

Semesterprojekt, WS 23/24

B7 – XR-Interaktion eines haptischen Stadtmodells

Autor*in: Timo Boomgaarden, Yasmine Haidri, Jonas Mantay, Christian Wolter

Projektbetreuer: Alexander Kramer

Datum:
Berlin, 19.03.2024

Inhaltsverzeichnis

1	Grundidee	4
2	Entwicklung.....	6
2.1	Ideenentwicklung	6
2.1.1	Technikentscheidungen.....	6
2.1.2	Narrativ-Entwicklung.....	7
2.2	Prozess- und Technikentscheidungen.....	9
2.2.1	Interaktionen	9
2.2.2	Potentiometer als Datenquelle	10
2.2.3	NFC-Leser als Datenquelle.....	10
2.2.4	Arduino	10
2.2.5	W-Lan Störung.....	11
2.2.6	Kommunikation	12
2.2.7	RPI.....	15
2.2.8	Unity.....	15
2.2.9	Visuelle Umsetzung	17
2.3	Komponentenübersicht.....	19
2.3.1	Arduino	19
2.3.2	RPI.....	20
2.3.3	MQTT.....	20
2.3.4	Unity.....	21
2.3.5	Datenformat	21
3	Fazit der Projektteilnehmenden	23
3.1	Timo Boomgaarden	23
3.2	Yasmine Haidri.....	23
3.3	Jonas Mantay.....	24
3.4	Christian Wolter	24
4	Deployment-Sheet	26
4.1	Unity.....	26
4.1.1	Installation von Unity und Visual Studio:.....	26
4.1.2	Konfiguration in Unity und Visual Studio:.....	26

4.1.3	Freischalten des Developer-Modus auf der HoloLens 2:.....	27
4.2	RPI.....	27
4.2.1	Raspberry Pi 3 starten.....	27
4.2.2	Starten des WLAN-Hotspots.....	27
4.2.3	Starten von MQTT	28
4.2.4	Testen, ob der MQTT-Broker funktioniert.	29
4.2.5	Einrichten des Autostarts.....	29
4.2.6	Bekannte Probleme	29
4.3	Arduino.....	29
4.3.1	Anforderungen	30
4.3.2	Pin-Layout.....	30
4.3.3	W-Lan Einstellungen.....	31
4.3.4	MQTT-Einstellungen.....	31
5	Quellenverzeichnis.....	32

Abbildungsverzeichnis

Abbildung 1 Komponentenübersicht	7
Abbildung 2 Stadtmodell Übersicht	8
Abbildung 3 Kommunikationsübersicht	13
Abbildung 4 JSON-Objekt	14
Abbildung 5 Schaltgrafik.....	20
Abbildung 6: Visual Studio Konfiguration	27
Abbildung 7 Netzwerk-Einstellung am Arduino	31
Abbildung 8 MQTT-Broker Einstellungen am Arduino	31

Tabellenverzeichnis

Tabelle 1 Pin-Layout Taster	19
Tabelle 2 Pin-Layout Potentiometer	19
Tabelle 3 Pin-Layout LED	19
Tabelle 4 Pin-Layout NFC-Leser.....	20
Tabelle 5 Pin-Layout Taster	30
Tabelle 6 Pin-Layout Potentiometer	30
Tabelle 7 Pin-Layout LED	30
Tabelle 8 Pin-Layout NFC-Leser.....	30

1 Grundidee

Die Initiale Idee des Projekts „Entwicklung eines digital-haptischen Stadtmodells zur Informationsvermittlung mittels XR/AR“ kommt von der Forschungsgruppe Creative Media und baut auf einer vorherigen Arbeit der Forschungsgruppe Creative Media in Kooperation mit dem DVGW e.V. auf. In dieser Kooperation wurde eine Interaktive Forschungswelt entwickelt, welche ein Stadtmodell auf ein physisches Modell augmentiert. Hierbei wurden aber keine real-haptischen Interaktionen verwendet, welche mit diesem Projekt verwirklicht werden sollen.

Im Rahmen des Semesterprojekts soll ein Prototyp eines digital-haptischen Stadtmodell, welches ungefähr 1 x 1m groß ist, entwickelt werden, welche Informationen vermittelt. Ein digital-haptisches Stadtmodell ist ein Modell einer Stadt, welche zum einen real existiert, aber auch einen digitalen Zwilling besitzt, mit dessen die Informationen angezeigt und vermittelt werden. Hierfür soll eine XR/AR-Applikation entwickelt werden, welche auf der HoloLens 2 läuft.

Zu den Anforderungen des Projekts gehören die Entwicklung eines Narratives für die Informationsvermittlung, die Erstellung eines digitalen Zwillings des Stadtmodells, welches auf das reale Stadtmodell mit Hilfe von XR-Hardware augmentiert wird und die Entwicklung eines Systems zur digitalen Interaktion. Zudem sollen augmentierte Point-of-Interests (POI) implementiert werden. Die Software auf Seiten der XR/AR-Applikation soll unter Verwendung von Unity und dem AR-Framework MRTK für die HoloLens entwickelt werden.

Hierbei soll sowohl das real-haptische, sowie das digitale Stadtmodell um modulare Fähigkeiten erweitert werden. Hierzu gibt die Forschungsgruppe Create Media folgende Beispiele an:

- Einbau eines NFC-Lesers, um die Augmentierung durch Austausch eines Elementes des haptischen Modells anzupassen
- Ansteuerung und Auslesen eines oder mehrerer haptischer Drehregler, um die digitale Informationspräsentation zu beeinflussen (z.B. dargestellte Zeitebene)
- Haptische Taster, um entweder die Informationsebene der digitalen Präsentation zu beeinflussen (Wechsel Tag-Nachtsicht) oder digital augmentierte Situationsbedingte Umfrage zur gerade dargestellten digitalen Präsentation
- Darstellung und Manipulation von POI-basierten 3D-Darstellungen (über einen ausgewählten POI erscheint als Erklärung z.B. eine 3D-Maschinen-Darstellung, mit der getrennt interagiert werden kann)
- Erkennung eines aus dem Modell entnommenen Teilobjektes (z.B. ein Haus) und anschließender Augmentierung dieses Teilobjektes in der Hand (Anzeige von Zusatzinformationen, einfache Interaktionen etc.)
- Rückspielen von digitalen Interaktionen auf das real-haptische Modell (z.B. Anschalten von Lichteffekten bei der Auswahl eines digitalen POIs).

Entsprechende Hardware für das Projekt sowie Räumlichkeiten für die Entwicklung werden von der Forschungsgruppe Creative Media für die Entwicklung des Projekts bereitgestellt. Auch das physische Modell sowie die Realisierung, der Erweiterungen, wie Taster, Schalter, NFC-Regler werden von der Forschungsgruppe Creative Media übernommen.

2 Entwicklung

2.1 Ideenentwicklung

Nach der initialen Einführung in das Projekt durch Herrn Kramer konnten wir mit der Projektplanung und der Ideenentwicklung starten. Hierzu mussten wir uns zu verschiedenen Themen Gedanken für die Umsetzung machen. Zum einen mussten wir herausfinden, welches Narrativ wir für unser Stadtmodell haben wollen, welche haptischen und digitalen Interaktionen wir verwenden wollen und welche Hardware wir in den entsprechenden Teilbereichen verwenden. Ein weiterer großer Punkt war die Organisation an sich. Hierbei hat sich die Frage gestellt, wie wir unser Projekt managen wollen.

Zum Thema Projektmanagement haben wir direkt von Anfang festgelegt, dass wir mit der Kanban Methode arbeiten und wöchentliche Absprachen mit Herrn Kramer, welcher unser Projektbetreuer ist, halten werden.

2.1.1 Technikentscheidungen

Auf Seiten der Technik mussten wir zum einen Überlegen welche Hardware wir verwenden wollen und auch gucken, wie die Hardware miteinander kommuniziert.

Hierbei gab es zwei Möglichkeiten. Die Hardware auf Seite des physischen Stadtmodell direkt mit der Hardware auf der Seite des digitalen Stadtmodell kommunizieren lassen, oder einen Server in der Mitte zu verwenden.

Die Direktverbindung hätte man mit Hilfe von Websockets lösen können. Hierbei besteht aber das Problem, dass diese Verbindung State-basiert ist. Sobald die Verbindung abbricht, müsste sie neu aufgebaut werden. Mit dieser Methode ist es auch komplizierter mehrere Clients gleichzeitig zu verwenden, was in unserem Projekt aber der Fall sein soll.

Die Verbindung über einen Server laufen zu lassen ist also die einfachere Methode, aber auch hier gibt es mehrere Möglichkeiten, wie man dies lösen kann. Hierzu hatten wir uns überlegt die Daten an einen Webserver zu schicken und die Daten von diesem wieder abzugreifen oder die Verwendung eines Kommunikationsprotokoll, wie z.B. MQTT¹.

Wir haben uns an dieser Stelle für eine Übertragung mittels des MQTT-Protokolls entschieden und um die Modularität des Projekts zu wahren auch für drei Hardware-Gruppen, die jeweils eine andere Aufgabe haben.

Auf Seiten des physischen Modells soll ein Arduino verbaut werden, als Server fungiert ein Raspberry Pi und der digitale Zwilling wird mit Hilfe einer HoloLens augmentiert. Warum wir uns für diese Techniken entschieden haben, wird in den folgenden Abschnitten erklärt.

¹ MQTT - The Standard for IoT Messaging: o. D., [online] <https://mqtt.org/>.

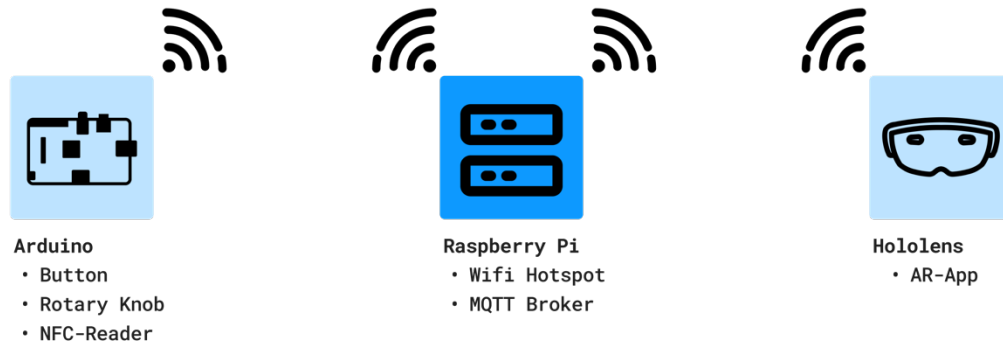


Abbildung 1 Komponentenübersicht

2.1.2 Narrativ-Entwicklung

Während wir bei den Technikentscheidungen schnell gewusst haben, was wir brauchen und wie wir gewissen Elemente Umsetzen wollen, ging die Entwicklung des Narratives sehr lange und hat sich bis kurz zum Ende gestreckt.

Wir wussten zwar, dass unser Stadtmodell und das Narrativ etwas mit dem Thema Umwelt zu tun haben soll, wie dieses aber genau aussieht wusste niemand so genau und hat auch einiges an Entwicklungszeit gekostet.

Die Entwicklung des aktuellen Narratives hat sich über mehrere Wochen gestreckt und hatte viele Zwischenschritte. Wir haben an dieser Stelle mit mehreren Ideen gearbeitet und diese zusammengeführt. Folgende Ideen wurden hierzu verwendet:

- Herausforderung von Städten verursacht durch den Klimawandel
- Winter/Sommer Energiemodell
- Ein Spielkonzept in dem durch Interaktionen die Stadt wieder Grün gemacht werden kann
- Darstellung des Energiemix.

Hieraus wurde dann das Narrativ herausgearbeitet, welches die Auswirkung des Energiemix auf die Umwelt darstellt und auch anzeigt welche Alternativen Technologien es gibt.

Um eine bessere Übersicht über die Auswirkungen auf die Stadt und die verwendeten Technologien zu erhalten haben wir die Stadt in vier Sektoren aufgeteilt. Einen Wohnsektor, welcher anzeigt wie sich wohnen auf die Umwelt auswirkt und welche Technologien man verwenden kann, um seinen Treibhausgas Ausstoß zu reduzieren. Die Visualisierung soll hierbei durch das Austauschen eines Wohnhauses stattfinden. Hierzu haben wir zwei Modelle, eines welche ein klimafreundliches Haus darstellt und eins, welches ein altes Haus, welches viel Treibhausgas freilässt darstellt.

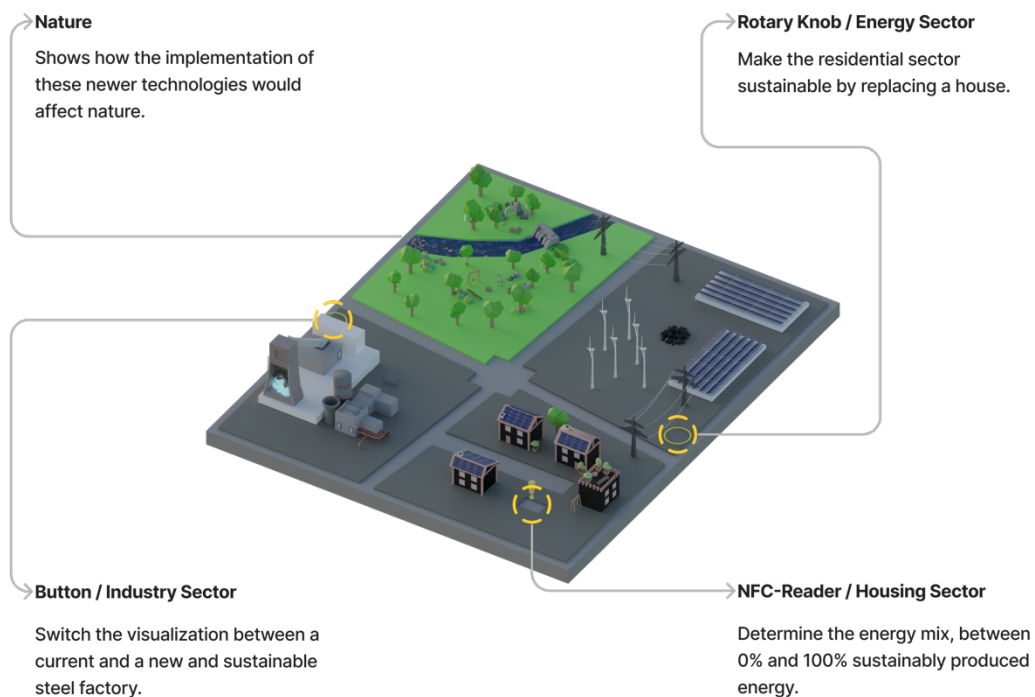


Abbildung 2 Stadtmodell Übersicht

Einen Industriesektor, welcher durch ein Stahlwerk dargestellt wird. Dieses Stahlwerk soll per Tastendruck mit einem Klimafreundlichen Stahlwerk ausgetauscht werden, welches mit Wasserstoff statt mit Kohle läuft. Was in unserem Modell so einfach funktioniert, wird aktuell noch getestet, wobei aktuell schon erste Stahlwerke mit dieser Technologie gebaut werden.

Einen Energiesektor, bei dem man den Energiemix zwischen 0% und 100% Erneuerbaren Energie wählen kann. 0% erneuerbare Energie wird hierbei mit zwei Kohlekraftwerke dargestellt, während 100% erneuerbare Energie mit Solaranlagen und Windkraftanlagen dargestellt werden. Der Übergang zwischen den beiden Zuständen soll flüssig dargestellt werden, das heißt das Elemente erst kleiner werden bevor sie verschwinden und für jedes verschwundene Element soll der Gegenpart erscheinen. Wenn man also langsam von 0% auf 100% dreht, werden die Kraftwerke nacheinander immer kleiner, bis sie verschwinden und wenn sie verschwunden sind, dann erscheinen neue Kraftwerke der gegenteiligen Energiequelle. Das Steuern des Energiemix soll mit einem Drehregler², stattfinden.

Der letzte Sektor ist eine Freifläche und stellt ein kleines Naturstück da. Hier werden, je grüner die Stadt ist mehr Bäume angezeigt. Das heißt das eine Veränderung in den anderen Sektoren auch eine Auswirkung auf die Begrünung auf der Freifläche hat.

² technisch: ein einstellbarer Widerstand

Unter jedem Sektor soll eine Informationstafel angebracht werden, die je nach Status in dem jeweiligen Segment dazugehörige Informationen darstellt. Auf der Freifläche sollen darüber hinaus Informationen über das Projekt angezeigt werden. Diese Informationen werden nicht durch haptische Interaktion, sondern durch virtuelles Drücken auf einzelne Reiter verändert.

2.2 Prozess- und Technikentscheidungen

2.2.1 Interaktionen

Während das Narrativ einen sehr langen Entwicklungsprozess hinter sich her zog, hatten wir von Anfang an eine Idee, welche Interaktionen wir einbauen wollen.

Vor dem Start der Entwicklung hatten wir eine größere Auswahl an Interaktionen zur Auswahl. Darunter waren Schalter, Drehregler, POI-basierte 3D-Darstellungen, erkennen von Objekten im Model und das Rückspielen von digitalen Interaktionen auf das reale Model.

Die Interaktionen, für die wir uns entschieden haben, werden ausschließlich haptisch durch einen Taster, ein Potentiometer und einem NFC-Leser ausgelöst. Des weiteren wurde eine LED als weiteres elektronisches Bauteil eingebaut, welches als Status LED für die W-Lan Verbindung fungiert.

Unsere Entscheidung für die Interaktionen ist mit Angesicht der Zeit und unseren geringen Vorkenntnissen in der Entwicklung von XR-Applikationen und Mikrocontrollern auf rein haptische Interaktionen gefallen, die aber eine gewisse Variabilität in den gesendeten Daten vorweisen. Die Interaktionen werden durch einen Taster, ein Potentiometer und einem NFC-Leser ausgelöst und diese eingebauten Interaktionen stellen unterschiedliche Daten zur Verfügung. Auch müssen alle elektronischen Bauteile unterschiedlich angeschlossen und programmiert werden.

Wir haben uns aber auch die Option offen gehalten digitale Interaktionen auf das reale Model zurückzuspielen. Hier war die Idee zum einen Lichter in den Häusern anzuschalten und die Drehgeschwindigkeit der Rotoren eines Windrads zu steuern. Aus zeitlichen Gründen wurden diese Interaktionen aber nicht mit eingebaut.

Während der Taster eine Bestätigung an den Arduino sendet, dass dieser betätigt wurde, werden die Werte des Potentiometers beim Durchlaufen der loop-methode³ abgefragt. Diese liegen im Bereich von 0 bis 1023 und der NFC-Leser sendet die 7 Byte lange UID der NFC-Karten.

Der NFC-Leser könnte neben der UID auch die Daten, die auf der Karte geschrieben sind, senden, hierzu müssten dann aber die Daten vor der Verwendung auf die NFC-Karten geschrieben werden.

³ oop() - Arduino-Referenz: o. D., [online]
<https://www.arduino.cc/reference/de/language/structure/sketch/loop/>.

Die Dual-Color LED wurde nachträglich eingebaut, um den aktuellen W-Lan Status anzuzeigen, da wir ansonsten keine Anzeige haben, welche uns anzeigt, ob aktuell eine W-Lan Verbindung besteht oder nicht.

2.2.2 Potentiometer als Datenquelle

Im Laufe der Entwicklung ist uns sehr schnell aufgefallen, dass wir bei dem Potenziometer eine Schwelle einbauen müssen ab wann dieser Daten sendet. Durch minimale Spannungsschwankungen am Potentiometer selbst hat dieser mehrere Datensätze pro Sekunde gesendet, welche nur minimal unterschiedlich sind. Hierbei haben wir nach zahlreichen Tests uns auf einen Wert von 25 festgelegt, was ungefähr 2,5% entspricht. Dieser Wert ist zwar hoch, aber nur durch diesen hohen Wert konnten wir den Großteil der Schwankungen abfangen.

Während der Fehler beim Potenziometer früh gefunden wurde, hatten wir gegen Ende wiederum Probleme mit dem Taster, dieser wurde ausgetauscht, da unser Entwicklungs-Taster nicht auf das Physische Stadtmodell gepasst hat. Der erste neue Taster hat hierbei nur teilweise funktioniert. Der zweite Taster, welcher eingebaut wurde, hat zwar besser funktioniert, war aber nicht 100% funktionsfähig, wodurch die Interaktion hin und wieder nicht oder anders funktioniert hat.

2.2.3 NFC-Leser als Datenquelle

Den NFC-Leser mussten wir auch austauschen. Wir hatten zu Beginn der Entwicklung einen NFC-Leser vom Typ PN522. Dieser hat aber nach mehreren Anläufen in der Entwicklung nicht funktioniert und hat uns keine Daten zurückgeliefert. Als Austausch kam dann ein PN532 NFC-Leser. Der Einbau und das Programmieren des neuen NFC-Lesers verliefen trotzdem nicht reibungslos. Wir hatten anfangs keine Dokumentation zum NFC-Lesers, was die Verkabelung verkompliziert hatte. Auch mussten wir uns durch mehrere Code-Libraries durchtesten, bis wir eine gefunden hatten, welche mit dem neuen NFC-Leser funktionierte.

Der NFC-Leser kommuniziert über das I2C-Protokoll mit dem Arduino. I2C ist außerdem ein Bus-System, mit dem sich mehrere Datenquellen an einen Arduino anschließen lassen⁴. Dieses Protokoll ist gegenüber dem UART- und SPI-Protokoll einfach aufgebaut und ideal für die Verwendung an Mikrocontrollern, da es nur eine Schnittstelle mit zwei Kabeln verwendet, wodurch die Verkabelung vereinfacht wird und die Komplexität der Schaltung relativ gering bleibt. I2C ist außerdem ein sehr zuverlässiges Protokoll.

2.2.4 Arduino

Auf der Seite des physischen Stadtmodells hatten wir mehrere Optionen, um die haptischen Interaktionen, welche im Vorherigen Abschnitt beschrieben wurden, zu verarbeiten und

⁴ Zambetti, Nicholas/Karl Söderby/Jacob Hylén: Inter-Integrated Circuit (I2C) Protocol, in: Arduino Documentation, 16.01.2024, [online] <https://docs.arduino.cc/learn/communication/wire/> (abgerufen am 18.03.2024).

weiterzuleiten. Alle Optionen sind Microcontroller, also Ein-Chip-Computersysteme, die einfache Aufgaben erledigen können.

Zur Auswahl standen ein Arduino, ein Raspberry Pi und ein ESP32, wobei unsere Auswahl am Schluss auf den Arduino gefallen ist, welcher die haptischen Interaktionen verarbeitet und weiterleitet.

Der Arduino war für uns die perfekte Wahl, da er genügend Leistung für die zu erledigten Aufgaben hat, kostengünstig ist, eine große Anzahl an Zubehör, Bibliotheken, Beispielcode und Anleitungen hat und in unserem Fall auch die richtige Konnektivität hat.

Wir finden, dass ein Raspberry Pi für unserer Aufgaben zu Leistungsstark und auch zu teuer ist und ein ESP32 zwar in der Preis- und Performance Kategorie dem Arduino überlegen ist, aber die Bibliothek an Zubehör und Anleitungen zu klein war, weswegen der Arduino die beste Option ist. Dennoch wird ein Raspberry Pi in unserem Projekt eingesetzt. Dieser fungiert als Kommunikationszentrale und W-Lan Access-Point.

Ein Arduino ist eine quelloffene Physical-Computing-Plattform, welcher aus einer Platine mit Mikrocontroller-Chip und Pins besteht. Pins können wahlweise als Inputs oder Outputs mit verschiedenen Funktionen belegt werden. Hierdurch können an einem Arduino verschiedenste Sensoren, Motoren und andere elektronische Bauteile angeschlossen und programmiert werden, um bestimmte Aufgaben auszuführen.

Bei unserem Projekt kam ein Arduino Uno Wifi Rev2 zum Einsatz. Dieser Arduino hat den Vorteil, dass er ein Netzwerk Modul bereits eingebaut hat, wodurch wir dieses nicht extra anbauen und programmieren mussten, was bei anderen Arduino-Varianten der Fall ist. Des Weiteren bietet der Arduino Uno uns genügend Anschlüsse, um alle elektronischen Bauteile anschließen zu können und es gibt auch alle benötigten Teile mit entsprechenden Software-Bibliotheken passend für den Arduino. Ein weiterer Grund für den Arduino Uno Wifi Rev2 war die schnelle Verfügbarkeit. Dieser war im Lager der Forschungsgruppe Creative Media vorhanden und wir konnten diesen sofort mitnehmen und mit dem Anschließen und Programmieren der elektronischen Bauteile Beginnen.

2.2.5 W-Lan Störung

Anfangs wurden alle Bauteile in der Loop nacheinander abgefragt, was aber mit dem Einbau des NFC-Lesers zu Problemen geführt hat. Dieser benutzt nämlich die Interrupt Funktion des Arduinos, wodurch alle Abläufe abgebrochen wurde, sobald der NFC-Leser einen neuen NFC-Tag erkannt hat. Dieses Problem konnten wir beheben, in dem wir zum einen den NFC-Leser nur alle 5 Sekunden abfragen und auch den Taster über die Interrupt Methode⁵ einbinden. Eine Interrupt Methode ist ein Hardware-nahes Programmsignal, welches auf Hardware Ein- und Ausgaben reagiert und eine vordefinierte Funktion auslöst, während ein

⁵ AttachInterrupt() - Arduino-Referenz: o. D., [online]
<https://www.arduino.cc/reference/de/language/functions/external-interrupts/attachinterrupt/>.

anderes Programm abgearbeitet wird⁶. Der Vorteil der Interrupt Methode ist, dass wir hierdurch die Bauteile nicht jedes Mal abfragen müssen, sondern die dazugehörigen Methoden erst dann abgerufen werden, sobald eine Eingabe von dem entsprechenden Bauteil kommt.

Der NFC-Leser hat hierbei noch eine Sonderstellung gehabt. Dessen Interrupt Methode mussten wir eingrenzen, da ansonsten unser gesamtes System blockiert wurde. Diese Interrupt Methode wird nur alle fünf Sekunden abgefragt, wodurch die Blockade des Systems umgangen wird. Der NFC-Leser hat aber nicht nur beim Abrufen der Daten Probleme verursacht. Auch hat dieser ein Problem mit dem Netzwerkmodul. Während der Verwendung des NFC-Lesers kommt es immer wieder zu Ausfällen in der W-Lan Verbindung, wodurch keine Daten mehr vom Arduino an die HoloLens gesendet werden können.

Dieses Netzwerkproblem konnten wir zum Teil in den Griff bekommen, indem vor jedem Sendevorgang geprüft wird, ob eine W-Lan Verbindung besteht und wenn dies nicht der Fall ist, dann wird probiert, die Verbindung wieder herzustellen. Wir konnten während der Entwicklung herausfinden, dass einige NFC-Tags das W-Lan Problem auslösen und andere wiederum nicht. Warum das genau der Fall ist, dafür haben wir bis zum Schluss leider keine Antwort oder eine Lösung gefunden.

Das Problem konnte durch die Abfrage des Netzwerkstatus weitestgehend eingegrenzt werden. Hierbei handelt es sich aber um einen Fix, welcher nur die Symptome bekämpft.

Bei der Überlegung, ob wir die Daten des NFC-Tags oder die ID des NFC-Tags zur Identifizierung des Objekts benutzen, haben wir uns auf Grund der Einfachheit für die Abfrage der ID entschieden.

Hierdurch sparen wir uns eine weitgehende Abfrage des NFC-Tags auf Seiten des Arduinos und wir müssen hierdurch die NFC-Karten nicht beschreiben, sondern lediglich die ID des Tags auslesen.

2.2.6 Kommunikation

Eine der wichtigsten Fragen, die wir uns am Anfang des Projekts gestellt haben, war, wie wir die Daten vom Arduino an die HoloLens übertragen. Da beide Systeme unabhängig voneinander funktionieren musste hier eine Kommunikation erstellt werden, mit der Daten vom Arduino an die HoloLens und auch Daten von der HoloLens an den Arduino gesendet werden können. Auch hierzu gab es verschiedene Möglichkeiten, wie wir das Problem lösen können.

Eine Bedingung für diese Kommunikation war, dass diese Drahtlos ist. Wir wollten keine Kabel an die HoloLens anschließen, da mit einer Kabelverbindung die Nutzererfahrung am Stadtmodell beeinträchtigt werden würde. Mit einer Drahtlos-Verbindung ist es einfacher sich

⁶ Robin: Interrupts (Assembler) – informatik abitur, o. D., [online] <https://informatik-abitur.de/assembler/interrupts/82/>.

um ein physisches Stadtmodell zu bewegen, während deine Kabelverbindung dieses freie Umherlaufen stark beeinträchtigt.

Wir sind durch Recherche schnell auf zwei Methoden gestoßen, die für unser Projekt funktionieren würden. Die Verwendung von Websockets, wobei wir diese Idee schnell wieder verworfen haben, und MQTT.

Die Entscheidung für MQTT als Übertragungsprotokoll hat viele Vorteile für uns gegenüber der Verwendung von Websockets. Zum einen kann ein Verbindungsabbruch leichter abgefangen und wieder aufgebaut werden, zum anderen ermöglicht uns MQTT mehrere Geräte auf beiden Seiten einzubinden und hierbei auch unterschiedliche Geräte zu verwenden und es ist sehr modular, man kann also Geräte entfernen und wieder hinzufügen, ohne dass man an einer Stelle Probleme bekommt.

MQTT ist ein leichtgewichtiges Übertragungsprotokoll, das auf einem Publish/Subscribe Modell basiert und die Übertragung von Nachrichten zwischen Geräten im gleichen Netzwerk ermöglicht. Hierzu werden drei Services benötigt. Demnach gibt es einen Publisher, der Daten zu einem bestimmten Thema (Topic) sendet. Weiterhin gibt es einen Subscriber, der an Daten, welches an ein Thema gesendet wird, interessiert ist und dieses Topic beim Broker abonniert und einen Broker, der als zentraler Vermittler in der Kommunikation zwischen dem Publishern und den Subscribern fungiert und die Nachrichten entsprechend weiterleitet. Themen, bzw. Topics sind dabei die Kanäle auf denen Nachrichten gesendet werden.

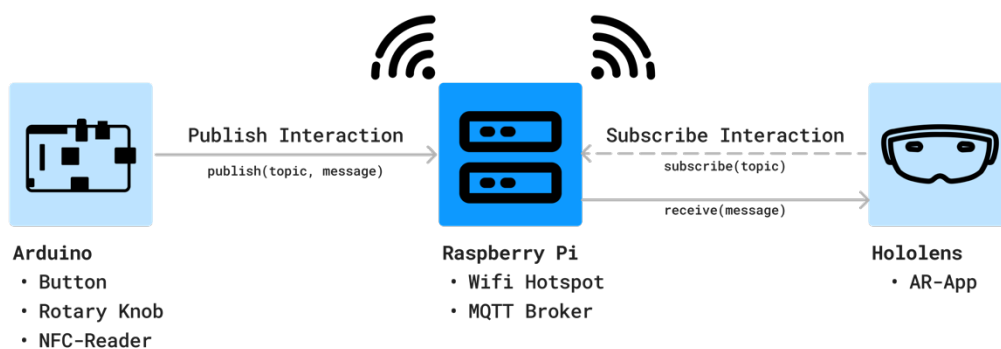


Abbildung 3 Kommunikationsübersicht

Da sowohl der Arduino, also auch die HoloLens MQTT unterstützt und es hierzu auch entsprechende Bibliotheken^{7,8} gibt war dies die beste Wahl für uns.

Nachdem wir uns auf ein Übertragungsprotokoll geeinigt haben, mussten wir uns noch überlegen was für ein Datenformat wir verwenden, um unsere Daten zu senden. MQTT lässt uns nur eine Nachricht mitsenden und es müssen mehrere Datensätze pro Nachricht versendet werden.

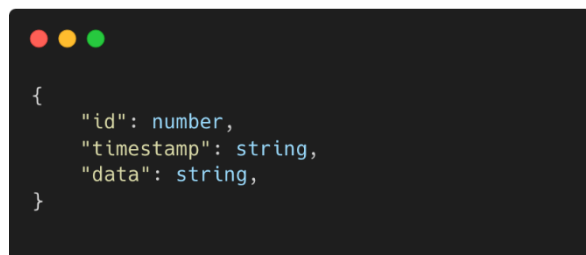
Bei der Auswahl des Datenformats hatten wir einige Möglichkeiten, wobei die zwei gängigsten Formate zum einen JSON und zum anderen XML sind. Wir haben uns hier für JSON entschieden, da dieses effizienter und einfach gegenüber XML ist.

Auf dem Arduino haben wir nur bedingt Ressourcen zur Verfügung, wodurch das Datenformat kompakt sein muss, um möglichst wenig Speicherplatz zu verbrauchen. Des Weiteren lässt sich JSON gegenüber XML einfacher generieren und auch parsen, was unsere Codebasis vereinfacht und JSON ist auch ziemlich weit verbreitet und wird von vielen Programmiersprachen, sowie Bibliotheken unterstützt. So auch vom Arduino.

Mit der Wahl des Datenformats konnten wir nun beginnen die Datenstruktur auszuarbeiten. Dabei haben wir uns als erstes angeguckt was die einzelnen elektronischen Bauteile für Daten liefern und anschließend diese in die Datenstruktur übertragen.

Heraus kam ein JSON-Objekt mit drei Key-Value-Paaren. Einer ID, einem Zeitstempel und den Daten der Interaktion.

Die ID hat hierbei einen Wert zwischen eins und drei und dient uns dazu die Interaktion zu definieren. Eins ist eine Taster-Interaktion, zwei eine Potenziometer-Interaktion und drei eine NFC-Leser Interaktion.

A screenshot of a code editor with a dark background and light-colored text. It shows a JSON object with three key-value pairs: "id" with the value "number", "timestamp" with the value "string", and "data" with the value "string". The object is enclosed in curly braces and the values are in quotes.

```
{  
  "id": number,  
  "timestamp": string,  
  "data": string,  
}
```

Abbildung 4 JSON-Objekt

Der Zeitstempel hat Anfangs von der genauen Uhrzeit gesendet, wir mussten das aber umschreiben und schicken jetzt die vergangenen Millisekunden seit Systemstart mit. Diesen Schritt mussten wir gehen, da der Arduino keine eingebaute Uhr hat und wir die aktuelle Uhrzeit aus dem Netzwerk abrufen mussten. Das funktionierte auch im Testbetrieb, da hier unser Netzwerk auch eine Verbindung ins Internet hat. Beim finalen System ist dies aber nicht vorgesehen gewesen, wodurch wir zwar eine Netzwerkverbindung hatten, aber kein

⁷ Söderby, Karl: Sending Data over MQTT, in: Arduino Documentation, 30.01.2024, [online] <https://docs.arduino.cc/tutorials/uno-wifi-rev2/uno-wifi-r2-mqtt-device-to-device/> (abgerufen am 18.03.2024).

⁸ TheDarkKnight: How to use MQTT in HoloLens?, in: Mixed Reality Developer Forum, 18.04.2017, [online] <https://forums.hololens.com/discussion/7222/how-to-use-mqtt-in-hololens>.

Zugriff aufs Internet. Somit konnten wir auch nicht das aktuelle Datum und die aktuelle Uhrzeit abrufen.

Die Daten werden als String mitgeschickt und behalten die Daten der Interaktion. Beim Taster ist dies ein leer String. Beim Potenziometer ein Wert zwischen 0 und 1023 und beim NFC-Leser die ID des NFC-Tags.

Alle Daten werden als Rohdaten versendet. Auf dem Arduino selbst findet keine Verarbeitung der Daten statt.

2.2.7 RPI

Um eine stabile Kommunikation zu ermöglichen, entschieden wir uns dazu einen lokalen WLAN-Hotspot auf einem Raspberry Pi 3 zu errichten, mit dem sich sowohl der Arduino, als auch die Hololens verbinden. Nach kurzem Einarbeiten in das Betriebssystem haben wir diesen vorerst anhand von dem Standard Raspberry Pi Os Netzwerkprotokoll „dhcpcd“ mithilfe der weiteren Pakete „hostapd“ und „dnsmasq“ eingerichtet. Dies funktionierte anfangs sehr gut, führte dann jedoch zu vermehrten Fehlern bei dem Versuch gleichzeitig den MQTT-Broker auf demselben System zu starten. Deshalb entschieden wir uns für einen anderen Ansatz, bei dem der Hotspot durch den „network-manager“ gehostet wird, eine Alternative zu „dhcpcd“, bei welchem diese Problematik nicht auftrat.

Neben der Tatsache, dass der Raspberry Pi 3 das WLAN-Netzwerk hostet, läuft auf dem RPI zusätzlich kontinuierlich ein MQTT-Broker. Das MQTT-Protokoll eignet sich in diesem Fall besonders gut, da wir aufgrund der Publish/Subscribe-Architektur mit mehreren Geräten unabhängig und effizient Daten versenden und empfangen können. Bezüglich der „Quality of Service“ entschieden wir uns für die Variante „At Most Once“, sodass jede Nachricht nur höchstens einmal gesendet wird. Das führt dazu, dass bei Verbindungsabbruch keine verlorenen Nachrichten erneut gesendet werden, da dies in unserer Anwendung als nicht sinnvoll erschien. Dies würde bei dauerhafter Bedienung nur zu Aufstauen von Nachrichten führen, welche sowieso nur Sinn für den Nutzer ergeben, wenn sie in Realtime ankommen.

2.2.8 Unity

Um eine Applikation für die HoloLens 2 zu entwickeln, entschieden wir uns für die von Microsoft empfohlene Engine Unity.⁹

Am Anfang des Projekts haben wir noch mit MacOS als Betriebssystem entwickelt. Da wir hier lediglich mit dem AR-Core Framework von Google gearbeitet haben. Da die für die Android- und IOS-Entwicklung verwendet wird, hat die Entwicklung unter MacOS auch funktioniert. Nachdem wir erste Test-Apps zum Platzieren von Elementen in einem Raum und verändern dieser Objekte geschrieben haben, kamen wir aber mit der Entwicklung unter MacOS an unsere Grenzen. Für die Entwicklung auf der HoloLens mussten wir deshalb

⁹ Microsoft, „Auswählen Ihrer Engine“, 12.07.2023, Link: <https://learn.microsoft.com/de-de/windows/mixed-reality/develop/choosing-an-engine?tabs=unity> (Stand 15.03.2024)

später auf Windows als Entwicklungsbetriebssystem wechseln, da wir nur mit diesem die Möglichkeit hatten für die HoloLens zu bauen und auch auf die HoloLens zu deployen.

Um uns mit Unity vertraut zu machen fingen wir an eine simple AR-App für ein Android Tablett zu entwickeln. In diese Vorab-Version hatte schon alle Grundideen der Interaktionen eingebaut, nur eben anhand von Cubes.

Zeitgleich haben wir die MQTT-Verbindung auf Unity-Seite eingebaut. Zu der Zeit war die RP noch nicht konfiguriert, weshalb wir vorerst versuchten, die MQTT-Verbindung in Unity lokal mit einem Mosquitto Broker herzustellen. Für Unity fanden wir die Library M2MQTT und konnten anhand dieser die 'MQTT Communication' Klasse programmieren, die einen Client erzeugt, sich zum Broker mit localhost als IP-Adresse verbindet und zum Topic abonniert, um dann den erhaltenen JSON-File rauszuparsen und Loggers für die verschiedene IDs in der Konsole zu drücken. Der lokale Broker auf dem Rechner funktionierte am Anfang nicht und wir mussten eine andere Lösung finden.

Es gab verschiedene Wege einen MQTT-Broker zu emulieren und Nachrichten zu senden. Zum Beispiel schrieben wir ein Python Skript das mithilfe der Python MQTT Publish Library "paho.mqtt.publish" die Nachricht am Mosquitto Broker schickt. Das war gut zum Testen der Interaktionen, da es ebenso als Mittel benutzt werden konnte, um die Zusendung von Nachrichten zu automatisieren, z.B. um einen Drehregler zu simulieren. Mit online Brokers bzw. Hive MQ haben wir auch experimentiert und damit konnten wir erfolgreich Nachrichten empfangen, was die Logik bestätigt hat, die wir in Unity aufgebaut haben. Dann haben wir unsere Mosquitto Konfiguration korrigiert und damit weitergearbeitet, bis die RP-Konfiguration erfolgreich war und wir die Broker IP Adresse mit dem RP-Adresse austauschen konnten.

Für die Grundideen haben wir Cubes in die Scene gesetzt, bei einem von denen wir durch den Knopfdruck die Farbe ändern konnten. Zudem konnten wir anhand eines Float-Values in der JSON-Message, die wir über das MQTT-Protokoll gesendet haben, einen anderen Cube rotieren lassen. Und als letzte noch vereinfachte Aktion, generierten wir einen neuen Cube anhand einer erkannten NFC-ID.

Wir implementierten außerdem QR-Code-Erkennung, wodurch ein 3D-Modell einer Straßenbahn in der AR-Umgebung eines Tablets auf definierte QR-Codes projiziert wird. Diese Erkennung wurde mithilfe von AR-Core Image Tracking realisiert¹⁰. Folgend darauf, haben wir die QR-Code Erkennung so ausgebaut, so dass verschiedene Prefabs durch verschiedenen QR-Codes dargestellt werden können. Die Idee war hierbei, die Funktionalität bereits zu implementieren, um später auf ein physisches Haus mit QR-Code eine tragbare Infotafel zu projizieren. Diese Idee setzten wir schlussendlich nicht um.

¹⁰ Microsoft, "AR-Foundation-Samples", 2023, Link: <https://github.com/Unity-Technologies/arfoundation-samples>

Als die einfache AR-App für das Tablet stand war unser nächstes Ziel, die Unity App auf der HoloLens 2 mit Hilfe des Mixed Reality Toolkits 3 zu bauen. Doch dann ist uns schnell klar geworden, dass unser anfänglicher plan, alles auf MacOS zu entwickeln nicht aufging, da Apple OpenXR nicht unterstützt. Von diesem Moment an sind wir vollständig auf Windows als Entwicklungsbetriebssystem umgestiegen.

Nachdem die MQTT-Verbindung per M2MQTT stand, haben wir unsere Scene in Unity in vier Hauptsektoren eingeteilt. Der Grund hierfür war um die Interaktionen und trennen und so sauberer und strukturierter arbeiten zu können. So hat dann zum Beispiel jede Interaktion erst mal ein einzelnes Skript bekommen. Zeitgleich haben wir eine eigene City-Map in die Scene eingebaut, die wir in Blender erstellt haben, um anhand von dem Modell die Interaktionen schon mehr in die Richtung bringen wollten, wie wir sie schlussendlich haben wollten.

Durch den Wechsel von OpenXR zu MRTK-3 mussten nun feststellen, dass eine Übertragung der QR-Code Funktionalität von OpenXR zu MRKT-3 zwar technisch möglich war, jedoch vermutlich sehr unperformant wäre. Deshalb implementierten wir eine neue Methode der QR-Code-Erkennung, basierend auf dem Microsoft.MixedReality.QR (Version 0.5.3037) NuGet-Paket. Dieses nutzt die interne QR-Erkennung der HoloLens 2. Das neue Feature stellt sicher, dass nur QR-Codes mit spezifischem Inhalt getrackt werden, um die Beeinflussung durch andere QR-Codes in der Umgebung zu verhindern.

Jedoch taten sich bei der Umsetzung in MRTK3 viele Probleme auf. Nachdem wir MRKT-3 überhaupt erstmal zum Funktionieren brachten, was einiges an Bugfixing und Workarounds mit sich zog, stellten wir fest, dass der neue Ansatz für den QR-Code in MRTK-3 nicht funktionierte, auch wenn wir einige manuelle Code Updates durchführen.

Die Umsetzung des Projektes mit MRTK 3 mussten wir trotz eines erweiterten Funktionsumfängen aussetzen, da Microsoft den Support für diese MRTK-Version eingestellt hat und damit viele Funktionen unzureichend dokumentiert und in Teilen nicht zu Ende entwickelt wurden. Auf Grund dessen griffen wir auf die vorherige Version MRTK2 mit einer umfangreicheren Dokumentation und vollständigeren Funktionsumfang zurück.

2.2.9 Visuelle Umsetzung

Nachdem wir dann auch erfolgreich das Projekt umgestellt haben, war unsere Hauptaufgabe dann noch alles zu verfeinern und unsere Idee visuell umzusetzen und alle Interaktionen weiter auszuarbeiten und mit unserem Narrativ zu verbinden. Die Scene erweiterten wir nach und nach mit 3D-Assets, die wir selbst in Blender modelliert haben oder heruntergeladen haben.

Da unser Stadtmodell zur Information Übertragung dient machten wir uns als Gruppe dazu Gedanken, wie wir das am besten machen und sind zum Entschluss gekommen, dass wir dem Nutzer die grobe Information visuell geben wollen, und die genauen wissenschaftlichen Informationen anhand von einer Texttafel übermitteln.

So haben wir zum einen zwei Varianten von Häusern eingebaut. Eine mit älterer Bauweise und eine mit moderner, CO₂ neutraler Bauweise. Beide Häuser besitzen einen eingebauten NFC-Chip, der ausgelesen wird und anhand dem die jeweiligen Häuser gerendert werden. Die Veränderung der Bauweise wurde die Veränderung durch das Entfernen der Schornsteine und dem hinzufügen von Solarpanelen auf den Dächern veranschaulicht.

Als weiteres Feature und Teil des Industriesektors bauten wir Rauch in Form eines Partikelsystems ein, der sich durch den Knopf von Rauch zu Wasserdampf ändern lässt. Zusätzlich tauscht sich das Prefab der Stahlwerke aus und auch der Inhalt bzw. das Material des entsprechenden Textboards.

Wir wollten auch in dem Stadtmodell etwas haben, wodurch wir auch dem Nutzer mehr von uns und dem Projekt erzählen können. Aufgrund dessen entschieden wir uns auch eine Texttafel einzubauen, die man mit einem virtuellen Button ändern kann, sodass der User trotz allen physischen Aktionen auch eine virtuelle Aktion durchführen kann. Dadurch kann der User auch noch einmal den Unterschied von den physischen und den virtuellen Interaktionen sehen.

Für den Drehregler wollten wir nun eine kontinuierliche Veränderung haben, die fließende Übergänge hat. Und so kam uns die Idee, die Prefabs des Energiesektors animiert auszutauschen. Die alten Prefabs werden nach und nach kleiner und verschwinden und die neuen erscheinen und werden nach und nach größer. Hier wurde der Drehregler Wert benutzt, um durch einen Algorithmus zu berechnen, wie alles animiert wird. Der gleiche Algorithmus wurde auch verwendet, um im Wald des Natursektors Bäume erscheinen zu lassen.

Als wir mit dem Modell schon relativ fortgeschritten waren, fiel uns auf, dass wir das Modell für jeden neuen Nutzer nicht zurücksetzen können ohne, dass man dafür die App neu starten muss. Deswegen bauten wir einen weiteren virtuellen Knopf ein, der die Infotafel unseres Projektes und das Modell auf den Anfangsstand stellt.

Nachdem all das implementiert war, fiel uns auch ein größerer Bug auf, den wir noch unbedingt beheben mussten. Es erschienen nämlich oft zwei augmentierte Stadtmodelle, nachdem man sich mit der HoloLens 2 etwas bewegt hat und die QR-Erkennung fehlschlug. Warum genau dieser Fehler entstand können wir nicht genau beurteilen, da sogar nach einem Neustart der App zwei Stadtmodelle an speziellen Orten im Raum vorhanden waren. Das behoben wir, indem wir vor Initialisierung eines Neuen Modells geschaut wir, ob es schon eines gibt, und dies dann löschen.

Nachdem wir das physische Modell gebaut haben, mussten wir die Positionen der beiden Modelle anpassen. So war alles weitgehend fertig. Danach haben wir noch einige Verschönerungen getätigt, wie 3D-Modelle einfügen oder auch die Windräder animieren und kleinere Bugfixes.

2.3 Komponentenübersicht

2.3.1 Arduino

Wie bereits unter Punkt 2.1 erwähnt, benutzen wir einen Arduino Uno Wifi Rev2. Dieser hat ein eingebautes Netzwerkmodul, wodurch wir dieses nicht separat einbauen und programmieren mussten.

Zur Entwicklung des Arduinos haben wir die Entwicklungsumgebung von Arduino verwendet. Die Arduino IDE haben wir in der Version 2.2.2 verwendet und für die Entwicklung mussten wir folgende Libraries installieren:

- ArduinoMqttClient 0.1.7
- WiFinINA 1.8.14
- Adafruit PN532 1.8.3
- ArduinoJson 7.0.1

Die elektronischen Bauteile, welche wir für die haptischen Interaktionen verwenden, werden über die GPIO-Pins mit dem Arduino verbunden. Wir verwenden hierbei einen Taster und einen 10k-Ohm Potentiometer, welche keine genauen Beschreibungen hatten. Ein PN532 NFC RFID Modul, eine ALLNET 4duino Two colour LED und eine Steckplatine, um alle elektronischen Bauteile mit dem Arduino zu verbinden.

Die elektronischen Bauteile sind wie folgt mit dem Arduino verbunden:

Taster	Arduino
5V	5V
GND	GND
Out	D4

Tabelle 1 Pin-Layout Taster

Potenziometer	Arduino
VCC	5V
Output	A0
GND	GND

Tabelle 2 Pin-Layout Potentiometer

LED	Arduino
GND	D-GND
R1	D11
R2	D10

Tabelle 3 Pin-Layout LED

NFC-Leser	Arduino
VCC	5V
GND	GND
SDA	SDA

SCL	SCL
IRQ	D2
RSTC	D3

Tabelle 4 Pin-Layout NFC-Leser

Da der Arduino nur einen 5V und einen GND-Pin hat, wurden alle Bauteile über die Steckplatine parallelgeschaltet.

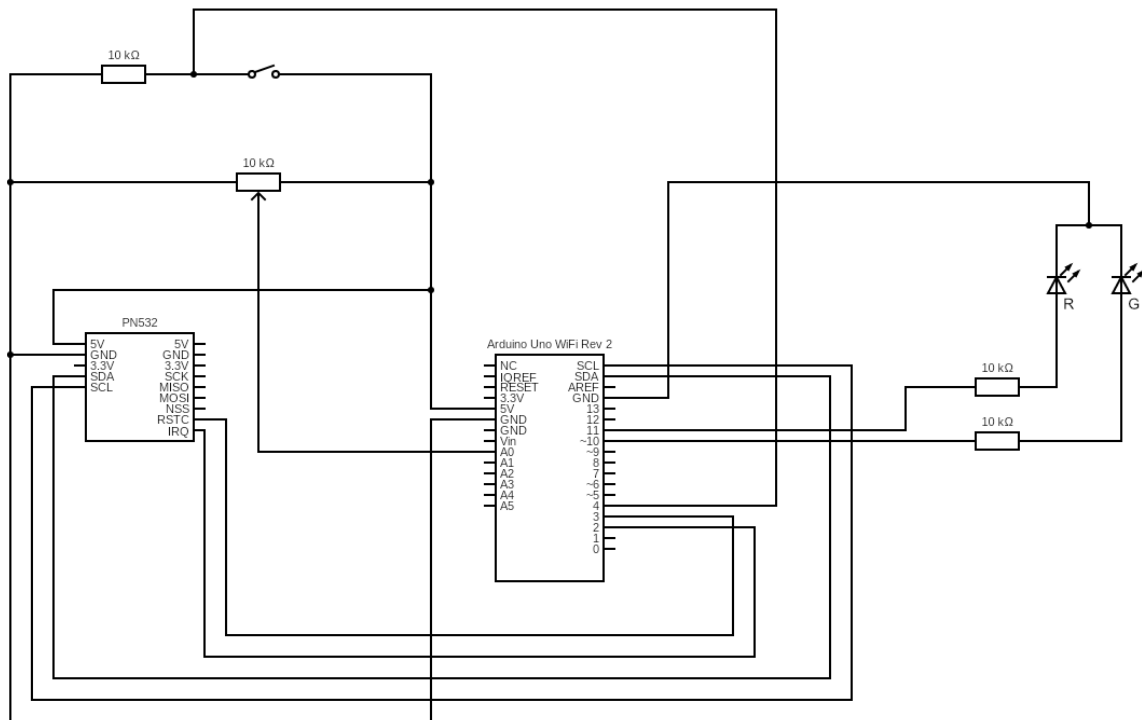


Abbildung 5 Schaltgrafik

2.3.2 RPI

Raspberry Pi 3 (Model B)

Betriebssystem:

Raspberry Pi Os (Legacy, 32bit) für RP3

Software:

NetworkManager 1.40

Mosquitto, Mosquitto-Clients 2.0.18

2.3.3 MQTT

Bei der Kommunikation haben wir uns für eine Drahtlose Übermittlung der Daten über das MQTT-Protokoll entschieden.

Hierbei haben wir zwei Clienten, den Arduino und die HoloLens, und einen Broker/Server, den Raspberry Pi. Der Arduino sendet, als Client, die Nachrichten an ein vordefiniertes Thema und den Broker und die HoloLens abonniert dieses Thema. Dadurch wird die gesendete Nachricht vom Broker an die HoloLens weitergeleitet und bekommt somit die Daten des Arduinos.

Damit diese Kommunikation funktioniert, verwenden wir auf Seite des Arduinos die Bibliothek `ArduinoMqttClient`¹¹ in der Version 0.1.7 und auf Seiten der HoloLens das `unity`-Paket `M2MQTT`¹² in der Version 4.3.0, welches in das Projekt eingebunden ist. Der Broker benutzt das aktuelle Mosquitto Debian Packet¹³.

2.3.4 Unity

Für die Unity Version haben wir uns auf die 2022.3.13f1 geeignet.

Um auf die Mixed Reality Funktionalitäten zugreifen zu können, wie zB. die Raumwahrnehmung, QR Code Erkennung, Interaktionen müssten wir folgende Packages installieren:

- Mixed Reality OpenXR Plugin 1.10.0
- Mixed Reality Toolkit 2
- NuGetForUnity: Version 4.0.2
 - Microsoft.MixedReality.QR (Version 0.5.3037) : SDK für die Erkennung von QR-Codes auf Windows Mixed Reality-Geräten.
 - Microsoft.VCRTForwarders.140 (Version 1.0.5): wurde implizit damit installiert.
- M2MQTT 4.3 (von eclipse/paho.mqtt.m2mqtt)
- SimpleJSON Library benutzt.
- TextMesh Pro

2.3.5 Datenformat

Bei den zu übertragenden Daten haben wir uns für das JSON-Format entschieden. Das JSON-Format hat eine einfache Syntax, welche Key-Value-Paare verwendet. Unser JSON-Objekt besteht aus insgesamt drei Key-Value-Paare. Einer ID, einem Zeitstempel und einem Datensatz.

```
{"id": 1, "timestamp": 123, "data": "message"}
```

¹¹ `ArduinoMqttClient` - Arduino reference: o. D., [online]
<https://www.arduino.cc/reference/en/libraries/arduinomqttclient/>.

¹² Eclipse: GitHub - eclipse/paho.mqtt.m2mqtt, in: GitHub, o. D., [online]
<https://github.com/eclipse/paho.mqtt.m2mqtt>.

¹³ Mosquitto Debian repository: in: Eclipse Mosquitto, 10.01.2013, [online]
<https://mosquitto.org/blog/2013/01/mosquitto-debian-repository/>.

Die ID gibt an um welche haptische Interaktion es sich handelt. Eins steht für den Taster, zwei für den Potenziometer und drei für den NFC-Leser. Der Zeitstempel gibt die aktuelle Zeit in Millisekunden des Arduinos seit Systemstart wieder. Data enthält die zu senden Daten von den jeweiligen Interaktionen. Beim Taster ist dies ein leerer String, beim Potenziometer ein Zahlenwert zwischen 0 und 1023 und beim NFC-Leser die ID des NFC-Tags.

3 Fazit der Projektteilnehmenden

3.1 Timo Boomgaarden

Ich gehe begeistert und mit Lust auf mehr aus diesem Projekt! Ich habe gelernt, wieviel mehr zu einem Projekt gehört, als einfach nur Code zu schreiben. Dabei wurde mir die Wichtigkeit von unter anderem Dingen wie Projektplanung, realistische Zeiteinschätzung, gegenseitige Motivation, gute und klare Kommunikation über aktuelle Vorhaben und Probleme, das Schaffen guter Rahmenbedingungen zum Arbeiten wie dem Pull-System klar.

Für mich außerdem überraschend war wie kompliziert es werden kann, die besten Libraries zu finden, zu verstehen und anzuwenden, sowie mit unvorhergesehenen Bugs umzugehen und diese irgendwie, auf kreative Art und Weise zu debuggen, da ein vernünftiger Konsolenoutput an vielerlei Stellen nicht möglich war.

Ich erinnere mich daran, wie zu Anfang des Projekts die Kommunikation und Zusammenarbeit untereinander wirklich holprig verlief, und bin entzückt, wenn ich an die gemeinsame Arbeitsweise der späteren Monate denke, nachdem Probleme in gemeinsamen Meetings angesprochen und Kompromisse und Lösungen für bessere Zusammenarbeit gefunden wurden.

3.2 Yasmine Haidri

Das Projekt war für mich eine sehr schöne Erfahrung. Ich hatte vorher gar keine Ahnung über AR, was am Anfang sehr herausfordernd klang, konnte aber am Ende viel mitnehmen. Was ich als größtes Learning gewonnen habe ist, dass manchmal hören sich die Dinge komplizierter als was sie sind und dass man sich davon nicht überfordern lassen sollte. Klar ist das Thema komplex, aber man muss nicht alles vom Scratch sich ausdenken da es mehrere SDKs, Packages und Libraries gibt die schon viel vereinfachen und eine gute Basis anbieten um drauf zu entwickeln und Freiraum zu Kreativität lassen. Zwar haben wir mehrmals viel Zeit verbraucht die zu verstehen und richtig zu nutzen oder mit Inkompatibilität zwischen den Versionen zu kämpfen, aber so ist ja immer der Entwicklungsprozess, wenn man beim Machen sich erst einarbeitet und die Dokumentation und Forschung im Bereich noch nicht ganz gut ist.

Ich fand es interessant, besseres Verständnis und einen Blick auf der Hardware zu kriegen, auch wenn ich selbst nicht darauf gearbeitet habe. Was ich besonders spannend in diesem Projekt fand, ist die Netzwerkkommunikation und mit Unity rumzuspielen und C# Programmierungserfahrung zu sammeln. Die Zusammenarbeit hat auch echt viel Spaß gemacht und sehr gut funktioniert, wir haben uns gegenseitig ständig motiviert und uns sehr gut komplementiert.

3.3 Jonas Mantay

Das Projekt war insgesamt eine gute Erfahrung für mich. Die Möglichkeit, Unity zu verwenden und die Programmierung zu erlernen, hat mich wirklich begeistert. Es war faszinierend zu sehen, wie vielseitig Unity ist und welche kreativen Möglichkeiten sich damit eröffnen. Spannend fand ich, wie wir Unity im Kontext der Mixed Reality nutzen konnten.

Die Arbeit mit der HoloLens war ein Höhepunkt. Das ständig wachsende Interesse der Allgemeinheit an Mixed Reality hat die Arbeit damit noch aufregender gemacht. Es war inspirierend zu sehen, wie diese Technologie das Potenzial hat weiter ausgebaut zu werden und in Zukunft noch mehr zu Mixed Reality wird.

Ich fand es auch gut in einem Team zu arbeiten und es auch einigermaßen realitätsnah zu gestalten, wie der Projektaufbau, die Arbeitseinteilung und Teamarbeit aussahen. Eine weitere Sache, bei der ich sehr viel lernen konnte, war Git. Wir mussten alle Git-Befehle im Terminal ausführen. Dadurch kann ich jetzt mit Git umgehen und weiß, wie man mit Mergekonflikte umgeht.

Meine Gruppe hat mir auch gefallen. Ich fand wir haben es gut gemeistert zu kommunizieren. Am Anfang hatten wir da unsere Schwierigkeiten, sind da dann aber nach einiger Zeit gut reingekommen, sodass ich zum Schluss auch das Vertrauen in meinen Teammitglieder hatte, dass alle Aufgaben gut gelöst werden.

3.4 Christian Wolter

Die Entwicklung eines digital-haptischen Stadtmodells mittels XR war etwas komplett Neues für mich. Es war das erste Mal, das ich mit einer HoloLens, mit AR-Applikationen und mit Microcontrollern gearbeitet habe und damit auch einhergehend eine neue und Aufregende Challenge. Der Start der Entwicklung war zwar etwas holprig, da wir erst einen Rhythmus finden mussten und die Entwicklung auf dem Mac nicht ohne Probleme lief aber diese Probleme konnten gelöst werden.

Während der Entwicklung habe ich die Rolle des Teamleiters übernommen und mich neben der Entwicklung auch um die Organisation gekümmert. Das heißt ich habe geguckt, das im Team alles reibungslos funktioniert, dass wir einen Fortschritt sehen und auch Aufgaben klar abgegrenzt um den Fokus auf das Projektziel zu waren. Zudem habe ich mich gegen Ende des Projekts um alle weiteren Abgaben, wie der Präsentation für den Kurs Projektmanagement, der Showtime-Website und den Stand auf der Showtime gekümmert.

Meine Entwicklung hat sich hauptsächlich auf die Programmierung des Arduinos beschränkt, wodurch ich als zweite Aufgaben die genannte Organisation übernommen habe. Ich habe aber auch bei der XR-Applikation unterstützt und zu Anfang das Projekt mit MRTK3 aufgesetzt, sowie mich mit der Entwicklung von HoloLens Applikationen auf dem Mac beschäftigt.

Alles im allen fand ich das Projekt und auch die Arbeit im Team sehr gut. Wir haben alle sehr gut zusammengearbeitet und die Aufteilung der Aufgaben waren auch fair gehandhabt, wodurch niemand zu viel oder zu wenig für das Projekt gemacht hat.

Auch Absprachen im Team und mit dem Projektbetreuer haben sehr gut funktioniert und wir haben, auch wenn es zeitlich sehr eingespannt war, einen guten Rhythmus gefunden, um diese Termine abzuhalten.

Das Projekt an sich und auch die Zusammenarbeit haben sehr gut funktioniert. Das Projekt war sehr spannend und hat mir persönlich neue Einblicke in die Entwicklung von Microcontrollern und AR-Applikationen gegeben. Zudem habe ich durch das Projekt meinen Sinn für die Projektorganisation geschärft.

4 Deployment-Sheet

4.1 Unity

4.1.1 Installation von Unity und Visual Studio:

Zu Beginn muss Unity Hub auf einem Windows-Rechner mit Windows 10 oder 11 als Betriebssystem installiert werden. Nach der Installation von Unity Hub muss die Unity Version 2022.3.13f1 installiert werden.

Für die Entwicklung in Unity wird Visual Studio 2022 benötigt. Beim Einrichten von Visual Studio müssen bestimmte Komponenten im Visual Studio Installer ausgewählt und installiert werden. Dies umfasst:

- .NET-Desktopentwicklung
- Desktopentwicklung mit C++
- Spieleentwicklung mit Unity
- Entwicklung für die universelle Windows-Plattform
- Windows 10 SDK-Version 10.0.19041.0 oder 10.0.18362.0, oder Windows 11 SDK
- USB-Gerätekonnektivität (erforderlich für die Bereitstellung/Debugging auf der HoloLens über USB)
- C++ (v142) Universal Windows Platform Tools

Eine detaillierte Anleitung zur Installation ist auch auf offiziellen Microsoft-Dokumentation zu finden: <https://learn.microsoft.com/en-us/windows/mixed-reality/develop/install-the-tools>

4.1.2 Konfiguration in Unity und Visual Studio:

Nach der Installation können Sie das Projekt in Unity öffnen und in den Build-Einstellungen die Plattform auf die universelle Windows-Plattform ändern. Anschließend kann das Projekt gebaut werden. Hierfür wird ein Ordner erstellt werden, in den das Projekt gebaut wird.

Die Build-Konfigurationen in Visual Studio müssen auf ARM oder ARM64, Release und Gerät geändert werden.

Eine detaillierte Anleitung zur Konfiguration ist auch auf offiziellen Microsoft-Dokumentation zu finden: <https://learn.microsoft.com/de-de/windows/mixed-reality/develop/advanced-concepts/using-visual-studio?tabs=hl2#enabling-developer-mode>

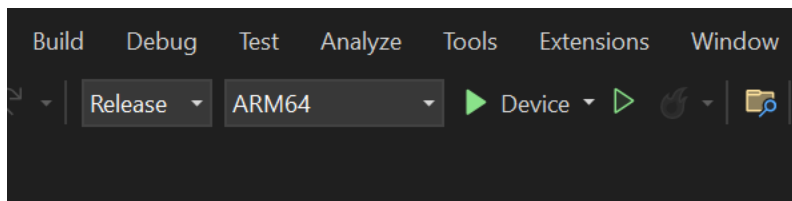


Abbildung 6: Visual Studio Konfiguration

4.1.3 Freischalten des Developer-Modus auf der HoloLens 2:

Bevor das Projekt auf der HoloLens 2 bereitgestellt werden kann, muss der Developer-Modus auf der HoloLens 2 aktiviert werden. Hierfür müssen in den Systemeinstellungen unter "Update und Sicherheit" die Entwicklerfunktionen aktiviert werden. Wenn die HoloLens 2 über ein Kabel mit dem Computer verbunden ist, kann die Anwendung über den Startknopf in Visual Studio gebaut werden.

Eine detaillierte Anleitung zur Freischaltung des Developer-Modus ist in derselben offiziellen Microsoft-Dokumentation zu finden wie die Konfiguration in dem vorherigen Schritt.

4.2 RPI

4.2.1 Raspberry Pi 3 starten

Es muss die neueste Debian Raspberry Pi OS Distribution installiert werden. Für diesen Schritt wird eine Internetverbindung benötigt.

Führen Sie die folgenden Befehle aus:

```
sudo apt-get update  
sudo apt-get upgrade  
sudo apt install -y mosquitto mosquitto-clients  
sudo apt install network-manager
```

Eine Internetverbindung ist nach diesem Schritt nicht mehr erforderlich.

4.2.2 Starten des WLAN-Hotspots

Für den WLAN-Hotspot wird der Dienst "Network-Manager" verwendet.

Da Raspberry Pi OS 2023-12-05 standardmäßig den Dienst "dhcpcd" verwendet, müssen wir umschalten und aufräumen. Dazu muss das folgende Kommando ausgeführt werden:

```
sudo raspi-config
```

In dem geöffneten Fenster muss auf Punkt 6. “Advanced Options” geklickt werden. Daraufhin muss der Reiter “AA Network Config” ausgewählt werden. Schließlich muss nun der NetworkManager ausgewählt und bestätigt werden. Anschließend kann auf finish geklickt und ein Reboot durchgeführt werden.

Daraufhin können sollten die nicht mehr benötigten Komponenten entfernt werden:

```
apt purge openresolv dhcpcd5
```

Das Einrichten des WLAN-Hotspots lässt sich im Raspberry Pi Os sehr schnell über die GUI lösen. Dafür muss auf das WLAN-Symbol in der oberen rechten Ecke geklickt, und anschließend auf “Erweiterte Optionen” geklickt werden. Dort wird nun “Create a WIFI Hotspot” gewählt.

*Als Netzwerkname wird “**HoloHub**” gewählt, die Wi-Fi-Security sollte auf WPA & WPA2 Personal eingestellt werden und das Passwort sollte auf “**b7ws2324**” gesetzt werden.*

Dies generiert eine gute Konfigurationsgrundlage. Diese kann mithilfe von:

```
sudo nano /etc/NetworkManager/system-connections/HoloHub.nmconnection
```

gefunden werden. Hier können zusätzliche Einstellungen vorgenommen werden. Für den lokalen WLAN-Hotspot sollte dies vorerst nicht erforderlich sein.

Um den WLAN-Hotspot automatisch zu starten, können Sie nach erneutem Klicken auf das WLAN-Symbol und Advanced Options auf “Edit Connections...” geklickt werden. In dem neuen Fenster muss nun unter “General” die Option “Connect Automatically with priority” ausgewählt werden.

Später wird der Mosquitto-Broker standardmäßig unter der IP-Adresse 10.42.0.1 erreichbar sein.

4.2.3 Starten von MQTT

MQTT ist das Protokoll, das einen Broker erstellt, auf dem Nachrichten unter verschiedenen Themen gesendet und abgerufen werden können. Themen können von Abonnenten und Clients beliebig definiert werden und müssen nicht vom MQTT-Server oder MQTT-Broker vordefiniert werden.

Weitere Informationen finden sich unter anderem in der [Manpage](#).

Nun muss eine Konfigurationsdatei für Mosquitto erstellt werden:

```
sudo nano /etc/mosquitto/conf.d/local.conf
```

In der Konfigurationsdatei müssen 2 Zeilen hinzugefügt werden:

```
listener 1883  
  
allow_anonymous true
```

4.2.4 Testen, ob der MQTT-Broker funktioniert.

Um den MQTT-Broker zu testen, lässt sich dieser gut im Verbose-Modus starten:

```
mosquitto -v -c /etc/mosquitto/conf.d/local.conf
```

Durch den Start im Verbose-Modus können wir genau beobachten, was im Broker passiert.

In 2 zusätzlichen Terminalregisterkarten können lokal Nachrichten empfangen und gesendet werden.

Um Nachrichten zu empfangen kann ein Subscriber-Terminal geöffnet werden:

```
mosquitto_sub -t "test"
```

In dem anderen Terminal geben Sie ein:

```
mosquitto_pub -t "test" -m "Hallo, Welt!"
```

Sie können sich auch von einem anderen Computer bei HoloHub anmelden. Danach sollten Sie in der Lage sein, Nachrichten mit folgendem Befehl zu senden:

```
mosquitto_pub -h 10.42.0.1 -p 1883 -t "test" -m "Hallo, Welt!"
```

4.2.5 Einrichten des Autostarts

Um den Broker beim Systemstart automatisch zu starten, verwenden Sie:

```
sudo systemctl enable mosquitto
```

```
sudo systemctl start mosquitto
```

Jetzt wird der MQTT-Broker gestartet, sobald der RP mit Strom verbunden ist.

Sobald das HoloLens 2 und Arduino mit dem HoloHub WLAN verbunden sind, wird die Kommunikation funktionieren.

4.2.6 Bekannte Probleme

Sollte es wider Erwartungen zu Problemen kommen, gibt es hier eine kurze Erläuterung was bei verschiedenen Fehler-Messages unternommen werden kann.

Bei dem Fehler "Adress already in use" läuft (vermutlich) bereits eine Version von MQTT. Diese kann gestoppt werden mithilfe von `$ sudo systemctl stop mosquitto`. Bei weiteren Problemen lohnt es sich die Prozesse zu killen.

Bei dem Fehler, wobei der Broker nur im lokalen Modus startet ist es sinnvoll sicherzustellen dass dieser Zugriff auf die Config-Datei hat.

4.3 Arduino

Die folgende Anwenderdokumentation erklärt, wie Sie den Arduino für das Projekt anpassen und starten können. Sollten Sie Einstellungen wie WLAN- oder Broker-Einstellungen ändern

wollen oder das Projekt weiterentwickeln möchten, empfiehlt es sich, die Arduino IDE zu verwenden.

4.3.1 Anforderungen

Für die Entwicklung, das Anpassen, Bauen und Deployen des Arduino Codes sollte die Arduino IDE verwendet werden. Hierzu müssen folgende Pakete installiert werden, damit der Arduino Code auch gebaut und deployt werden kann:

- ArduinoMqttClient by Arduino
- Wi-Fi-NINA by Arduino
- ArduinoJson by Benoit Blanchon
- Adafruit PN532 by Adafruit

4.3.2 Pin-Layout

Taster	Arduino
5V	5V
GND	GND
Out	D4

Tabelle 5 Pin-Layout Taster

Potenzimeter	Arduino
VCC	5V
Output	A0
GND	GND

Tabelle 6 Pin-Layout Potentiometer

LED	Arduino
GND	D-GND
R1	D11
R2	D10

Tabelle 7 Pin-Layout LED

NFC-Leser	Arduino
VCC	5V
GND	GND
SDA	SDA
SCL	SCL
IRQ	D2
RSTC	D3

Tabelle 8 Pin-Layout NFC-Leser

4.3.3 W-Lan Einstellungen

Das zu verwendendes Netzwerk kann in der Datei "wifi_secrets.h" hinzugefügt werden. Dazu müssen die Daten für das Netzwerk (SSID und Passwort) in den entsprechenden Feldern eingegeben werden. Diese wurden bei der Einrichtung des W-Lan Access-Points vergeben (siehe 4.2.2). Bitte stellen Sie sicher, dass dieses Netzwerk auch das entsprechende Netzwerk ist, in dem der MQTT-Broker läuft, da nur so eine Verbindung zum Broker aufgebaut werden kann.

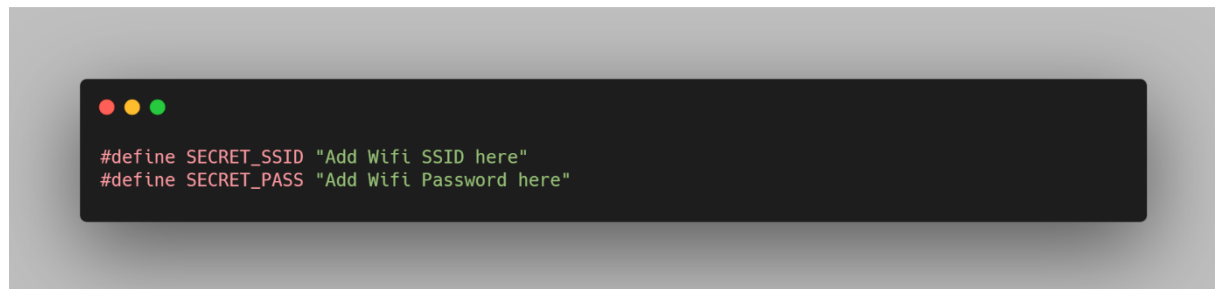


Abbildung 7 Netzwerk-Einstellung am Arduino

4.3.4 MQTT-Einstellungen

Die MQTT-Broker-Daten können in der Datei "broker_conf.h" geändert werden. Sie können die Daten für den MQTT-Broker auf dem Gerät, auf dem der Broker läuft, abrufen. Die Broker-IP ist meistens die IP des Hosts und der Port ist standardmäßig 1883.

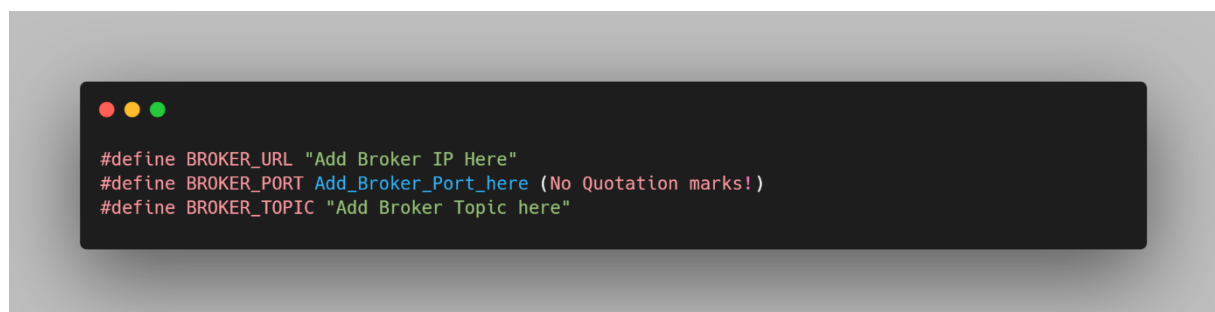


Abbildung 8 MQTT-Broker Einstellungen am Arduino

Hinweis: Bitte stellen Sie sicher, dass die angegebenen Daten korrekt sind, um eine reibungslose Funktion des Systems zu gewährleisten. Für die Entwicklung wird die Verwendung der Arduino IDE empfohlen.

5 Quellenverzeichnis

ArduinoMQttClient - Arduino reference: o. D., [online]

<https://www.arduino.cc/reference/en/libraries/arduinomqttclient/>.

AttachInterrupt() - Arduino-Referenz: o. D., [online]

<https://www.arduino.cc/reference/de/language/functions/external-interrupts/attachinterrupt/>.

Eclipse: GitHub - eclipse/paho.mqtt.m2mqtt, in: GitHub, o. D., [online]

<https://github.com/eclipse/paho.mqtt.m2mqtt>.

oop() - Arduino-Referenz: o. D., [online]

<https://www.arduino.cc/reference/de/language/structure/sketch/loop/>.

Mosquitto Debian repository: in: Eclipse Mosquitto, 10.01.2013, [online]

<https://mosquitto.org/blog/2013/01/mosquitto-debian-repository/>.

MQTT - The Standard for IoT Messaging: o. D., [online] <https://mqtt.org/>.

Robin: Interrupts (Assembler) – informatik abitur, o. D., [online] [https://informatik-](https://informatik-abitur.de/assembler/interrupts/82/)

[abitur.de/assembler/interrupts/82/](https://informatik-abitur.de/assembler/interrupts/82/).

Söderby, Karl: Sending Data over MQTT, in: Arduino Documentation, 30.01.2024, [online]

<https://docs.arduino.cc/tutorials/uno-wifi-rev2/uno-wifi-r2-mqtt-device-to-device/> (abgerufen am 18.03.2024).

TheDarkKnight: How to use MQTT in HoloLens?, in: Mixed Reality Developer Forum,

18.04.2017, [online] <https://forums.hololens.com/discussion/7222/how-to-use-mqtt-in-hololens>.

Zambetti, Nicholas/Karl Söderby/Jacob Hylén: Inter-Integrated Circuit (I2C) Protocol, in:

Arduino Documentation, 16.01.2024, [online]

<https://docs.arduino.cc/learn/communication/wire/> (abgerufen am 18.03.2024).

Microsoft, "Auswählen Ihrer Engine", 12.07.2023, Link: <https://learn.microsoft.com/de-de/windows/mixed-reality/develop/choosing-an-engine?tabs=unity> (Stand 15.03.2024)

Microsoft, "AR-Foundation-Samples", 2023, Link: <https://github.com/Unity-Technologies/arfoundation-samples>