future Wi-Fi 7". In: *IEEE Access* 8 (2020), S. 88664–88688 (zitiert auf S. 12).

[KLKG15] E. Khorov, A. Lyakhov, A. Krotov, A. Guschin. „A survey on IEEE 802.11 ah: An enabling networking technology for smart cities". In: *Computer communications* 58 (2015), S. 53–69 (zitiert auf S. 12).

[KNKA14] C. Koch, M. Neges, M. König, M. Abramovici. „Natural markers for augmented reality-based indoor navigation and facility maintenance". In: *Automation in Construction* 48 (2014), S. 18–30 (zitiert auf S. 21).

[KWAS19] P. Knierim, P. W. Woundefinedniak, Y. Abdelrahman, A. Schmidt. „Exploring the Potential of Augmented Reality in Domestic Environments". In: MobileHCI '19. Taipei, Taiwan: Association for Computing Machinery, 2019. ISBN: 9781450368254. DOI: 10.1145/3338286.3340142 (zitiert auf S. 38).

[Lin16a] Linkerkit. *Datei:7374afc1cd00c0f723bcee7c7e2acd0f.media.400x352.png – Linkerkit.de*. https://www.linkerkit.de/index.php?title=Datei:7374afc1cd00c0f723bcee7c7e2acd0f.media.400x352.png. Aug. 2016 (zitiert auf S. 44).

[Lin16b] Linkerkit. *Datei:LK-GeräuscheSen.jpg – Linkerkit.de*. https://www.linkerkit.de/index.php?title=Datei:LK-GeräuscheSen.jpg. Aug. 2016 (zitiert auf S. 44).

[Lin16c] Linkerkit. *Datei:LK-Temp.png – Linkerkit.de*. https://www.linkerkit.de/index.php?title=Datei:LK-Temp.png. Aug. 2016 (zitiert auf S. 44).

[Lob16] N. Lobo. „Intelli-mirror: An augmented reality based IoT system for clothing and accessory display". In: *2016 International Conference on Internet of Things and Applications (IOTA)*. 2016, S. 95–100. DOI: 10.1109/IOTA.2016.7562702 (zitiert auf S. 42).

[LQG17] G. López, L. Quesada, L. A. Guerrero. „Alexa vs. Siri vs. Cortana vs. Google Assistant: a comparison of speech-based natural user interfaces". In: *International Conference on Applied Human Factors and Ergonomics*. Springer. 2017, S. 241–250 (zitiert auf S. 14).

[Lu17] Y. Lu. „Industry 4.0: A survey on technologies, applications and open research issues". In: *Journal of industrial information integration* 6 (2017), S. 1–10 (zitiert auf S. 14).

[MAC18] K. Michalakis, J. Aliprantis, G. Caridakis. „Visualizing the Internet of Things: Naturalizing Human-Computer Interaction by Incorporating AR Features“. In: *IEEE Consumer Electronics Magazine* 7.3 (2018), S. 64–72. DOI: 10.1109/MCE.2018.2797638 (zitiert auf S. 20).

[MAK13] L. Müller, I. Aslan, L. Krüßen. „GuideMe: A mobile augmented reality system to display user manuals for home appliances“. In: *International Conference on Advances in Computer Entertainment Technology*. Springer. 2013, S. 152–167 (zitiert auf S. 42).

[MDA+20] B. Marques, P. Dias, J. Alves, E. Fonseca, B. S. Santos. „Configuration and Use of Pervasive Augmented Reality Interfaces in a Smart Home Context: A Prototype“. In: *Human Systems Engineering and Design II*. Hrsg. von T. Ahram, W. Karwowski, S. Pickl, R. Taiar. Cham: Springer International Publishing, 2020, S. 96–102. ISBN: 978-3-030-27928-8 (zitiert auf S. 35).

[MDAM10] M. Maidi, J.-Y. Didier, F. Ababsa, M. Mallem. „A performance study for camera pose estimation using visual marker based tracking“. In: *Machine Vision and Applications* 21.3 (2010), S. 365–376 (zitiert auf S. 20).

[ME19] T. Masood, J. Egger. „Augmented reality in support of Industry 4.0—Implementation challenges and success factors“. In: *Robotics and Computer-Integrated Manufacturing* 58 (2019), S. 181–195 (zitiert auf S. 31).

[Meu17] R. van der Meulen. *8.4 Billion Connected Things Will be in Use 2017 | Gartner*. https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016. Feb. 2017 (zitiert auf S. 11).

[MFB+17] R. Masoni, F. Ferrise, M. Bordegoni, M. Gattullo, A. E. Uva, M. Fiorentino, E. Carrabba, M. Di Donato. „Supporting Remote Maintenance in Industry 4.0 through Augmented Reality“. In: *Procedia Manufacturing* 11 (2017). 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy, S. 1296–1302. ISSN: 2351-9789. DOI: 10.1016/j.promfg.2017.07.257. URL: https://www.sciencedirect.com/science/article/pii/S2351978917304651 (zitiert auf S. 31).

[MG+11] P. Mell, T. Grance et al. „The NIST definition of cloud computing“. In: (2011) (zitiert auf S. 13).

[MGS19] A. Mahroo, L. Greci, M. Sacco. „HoloHome: An Augmented Reality Framework to Manage the Smart Home“. In: *Augmented Reality, Virtual Reality, and Computer Graphics*. Hrsg. von L. T. De Paolis, P. Bourdot. Cham: Springer International Publishing, 2019, S. 137–145. ISBN: 978-3-030-25999-0 (zitiert auf S. 33).

[MHS14] S. Mayer, Y. N. Hassan, G. Sörös. „A Magic Lens for Revealing Device Interactions in Smart Environments“. In: *SIGGRAPH Asia 2014 Mobile Graphics and Interactive Applications*. SA '14. Shenzhen, China: Association for Computing Machinery, 2014. ISBN: 9781450318914. DOI: 10.1145/2669062.2669077 (zitiert auf S. 35).

[Mic] Microsoft. *HoloLens 2: Technische Daten und Funktionen – Microsoft HoloLens 2*. https://www.microsoft.com/de-de/d/hololens-2/91PNZZZNZWCP (zitiert auf S. 53).

[Mic21a] Microsoft. *HoloLens 2-Hardware*. https://docs.microsoft.com/de-de/hololens/hololens2-hardware. Okt. 2021 (zitiert auf S. 53).

[Mic21b] Microsoft. *HoloLens-Hardware (1. Generation)*. https://docs.microsoft.com/de-de/hololens/hololens1-hardware. Okt. 2021 (zitiert auf S. 53).

[Mic21c] Microsoft. *Nachverfolgen von QR-Codes - Mixed Reality | Microsoft Docs*. https://docs.microsoft.com/de-de/windows/mixed-reality/develop/platform-capabilities-and-apis/qr-code-tracking. Okt. 2021 (zitiert auf S. 64).

[MK94] P. Milgram, F. Kishino. „A taxonomy of mixed reality visual displays“. In: *IEICE TRANSACTIONS on Information and Systems* 77.12 (1994), S. 1321–1329 (zitiert auf S. 16, 17).

[MKBD20] A. Mishra, S. Karmakar, A. Bose, A. Dutta. „Design and development of IoT-based latency-optimized augmented reality framework in home automation and telemetry for smart lifestyle“. In: *Journal of Reliable Intelligent Environments* 6 (2020), S. 169–187 (zitiert auf S. 34).

[MPK+18] M. Makolkina, V. D. Pham, R. Kirichek, A. Gogol, A. Koucheryavy. „Interaction of AR and IoT Applications on the Basis of Hierarchical Cloud Services“. In: *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*. Hrsg. von O. Galinina, S. Andreev, S. Balandin, Y. Koucheryavy. Cham: Springer International Publishing, 2018, S. 547–559. ISBN: 978-3-030-01168-0 (zitiert auf S. 38).

[MSZP19] M. Minoufekr, P. Schug, P. Zenker, P. W. Plapper. „Modelling of CNC Machine Tools for Augmented Reality Assistance Applications using Microsoft Hololens“. In: *ICINCO (2)*. 2019, S. 627–636 (zitiert auf S. 29).

[MSZV19] D. Mourtzis, V. Samothrakis, V. Zogopoulos, E. Vlachou. „Warehouse design and operation using augmented reality technology: A papermaking industry case study“. In: *Procedia Cirp* 79 (2019), S. 574–579 (zitiert auf S. 29).

[MTUK95] P. Milgram, H. Takemura, A. Utsumi, F. Kishino. „Augmented reality: A class of displays on the reality-virtuality continuum“. In: *Telemanipulator and telepresence technologies*. Bd. 2351. International Society for Optics und Photonics. 1995, S. 282–292 (zitiert auf S. 16, 17).

[MZDG93] P. Milgram, S. Zhai, D. Drascic, J. Grodski. „Applications of augmented reality for human-robot communication“. In: *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93)*. Bd. 3. 1993, 1467–1472 vol.3. DOI: 10.1109/IROS.1993.583833 (zitiert auf S. 16).

[NKD11] A. Nigam, P. Kabra, P. Doke. „Augmented Reality in agriculture“. In: *2011 IEEE 7th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2011, S. 445–448. DOI: 10.1109/WiMOB.2011.6085361 (zitiert auf S. 26).

[NTBG15] X. T. Nguyen, H. T. Tran, H. Baraki, K. Geihs. „FRASAD: A framework for model-driven IoT Application Development“. In: *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. 2015, S. 387–392. DOI: 10.1109/WF-IoT.2015.7389085 (zitiert auf S. 40).

[OSK12] T. Oskiper, S. Samarasekera, R. Kumar. „Multi-sensor navigation algorithm using monocular camera, IMU and GPS for large scale augmented reality“. In: *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2012, S. 71–80. DOI: 10.1109/ISMAR.2012.6402541 (zitiert auf S. 21).

[Pae14] V. Paelke. „Augmented reality in the smart factory: Supporting workers in an industry 4.0. environment“. In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. 2014, S. 1–4. DOI: 10.1109/ETFA.2014.7005252 (zitiert auf S. 30).

[PCK+20] S. Park, H. Cha, J. Kwon, H. Kim, C. Im. „Development of an Online Home Appliance Control System Using Augmented Reality and an SSVEP-Based Brain-Computer Interface“. In: *2020 8th International Winter Conference on Brain-Computer Interface (BCI)*. 2020, S. 1–2. DOI: 10.1109/BCI48061.2020.9061633 (zitiert auf S. 33).

[PFP+17] R. Pierdicca, E. Frontoni, R. Pollini, M. Trani, L. Verdini. „The Use of Augmented Reality Glasses for the Application in Industry 4.0“. In: *Augmented Reality, Virtual Reality, and Computer Graphics*. Hrsg. von L. T. De Paolis, P. Bourdot, A. Mongelli. Cham: Springer International Publishing, 2017, S. 389–401. ISBN: 978-3-319-60922-5 (zitiert auf S. 9, 30).

[PKP+15] B. Pokrić, S. Krčo, M. Pokrić, P. Knežević, D. Jovanović. „Engaging citizen communities in smart cities using IoT, serious gaming and fast markerless Augmented Reality“. In: *2015 International Conference on Recent Advances in Internet of Things (RIoT)*. 2015, S. 1–6. DOI: 10.1109/RIOT.2015.7104905 (zitiert auf S. 28).

[PKP14] B. Pokrić, S. Krčo, M. Pokrić. „Augmented Reality Based Smart City Services Using Secure IoT Infrastructure“. In: *2014 28th International Conference on Advanced Information Networking and Applications Workshops*. 2014, S. 803–808. DOI: 10.1109/WAINA.2014.127 (zitiert auf S. 28).

[PN21] V. Ponnusamy, S. Natarajan. „Precision Agriculture Using Advanced Technology of IoT, Unmanned Aerial Vehicle, Augmented Reality, and Machine Learning“. In: *Smart Sensors for Industrial Internet of Things*. Springer, 2021, S. 207–229 (zitiert auf S. 25).

[Pon] J. Pontin. *ETC: Bill Joy's Six Webs | MIT Technology Review*. https://www.technologyreview.com/2005/09/29/230292/etc-bill-joys-six-webs/ (zitiert auf S. 11).

[PRLH19] Y. J. Park, H. Ro, N. K. Lee, T.-D. Han. „Deep-cARe: Projection-Based Home Care Augmented Reality System with Deep Learning for Elderly“. In: *Applied Sciences* 9.18 (2019). ISSN: 2076-3417. DOI: 10.3390/app9183897. URL: https://www.mdpi.com/2076-3417/9/18/3897 (zitiert auf S. 27).

[PSP+21] S. Prange, A. Shams, R. Piening, Y. Abdelrahman, F. Alt. „PriView– Exploring Visualisations to Support Users' Privacy Awareness“. In: CHI '21. Yokohama, Japan: Association for Computing Machinery, 2021. ISBN: 9781450380966. DOI: 10.1145/3411764.3445067 (zitiert auf S. 38).

[PT19] P. Phupattanasilp, S.-R. Tong. „Augmented Reality in the Integrative Internet of Things (AR-IoT): Application for Precision Farming“. In: *Sustainability* 11.9 (2019). ISSN: 2071-1050. DOI: 10.3390/su11092658. URL: https://www.mdpi.com/2071-1050/11/9/2658 (zitiert auf S. 25, 26).

[PTG+18] J. A. Purmaissur, P. Towakel, S. P. Guness, A. Seeam, X. A. Bellekens. „Augmented-Reality Computer-Vision Assisted Disaggregated Energy Monitoring and IoT Control Platform“. In: *2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC)*. 2018, S. 1–6. DOI: 10.1109/ICONIC.2018.8601199 (zitiert auf S. 35).

[Puy] E. Puybaret. *Sweet Home 3D : Galerie*. http://www.sweethome3d.com/de/gallery.jsp (zitiert auf S. 44).

[PYK19] Y. Park, S. Yun, K.-H. Kim. „When IoT Met Augmented Reality: Visualizing the Source of the Wireless Signal in AR View“. In: MobiSys '19. Seoul, Republic of Korea: Association for Computing Machinery, 2019, S. 117–129. ISBN: 9781450366618. DOI: 10.1145/3307334.3326079 (zitiert auf S. 39).

[RDP16] S. Russell, G. Dublon, J. A. Paradiso. „HearThere: Networked Sensory Prosthetics Through Auditory Augmented Reality“. In: *Proceedings of the 7th Augmented Human International Conference 2016*. AH '16. Geneva, Switzerland: Association for Computing Machinery, 2016. ISBN: 9781450336802. DOI: 10.1145/2875194.2875247 (zitiert auf S. 17).

[RTD+19] R. Revetria, F. Tonelli, L. Damiani, M. Demartini, F. Bisio, N. Peruzzo. „A Real-Time Mechanical Structures Monitoring System Based On Digital Twin, Iot and Augmented Reality“. In: *2019 Spring Simulation Conference (SpringSim)*. 2019, S. 1–10. DOI: 10.23919/SpringSim.2019.8732917 (zitiert auf S. 28).

[RU13] I. Rabbi, S. Ullah. „A survey on augmented reality challenges and tracking“. In: *Acta graphica: znanstveni časopis za tiskarstvo i grafičke komunikacije* 24.1-2 (2013), S. 29–46 (zitiert auf S. 21).

[Sal] Salesforce. *The History of Salesforce - Salesforce News*. https://www.salesforce.com/news/stories/the-history-of-salesforce/ (zitiert auf S. 13).

[SAR+19] Y. Sun, A. Armengol-Urpi, S. N. Reddy Kantareddy, J. Siegel, S. Sarma. „MagicHand: Interact with IoT Devices in Augmented Reality Environment“. In: *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 2019, S. 1738–1743. DOI: 10.1109/VR.2019.8798053 (zitiert auf S. 33).

[Sat17] M. Satyanarayanan. „The Emergence of Edge Computing“. In: *Computer* 50.1 (2017), S. 30–39. DOI: 10.1109/MC.2017.9 (zitiert auf S. 13).

[SBPI14] L. R. Suzuki, K. Brown, S. Pipes, J. Ibbotson. „Smart building management through augmented reality“. In: *2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS)*. 2014, S. 105–110. DOI: 10.1109/PerComW.2014.6815174 (zitiert auf S. 41).

[SCM15] S. Sarkar, S. Chatterjee, S. Misra. „Assessment of the Suitability of Fog Computing in the Context of Internet of Things“. In: *IEEE Transactions on Cloud Computing* 6.1 (2015), S. 46–59 (zitiert auf S. 14).

[SDMD21] G. R. Shinde, P. S. Dhotre, P. N. Mahalle, N. Dey. „IoT Use Cases“. In: *Internet of Things Integrated Augmented Reality*. Singapore: Springer Singapore, 2021, S. 19–33. ISBN: 978-981-15-6374-4. DOI: 10.1007/978-981-15-6374-4_2 (zitiert auf S. 27).

[SGA19] R. Seiger, M. Gohlke, U. Aßmann. „Augmented Reality-Based Process Modelling for the Internet of Things with HoloFlows“. In: *Enterprise, Business-Process and Information Systems Modeling*. Hrsg. von I. Reinhartz-Berger, J. Zdravkovic, J. Gulden, R. Schmidt. Cham: Springer International Publishing, 2019, S. 115–129. ISBN: 978-3-030-20618-5 (zitiert auf S. 9, 41).

[SHL12] H. Son, S. Han, D. Lee. „Contextual Information Provision on Augmented Reality with IoT-Based Semantic Communication“. In: *2012 International Symposium on Ubiquitous Virtual Reality*. 2012, S. 46–49. DOI: 10.1109/ISUVR.2012.22 (zitiert auf S. 40).

[SHS+20] A. C. F. da Silva, P. Hirmer, J. Schneider, S. Ulusal, M. T. Frigo. „Mbp: Not just an iot platform“. In: *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE. 2020, S. 1–3 (zitiert auf S. 15).

[SJ18] H. Subakti, J.-R. Jiang. „Indoor augmented reality using deep learning for industry 4.0 smart factories“. In: *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*. Bd. 2. IEEE. 2018, S. 63–68 (zitiert auf S. 29).

[SKKA21] R. Seiger, R. Kühn, M. Korzetz, U. Aßmann. „HoloFlows: modelling of processes for the Internet of Things in mixed reality“. In: *Software and Systems Modeling* (2021), S. 1–25 (zitiert auf S. 41).

[SM18] A. A. Simiscuka, G.-M. Muntean. „Synchronisation between real and virtual-world devices in a VR-IoT environment“. In: *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. IEEE. 2018, S. 1–5 (zitiert auf S. 38).

[SMC12] D. Schmidt, D. Molyneaux, X. Cao. „PICOntrol: Using a Handheld Projector for Direct Control of Physical Devices through Visible Light“. In: *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. UIST '12. Cambridge, Massachusetts, USA: Association for Computing Machinery, 2012, S. 379–388. ISBN: 9781450315807. DOI: 10.1145/2380116.2380166 (zitiert auf S. 21, 34).

[Sut68] I. E. Sutherland. „A head-mounted three dimensional display“. In: *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*. 1968, S. 757–764 (zitiert auf S. 18, 21).

[Tec] B. Technology. *Core Specification 4.0*. https://www.bluetooth.com/specifications/specs/core-specification-4-0/ (zitiert auf S. 12).

[TQZ+20] Z. Tan, H. Qu, J. Zhao, S. Zhou, W. Wang. „UAV-Aided Edge/Fog Computing in Smart IoT Community for Social Augmented Reality“. In: *IEEE Internet of Things Journal* 7.6 (2020), S. 4872–4884. DOI: 10.1109/JIOT.2020.2971325 (zitiert auf S. 39).

[TUKE19] S. Tanríseven, N. Uğur, A. Kanak, S. Ergün. „ERINOKS: EneRgy-Efficient INduction-Based Food Processing for Optimized KitchenS“. In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2019, S. 1–5. DOI: 10.1109/ISCAS.2019.8702571 (zitiert auf S. 37).

[UIAH12] A. M. Ullah, M. R. Islam, S. F. Aktar, S. K. A. Hossain. „Remote-touch: Augmented reality based marker tracking for smart home control“. In: *2012 15th International Conference on Computer and Information Technology (ICCIT)*. 2012, S. 473–477. DOI: 10.1109/ICCITechn.2012.6509774 (zitiert auf S. 34).

[UK18] O. Uviase, G. Kotonya. „IoT Architectural Framework: Connection and Integration Framework for IoT Systems“. In: *Electronic Proceedings in Theoretical Computer Science* 264 (Feb. 2018), S. 1–17. ISSN: 2075-2180. DOI: 10.4204/eptcs.264.1. URL: http://dx.doi.org/10.4204/EPTCS.264.1 (zitiert auf S. 40).

[UL17] I. Unwala, J. Lu. „IoT Protocols: Z-Wave and Thread“. In: *International Journal on Future Revolution in Computer Science & Communication Engineering* 3.11 (2017), S. 355–359 (zitiert auf S. 12).

[VLA+20] M. T. Vega, C. Liaskos, S. Abadal, E. Papapetrou, A. Jain, B. Mouhouche, G. Kalem, S. Ergüt, M. Mach, T. Sabol et al. „Immersive interconnected virtual and augmented reality: a 5G and IoT perspective“. In: *Journal of Network and Systems Management* 28.4 (2020), S. 796–826 (zitiert auf S. 14).

[WAV] D. WAVE. *Information capacity and versions of QR Code*. https://www.qrcode.com/en/about/version.html (zitiert auf S. 63).

[WAYO14] C. Withanage, R. Ashok, C. Yuen, K. Otto. „A comparison of the popular home automation technologies“. In: *2014 IEEE Innovative Smart Grid Technologies-Asia (ISGT ASIA)*. IEEE. 2014, S. 600–605 (zitiert auf S. 12, 14).

[WCPC19] G. White, C. Cabrera, A. Palade, S. Clarke. „Augmented Reality in IoT“. In: *Service-Oriented Computing – ICSOC 2018 Workshops*. Hrsg. von X. Liu, M. Mrissa, L. Zhang, D. Benslimane, A. Ghose, Z. Wang, A. Bucchiarone, W. Zhang, Y. Zou, Q. Yu. Cham: Springer International Publishing, 2019, S. 149–160 (zitiert auf S. 18, 37).

[Wei91] M. Weiser. „The Computer for the 21 st Century“. In: *Scientific American* 265.3 (1991), S. 94–105. ISSN: 00368733, 19467087. URL: http://www.jstor.org/stable/24938718 (zitiert auf S. 11).

[WRS+15] H. Wirtz, J. Rüth, M. Serror, T. Zimmermann, K. Wehrle. „Enabling ubiquitous interaction with smart things“. In: *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE. 2015, S. 256–264 (zitiert auf S. 9, 36).

[Xia] XiaomiProducts. *Xiaomi Mi Flower Care Plant Sensor*. https://www.xiaomiproducts.nl/de/xiaomi-mi-flower-care-plant-sensor.html (zitiert auf S. 43, 44).

[XLZH17] R. Xiao, G. Laput, Y. Zhang, C. Harrison. „Deus EM Machina: on-touch contextual functionality for smart IoT appliances“. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 2017, S. 4000–4008 (zitiert auf S. 21).

[XWW+21] M. Xia, Y. Wang, X. Wang, Z. Cheng, K. Chi. „INSIGHT: An AR-Enabled User Interface for Vision-Based Markerless Interaction with IoT Nodes“. In: *2021 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE. 2021, S. 1–7 (zitiert auf S. 22, 35).

[YFN+19] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J. P. Jue. „All one needs to know about fog computing and related edge computing paradigms: A complete survey“. In: *Journal of Systems Architecture* 98 (2019), S. 289–330 (zitiert auf S. 13).

[YLH+20] Z. Yusuf, V. Lukic, J. Heppelmann, C. Melrose, N. Ravi, U. Gill, A. Rosello. „Unleashing the Power of Data with IoT and Augmented Reality“. In: (2020) (zitiert auf S. 11, 32).

[ZCDE18] L. Zhang, S. Chen, H. Dong, A. El Saddik. „Visualizing Toronto City Data with HoloLens: Using Augmented Reality for a City Model“. In: *IEEE Consumer Electronics Magazine* 7.3 (2018), S. 73–80. DOI: 10.1109/MCE.2018.2797658 (zitiert auf S. 28).

[ZDB08] F. Zhou, H. B.-L. Duh, M. Billinghurst. „Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR“. In: *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*. IEEE. 2008, S. 193–202 (zitiert auf S. 17–19).

Alle URLs wurden zuletzt am 14. 10. 2021 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift