

Cross-Innovation-Class

”How-To Smart-City”

Wald | Stadt | Labor Iserlohn



Fachhochschule Wedel: Luca Krause, Paul Obernesser
Akademie für Mode & Design: Louisa Soetbeer, Luca Salmina
HafenCity Universität Hamburg: Kayra Schiffer, Clara Hartmann,
Johannes Perschmann

Juli 2022

Inhalt

1 Einleitung	3
1.1 Kontext der Cross-Innovation-Class	3
1.2 CIC-Prozess	4
1.3 Industriepartner	4
1.4 Skizzierung der Idee	5
2 Projektbeschreibung	6
2.1 Weiterführende Idee und Realisierung	6
2.2 Aufgabenverteilung innerhalb des Teams	8
3 Herausforderung bei der Realisierung	9
3.1 Druckbarkeit des Stadtmodells	9
3.2 NFC-Tags und Spule für Strom gleichzeitig	10
3.3 Generierung der Inhalte für die Projektion	10
3.4 Die Busroute	11
3.5 Positionierung der Projektion	11
3.6 Größe der Vitrinen	12
3.7 Vitrinen Spannung	12
3.8 Herausforderungen bei der interdisziplinären Arbeit	14
4 Ergebnis	14
4.1 Stadtmodell	14
4.2 Infrastruktur	16
4.2.1 Diskussion Technik Auswahl	16
4.2.2 Kommunikation	17
4.3 Sockel	17
4.4 Vitrinen	20
4.4.1 Konstruktion	21
4.4.2 Baum	21
4.4.3 aBUS	23
4.5 Projektion	24
4.5.1 Generierung der Inhalte	24
4.5.2 Kalibrierung	25
4.6 Anzeige der Projektionsfläche	28
5 Fazit	30
5.1 Lesson's Learned	31
6 Ausblick	31

7 Repository 32

1 Einleitung

1.1 Kontext der Cross-Innovation-Class

Die Cross-Innovation-Class ist ein Projekt der Kreativgesellschaft Hamburg. Dabei arbeitet sie mit drei Hochschulen/Universitäten aus unterschiedlichen Fachbereichen zusammen. In diesem Semester waren dies die Akademie für Mode und Design, die HafenCity-Universität und die Fachhochschule Wedel. Aus diesen Lehranstalten werden jeweils ungefähr sechs Personen in eine Gruppe gesteckt, wobei das Verhältnis an Studenten aus jedem Fachbereich ungefähr gleich ist.

Zu jeder Gruppe gibt es einen Praxispartner, ein Unternehmen, welches in Innovationen und Nachhaltigkeit investieren möchte. Dieser Partner gibt seiner zugewiesenen Gruppe eine Fragestellung und unterstützt sie beim Beantworten. Teilweise konnten sich die Studenten mit Erst-/Zweitwunsch ihren Praxispartner wählen.

Die Mitarbeiter der Kreativgesellschaft hilft den Gruppen durch verschiedene Workshops und Inputs zum Ideen sammeln, Konzepte entwickeln und Erstellen von Prototypen. Während dieser Workshops kommt es außerdem immer wieder zu Vorstellungen der Zwischenzustände, in der man seinen eigenen Fortschritt mit den anderen Gruppen vergleichen kann und auch Austausch mit Ihnen hat und so eventuell gleiche Probleme besprechen kann. Zudem gibt es am Ende ein Pitch-Training, in dem das eigene Mindset und das Vorstellen eines Produktes verbessert werden kann.

Die Cross-Innovation-Class 2022 fand im Rahmen des Themas der Urbanen Resilienz statt, in der es darum geht, wie wir, als Bewohner größerer Städte, die auf uns zukommenden Herausforderungen des 21. Jahrhunderts lösen. Bei diesen Herausforderungen gibt es beispielsweise Flüchtende, das Klima, Pandemien, Armut, und etliches weiteres. Das Lösen dieser Krisen soll dabei aber auch das Zusammenleben der Menschen so angenehm, wie nur möglich, gestalten.

Am Ende der Zusammenarbeit im Rahmen der Cross-Innovation-Class kann es außerdem zwischen dem jeweiligen Praxispartner und den Studenten aus deren Team zu einem Arbeits-/Weiterführungsvertrag kommen, in der die Studenten in darauf folgenden Monaten zum Beispiel als Werkstudent an dem Projekt weiterarbeiten und dafür honoriert werden, oder das Projekt wird an das Unternehmen verkauft, wenn von studentischer Seite kein Interesse mehr besteht, das Projekt weiterzuführen. Der Prototyp ist aber geistiges Eigentum der Studenten.

1.2 CIC-Prozess

Der Prozess der Cross-Innovation-Class ist im Grunde der "Double Diamond" (siehe Abbildung 1). Dabei hat man einen Problem- und einen Lösungsraum. Zu Anfang hat man eine Fragestellung oder ein Problem, welches gelöst werden soll. Im Anschluss daran wird Feldforschung betrieben, um den Problemraum zu erweitern und um ihn im Anschluss zu definieren. Hat sich nun ein bestimmtes Problem herausgestellt, tritt man in den Lösungsraum ein, in dem eine Lösung zu dem vorher festgestellten Problem herausgefunden werden soll. Anfangs werden Ideen und Entwürfe gesammelt und anschließend wieder in der Zahl minimiert, um sich auf eine oder einzelne Lösungen zu konzentrieren. Diese Lösungen werden im Anschluss zu einem Prototypen entwickelt.

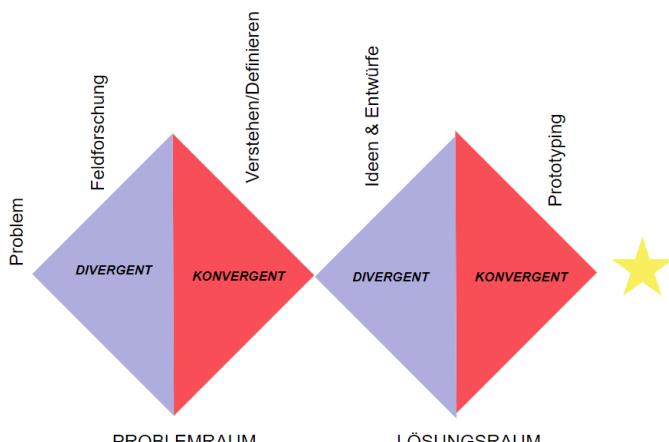


Abbildung 1: Double Diamond

1.3 Industriepartner

Unser Industrie-/Praxispartner war das Wald | Stadt | Labor Iserlohn. Das Wald | Stadt | Labor ist im Grunde ein multifunktionaler Raum, an dem Ideen der Bürger:innen für die nachhaltige Stadtentwicklung gesammelt und diskutiert werden können. Zudem werden dort Veranstaltungen, Schulungen und Sprechstunden durchgeführt. An diesen Treffen kann jede Person teilnehmen, die es möchte und dort und auch zwischen den werden außerdem allgemeine Fragen zur nachhaltigen Stadtentwicklung, Smart Cities und der Digitalisierung beantworten.

Im Wald | Stadt | Labor spielt vor allem die digitale Transformation der Stadt Iserlohn eine große Rolle. Darauf hinaus werden noch andere wichtige Themen, wie der Klimawandel, die Mobilität, Nachhaltigkeit und Stadtentwicklung behandelt. Das Team „Iserlohn digital“ hat außerdem eine Förderung zur Smart-City erhalten und setzt mit Hilfe der Förderung Projekte (Workshops, Veranstaltungen, etc.) um.

Unser Praxispartner hat uns die folgende Frage gestellt, auf der alle folgenden Ideen und Konzepte basieren:

Wie lässt sich das Thema SMART CITY für die Stadtgesellschaft im Wald | Stadt | Labor Iserlohn erlebbar und greifbar gestalten (z.B. Smart Gadgets, Sensorik)?

1.4 Skizzierung der Idee

Um diese Fragestellung zu beantworten, hatten wir mehrere Ideen. Da gab es zum Beispiel einen Smileyscanner, der, wenn eine vorbei laufende / angehaltene Person lächelt, die Tür öffnet. Wir haben uns allerdings dagegen entschieden, weil wir dachten, dies würde Besucher eher abschrecken und der Scanner könnte eventuell Besucher gar nicht in das Wald | Stadt | Labor lassen, weil kein Lachen erkannt wird. Außerdem wäre es rechtlich bedenklich, eine Kamera in der Öffentlichkeit aufzustellen.

Im Anschluss daran war dort erst die Idee mit einem Stadtmodell und dann noch eine andere Idee, die auf RFID-Kommunikation zwischen verschiedenen Komponenten basiert. Teilweise hatten wir auch noch eine Idee, die auf der Zusammenführung von Sensordaten beruht, dazu aber im Ausblick später mehr.

Diese beiden Ideen haben wir zusammengeführt in ein größeres Projekt. Dabei haben wir ein Stadtmodell, das als Projektionsfläche für einen Stadtplan dient, der beliebig erstellt werden kann und daneben befinden sich noch Informationen zu diesem Stadtmodell oder anderen Dingen, die in der Stadt passieren. Diese Projektion kann durch verschiedene mit NFC-Tags versehenen Gegenstände verändert werden, wenn diese auf einen Sockel gestellt werden.

Durch die Zusammenführung der zwei Konzepte hat nun der Besucher des Wald | Stadt | Labor Iserlohn etwas Haptisches in der Hand und kann damit interagieren. Stellt er einen Gegenstand auf den Sockel, kommt es zu einer sofortigen visuellen Veränderung. Versteht die besuchenden Personen das Konzept nicht, können sie bei den Mitarbeitern nachfragen und kommen

so mit Ihnen in ein Gespräch über vernetzte Städte, die im Grunde eine Vernetzung verschiedenster digitaler Strukturen darstellt.

2 Projektbeschreibung

2.1 Weiterführende Idee und Realisierung

Unsere Idee basiert also auf einem interaktiven Stadtmodell (siehe Abbildung 2). Dieses Stadtmodell ist 3D-gedruckt und wird anders, als viele andere Modelle statt auf einen Tisch, an der Wand angebracht. Auf dieses Stadtmodell werden nun Bilder, wie zum Beispiel Straßenpläne, projiziert. Neben diesem Stadtmodell befinden sich außerdem erweiterte Informationen zu den übertragenen Bildern.

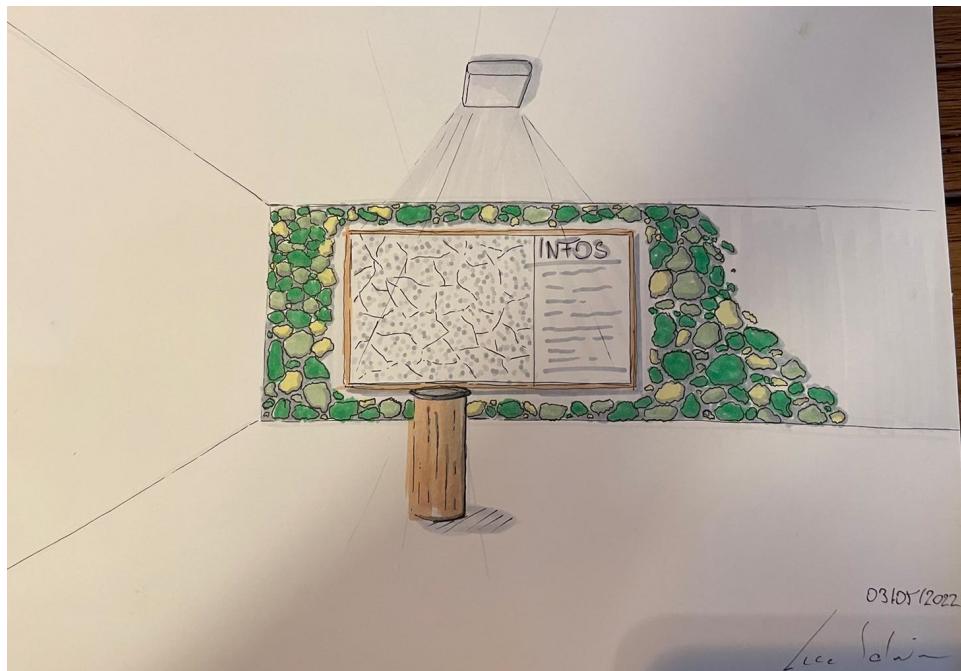


Abbildung 2: Anbringung an die Wand

Die Entscheidung, welche der gespeicherten Bilder und Informationen dort angezeigt werden, wird dem Nutzer des Modells überlassen. Dafür gibt es einen Sockel, auf den Vitrinen gestellt werden können.

Dieser Sockel besteht aus drei ausgehöhlten Teil-Holzstämmen und einem Betondeckel (wie in Abbildung 4 zu sehen). Steht keine Vitrine auf dem

Sockel, werden Standardinformationen angezeigt, dass doch bitte eine Vitrine auf den Sockel gestellt werden sollte. Steht eine Vitrine auf dem Sockel, werden Informationen zu dem Inhalt der Vitrine angezeigt.

Um dem Nutzer ein weiteres visuelles Feedback zu geben, ist der Baumstamm von innen beleuchtet, sobald eine Vitrine auf dem Sockel steht.

Der Deckel des Sockels ist zusätzlich leicht abnehmbar, um sich gut um die Technik kümmern zu können und Besuchern des Wald | Stadt | Labors Iserlohn die Technik erklären zu können, sofern sie interessiert daran sind.

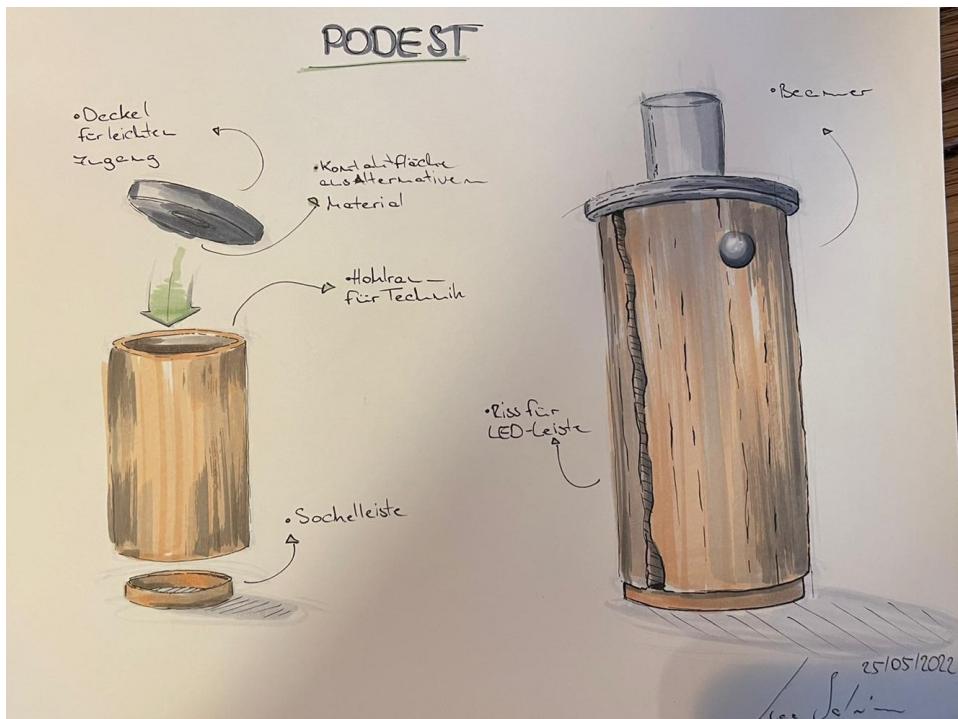


Abbildung 3: Sockel Aufbau

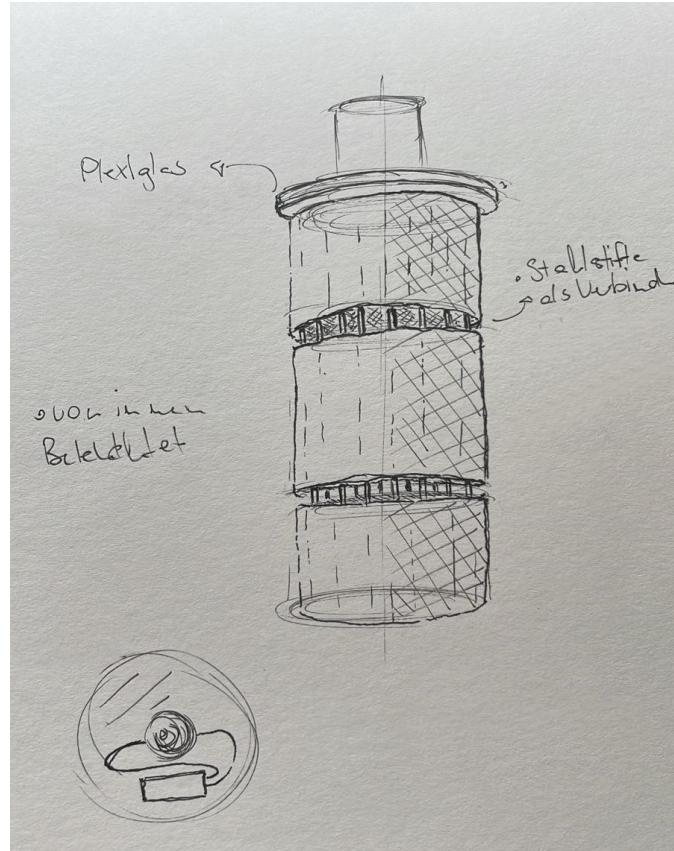


Abbildung 4: detaillierterer Sockel Aufbau

2.2 Aufgabenverteilung innerhalb des Teams

Kalibrierung:

Paul

OpenStreetMap:

Paul

mit Unterstützung von Luca

MQTT:

Paul

Luca

RFID-Reader:

Luca

Konstruktion:

Paul

3D-Druck:

15 Platten	Luca
10 Platten	Paul
2 Basisteile	Paul
Baum	Paul

Motor:

Paul

Luca

Beleuchtung des Baumstamms:

Paul

Luca

Zusammenbau der technischen Komponenten:

Paul

Luca

Zusammenbau des finalen Prototypen:

Paul

Luca Salmina

Louisa Soetbeer

Dokumentation:

Paul

Luca

3 Herausforderung bei der Realisierung

3.1 Druckbarkeit des Stadtmodells

Um ein Stadtmodell drucken zu können, benötigen wir zuallererst die Daten eines Stadtmodells. Das passende Stadtmodell haben wir aus OpenStreetMap extrahiert. Zuvor haben wir versucht, die Daten aus Google Maps zu erhalten, was aber nicht geklappt hat. Im Anschluss daran haben wir aus dem Wald | Stadt | Labor einen Zugang zum Geoportal Iserlohn bekommen, aus dem wir allerdings auch keine guten 3D-Modelle exportieren konnten.

Eine Herausforderung bestand darin, dieses Modell drucken zu können. Das exportierte Modell bestand aus vielen Objekten, die zum Teil fehlende Flächen aufwiesen, weshalb die Objekte beim Druck weggefallen wären oder das Modell vom Slicer gar nicht erst verarbeitet werden konnte.

Außerdem wurden bei dem Export nur die Objekte (Häuser etc.) exportiert, also mussten wir den Objekten noch einen Boden hinzufügen. Die Herausforderung dabei: Der Boden sollte nicht zu dick sein, um die Druckzeit nicht unnötig in die Länge zu ziehen, aber stabil genug sein, damit die Teile nicht einfach verbogen werden können. Dabei haben wir uns dann auf eine Dicke von 1.5 mm geeinigt.

3.2 NFC-Tags und Spule für Strom gleichzeitig

Anfangs hatten wir gedacht, dass es uns Probleme bereiten würde, wenn wir die Spule zur Spannungsversorgung und die NFC-Tags nebeneinander platzieren, weil die Magnetfelder eventuell interferieren könnten und die Tags somit nicht erkannt werden könnten. Allerdings behindert der Strom die Kommunikation mit den NFC-Tags nicht.

Außerdem mussten wir beide Bauteile in der Mitte des Sockels und der Vitrinen platzieren, damit die Stromübertragung und die Erkennung einwandfrei funktionieren. Diese müssen allerdings auch so nah wie möglich an ihren technischen Partnern sein, damit erstens mehr Strom übertragen wird und zweitens die Vitrine zuverlässig erkannt wird. Deshalb haben wir uns dafür entschieden, die Spule und die NFC-Tags in den Vitrinen so weit unten wie möglich zu platzieren. Da eines der Bauteile in der Mitte platziert werden sollte, muss es von dem anderen Bauteil eine höhere Anzahl geben und um die Mitte platziert werden. Dabei haben wir uns aus Kosten- und Stromübertragungsgründen dafür entschieden, die Spule in der Mitte zu platzieren und 6 kostengünstige NFC-Tags um diese Spule zu platzieren. Ein anderer Lösungsweg wäre gewesen, dass der Benutzer die Vitrine immer in einer bestimmten Position auf den Sockel stellen muss. Das kam für uns aber nicht in Frage, weil das Modell für den Benutzer so einfach wie nur möglich zu nutzen sein sollte.

3.3 Generierung der Inhalte für die Projektion

Die Bilder, die für die Projektion genutzt werden basieren auf den selben OpenStreetMap Daten, welche auch für den 3D Druck des Stadtmodells genutzt werden. Mithilfe der Python-Bibliothek <https://github.com/gboeing/osmnx> lassen sich die Daten des OSM-Exports einlesen und visualisieren.

Da die Bibliothek vor allem auf die Analyse von Straßennetzwerken spezialisiert ist, stellte sich die Anpassbarkeit der erzeugten Bilder als nur unzureichend heraus. So konnte man z.B. keine Straße farbig hervorheben. Auch zwischen Typen von Straßen unterscheidet osmnx nicht, was zu Ergebnissen geführt hat, in denen 3-spurige Hauptstraßen gleichgroß wie Fußwege dargestellt wurden.

Diese Probleme haben zu weiteren Tests geführt. Letztendlich nutzen wir osmnx nur noch, um die Daten einzulesen, anschließend wird daraus ein Pandas DataFrame. Dieses DataFrame lässt sich sehr einfach manipulieren. So kann man sich hier jetzt z.b. nur die Straßenabschnitte rausfiltern, welche zu der Busroute gehören.

Anschließend generieren wir mit Hilfe von Matplotlib Bilder aus diesem Datensatz.

3.4 Die Busroute

Die Busroute zu markieren stellte sich, wie die Generierung der Bilder im Allgemeinen, als komplizierter als erwartet raus. So liebt osmnx beim Erstellen des DataFrames aus der OSM-Export Datei nur die Nodes ein und keine Relationen, dies sind beides unterschiedliche Datentypen, welche OSM nutzt. Dies führt dazu, dass die Busroute, welche als Relation in OSM abgelegt ist, nicht mit importiert wird. Die Lösung war in diesem Fall das manuelle Extrahieren der Node Ids aus der Relation, das Markieren der Route anhand dieser Node Ids und das Sortieren der Node Ids mittels selbst geschriebenen Scripts, welches die Punkte über den euklidischen Abstand sortiert (siehe <https://github.com/LucaKrause1/CICISerlohn/blob/main/PiScripts/Projection/generateImg/OSMPlotTest2.ipynb>).

3.5 Positionierung der Projektion

Nachdem wir circa 9 Platten des Stadtmodells gedruckt hatten, haben wir mit den ersten Tests begonnen. Es hat sich sehr schnell rausgestellt, dass das Anpassen der Projektion durch Bewegen des Beamers sehr arbeitsintensiv ist und wir eine Möglichkeit benötigen, den Beamer millimetergenau im Raum zu platzieren. Da so eine Anforderung schwierig zu erfüllen ist, haben wir uns dazu entschieden, ein Kalibrierungs-Programm zu schreiben (mehr dazu im Abschnitt zur Kalibrierung (4.5.2)).

3.6 Größe der Vitrinen

Bei der Auswahl des Durchmessers der Plexiglasröhren galt es einige Punkte zu beachten. Zum einen sollten die Vitrinen nicht zu groß werden, da sie sonst zu viel Platz wegnehmen und vor Allem zu schwierig in der Handhabung werden. Zum Anderen muss genug Platz in der Vitrine sein um die gesamte Elektronik unterbringen zu können. Hier sind wir im ersten Schritt der Planung noch davon ausgegangen, dass der NFC-Reader in den Vitrinen angesiedelt ist und erkennt, auf welchem Sockel die Vitrine steht. Dies hätte den Vorteil gehabt, dass kein ESP32 in dem Sockel benötigt worden wäre. Im ersten Moment haben wir diesen Vorteil gesehen. Bei Näheren Überlegungen haben wir uns dann aber für die Platzierung des NFC-Readers im Sockel entschieden und damit für die Platzierung der NFC Tags in den einzelnen Vitrinen. Die Vorteile sind nun zum einen die Platzersparnis in der Vitrine, sowie eine einfachere Konstruktion der Vitrine. Außerdem ist es so besser modularisiert. So hat der ESP32 in der Vitrine nur die Funktion diese zu betreiben und Daten zu empfangen und diese, wie auch immer geartet, anzuzeigen. Der ESP32 in dem Sockel kann sich so ausschließlich um den NFC-Reader kümmern und die Daten entsprechend kommunizieren. Nach dieser Entscheidung musste als Nächstes geklärt werden, wie viele NFC Tags für eine komplette Abdeckung benötigt werden. Wie bereits Diskutiert, werden die NFC-Tags in einem Kreis um die Spule angeordnet. Wir haben nun getestet wie weit die Distanz von Kreismittelpunkt des NFC-Tags zu dem Mittelpunkt des NFC-Reader sein darf, damit immer noch zuverlässig gelesen wird. Als Ergebnis haben wir hier eine Obergrenze von ca. 25mm. Anhand dieses Wertes haben wir uns für 6 NFC Tags entschieden, wie in der Abbildung 5 zu sehen.

3.7 Vitrinen Spannung

Bei der Vitrine mit dem Baum ist uns aufgefallen, dass sie nur unzuverlässig funktioniert. Die LED's leuchten teilweise nicht und teilweise nur in einer Farbe, obwohl sie kontinuierlich die Farbe wechseln sollten. Als Erstes haben wir Wackelkontakte und Spannungsproblem in der Versorgung der Versorgerpule ausgeschlossen. Hier war soweit alles normal. Anschließend haben wir Software Probleme ausgeschlossen. Als nächstes haben wir ein Oszilloskop an die Empfängerspule angeschlossen und damit gleichzeitig die Eingangsspannung des ESP32 gemessen. Der folgende Verlauf hat sich daraus ergeben: Wie man sehen kann, steigt die Spannung linear an. Die Vermutung ist nun, dass der ESP32 sich zu lange in einem undefinierten Spannungsbe-

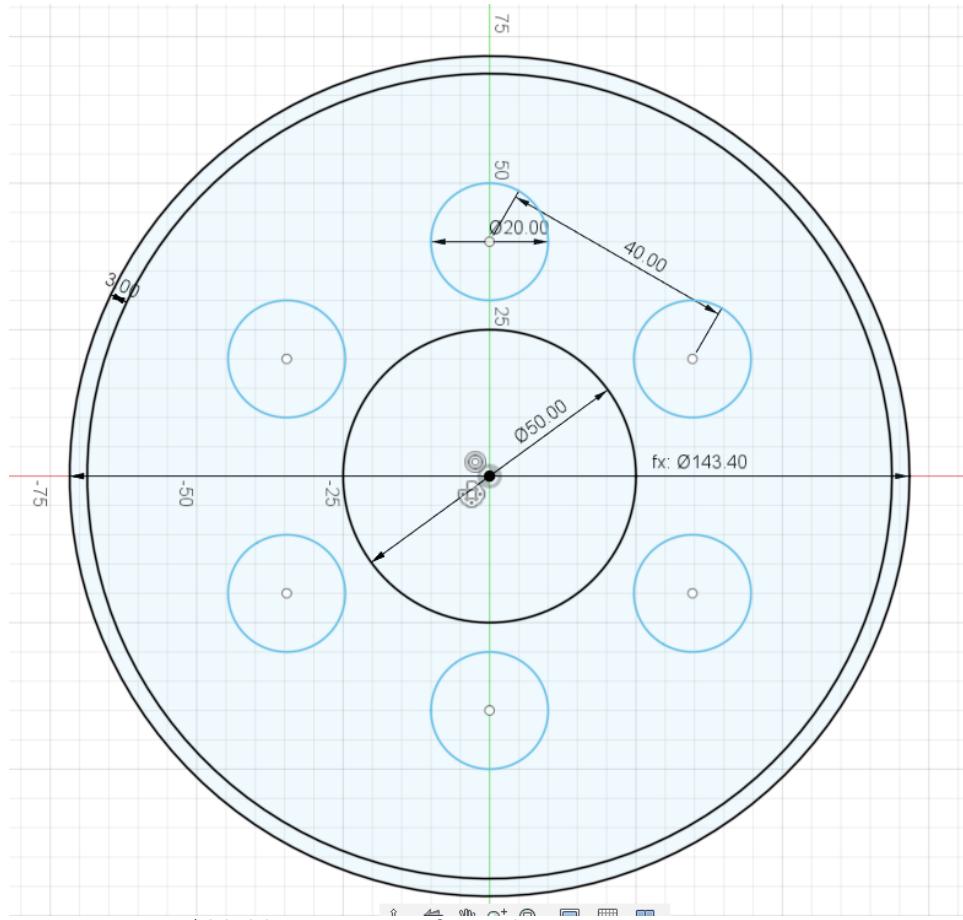


Abbildung 5: Bemaßung der einer Basis-Vitrine

reich befindet und damit in ein Brownout läuft. Dafür spricht auch, dass der ESP32 nach Betätigung des Reset Buttons normal läuft. Außerdem spricht noch dafür, dass der ESP32 tatsächlich den GPIO-Pin nicht schaltet, also vermutlich gar kein Programm Ablauf stattfindet. Dies haben wir ebenfalls mit dem Oszilloskop überprüft. Wir vermuten, dass eine Vorschaltung, welche erst ab knapp 5V durchschaltet, das Problem löst. Allerdings hatten wir zu dem Zeitpunkt dieser Erkenntnis nicht mehr genug Zeit bis zum Ende des Projektes um diese Verbesserung zu testen und umzusetzen.

3.8 Herausforderungen bei der interdisziplinären Arbeit

Die Herausforderungen der interdisziplinären Arbeit bestehen vor allem dar-aus, dass die Personen aus verschiedenen Disziplinen andere Vorstellungen vom Endprodukt haben. Dabei wollen zum Beispiel Designer gerne Flache, dünne Objekte erstellen, in die aber leider keine Technik verbaut werden kann.

An wirkliche Probleme sind wir allerdings nicht gestoßen, da wir das Konzept gemeinsam entwickelt und weiterentwickelt haben. Hatten verschie-dene Personen eine neue Idee, wurde sie allen Teammitgliedern vorgestellt und geschaht, ob diese realisierbar ist. Sollte die Idee nicht in dieser Art und Weise realisierbar sein, haben wir geschaht, ob es Kompromisse irgendwel-cher Art gibt, um die Idee vielleicht doch zu integrieren. Als Beispiel dafür gibt es zum Beispiel die Betonplatte auf dem Sockel. Da durch die Platte des Sockels die RFID-Kommunikation und die Stromübertragung läuft, muss die Platte zumindest in der Mitte sehr dünn sein. Dabei haben wir uns darauf geeinigt, ein Loch in der Mitte zu lassen und eine Erhebung hineinzubauen, auf der dann die Vitrinen stehen können und die RFID-Kommunikation und die Stromübertragung vernünftig funktioniert.

4 Ergebnis

4.1 Stadtmodell

Das Stadtmodell ist 3D-gedruckt und ungefähr 80cm x 80cm groß. Es stellt einen Ausschnitt der Iserlohner Innenstadt dar. Genauer ist es der Ausschnitt mit den Koordinaten von 7.68292 O bis 7.70601 O und von 51.36682 N bis 51.38097 N. Diesen Ausschnitt haben wir gewählt, da in der Südwestlichen Ecke des Ausschnittes die Fachhochschule liegt und in der Nordöstlichen Ecke das Wald | Stadt | Labor Iserlohn. Das Stadtmodell basiert auf den Daten von OpenStreetMap. Wir haben das Blender Plugin "blender-osm"<https://github.com/vvoovv/blender-osm> genutzt, um das Modell zu erstellen. Das generierte Modell ist eine simplifizierte Version der Realität. So sind die Gebäude fast ausschließlich die jeweiligen Grundrisse in die Höhe gezogen.

Anschließend haben wir die Datei manuell für den 3D-Druck vorbereitet. Hierfür haben wir das Modell in 25 Teile geteilt, die maximal 15cm x 15cm groß sind. Dies ist notwendig, um die Teile auf unseren beiden Druckern drucken zu können. Nun folgte der Druck.



Abbildung 6: Finales Stadtmodell mit Infofläche

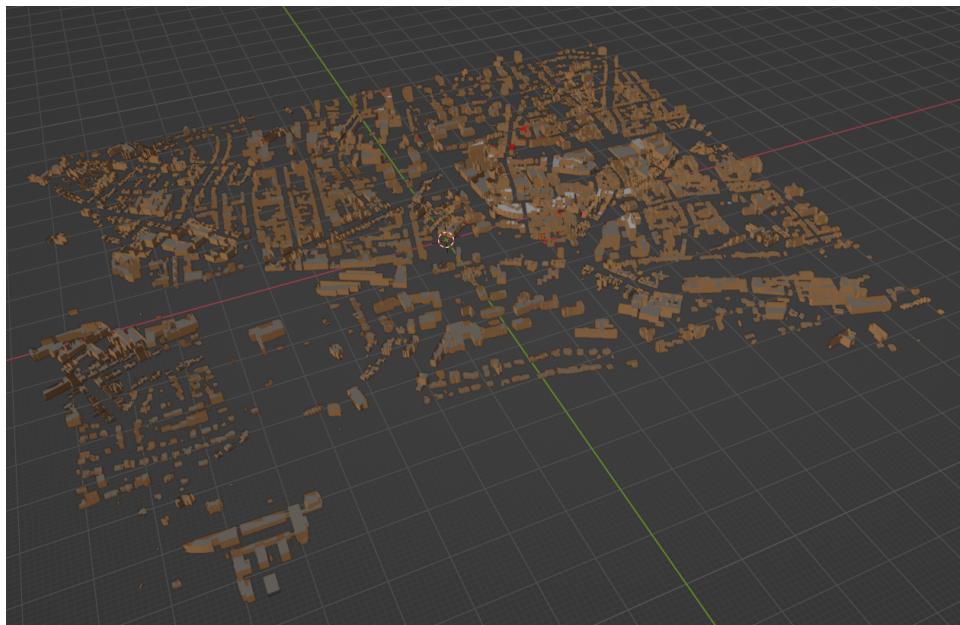


Abbildung 7: Stadtmodell in Blender

Die 25 Teile des Stadtmodells sind unter <https://github.com/LucaKrause1/CICIserlohn/tree/main/3D-Komponenten/Stadtmodell> zu finden.

4.2 Infrastruktur

4.2.1 Diskussion Technik Auswahl

In den Vitrinen und im Sockel werden jeweils Mikrocontroller benötigt, um die von uns angedachten Aufgaben auszuführen. Außerdem müssen die Mikrocontroller mit einem zentralen Punkt kommunizieren können, besonders der Mikrocontroller im Sockel, um zu signalisieren wann eine Vitrine auf diesen gestellt wird und welche.

Wir haben uns bei den Mikrocontrollern für ein ESP32 Development Board entschieden, genauer das ESP32-DevKitC-32E. Diese Entscheidung ist auf Grundlage der bereits beschriebenen Anforderungen und unter folgenden weiteren Einschränkungen gefallen.

Der größte Faktor ist hier die Erfahrung mit diesen Boards gewesen. Wir haben beide schon Erfahrungen mit sehr ähnlichen Boards und noch tiefer gehende Erfahrung mit der Arduino IDE, welche zum Programmieren der Development Boards genutzt werden kann.

Zum Einen musste das Development Board natürlich (schnell) bestellbar sein. Da wir bereits andere Teile auf der Bestellliste hatten, die wir bei <https://www.mouser.de/> bestellen wollten, haben wir als erstes bei Mouser nach ESP32 Development Boards gesucht und sind schnell fündig geworden. In der großen Auswahl an ESP32 Development Boards bei Mouser mussten wir uns erst einmal zurecht finden. Letztendlich haben wir uns, wie bereits erwähnt, für das ESP32-DevKitC-32E Board entschieden. Es erschien uns als am ähnlichsten zu den Boards, mit denen wir bereits Erfahrungen gemacht haben und hat wenig Besonderheiten.

Um uns nicht an Besonderheiten unterschiedlicher Boards anpassen zu müssen, haben wir für alle Komponenten, welche einen Mikrocontroller benötigen, den ESP32-DevKitC-32E benutzt. Auch wenn diese nicht zwingend nötig gewesen wären, weil z.B. die Kommunikation gar nicht genutzt wird.

Um den Beamer anzusteuern, fiel unsere Wahl auf einen Raspberry Pi 3B+. Diese Entscheidung ist unter folgenden Anforderungen getroffen worden. Zum Einen musste das hier eingesetzte Board/PC in der Lage sein einen Beamer anzusteuern, am besten per HDMI. Zum Anderen muss es auch in der Lage sein mit den ESP32 Boards zu kommunizieren und zuletzt muss es genug Rechenleistung haben, um mit OpenCV Bilder zu manipulieren, da

dies für die Projektion zwingend nötig ist.

Das spezielle Modell haben wir gewählt, weil Markus noch einen in seinem Büro verfügbar hatte und es ansonsten starke Lieferengpässe bei allen Raspberry Pi's gab.

Aus diesen Entscheidungen hat sich der folgende Aufbau ergeben:

- 1 ESP32-DevKitC-32E pro Vitrine
- 1 ESP32-DevKitC-32E für den RFID-Reader in dem Sockel
- 1 ESP32-DevKitC-32E für die LEDs im Sockel
- 1 Raspberry Pi für die Ansteuerung des Beamers und als MQTT Server

4.2.2 Kommunikation

Der Raspberry Pi agiert im STA Modus, er spannt ein WiFi Netz auf, in das sich alle ESP's einloggen. Außerdem läuft auf dem Pi der Mosquitto MQTT Broker, bei dem alle Teilnehmer des MQTT-Netzwerkes bestimmte Topics abonnieren und in diesen ihre Daten veröffentlichen können. Als Beispiel haben wir hier den ESP32, der an den RFID-Reader angeschlossen ist. Hat dieser einen neuen NFC-Tag entdeckt, veröffentlicht er die Information, zu welcher Vitrine dieser Tag gehört, in diesem Topic. Für MQTT haben wir uns entschieden, da dieses Protokoll bekannt ist, gut dokumentiert und es Bibliotheken für Python und Arduino gibt.

4.3 Sockel

Wie auch schon in den Abbildungen 3 und 4 zu sehen, besteht der Sockel sowohl aus einem drei-geteilten Holzstamm als auch aus einer mit Beton beschichteten Holzplatte. In diesem Sockel gibt es zwei ESP32 Boards. An den einen ESP32 ist ein LED-Streifen angebracht, der den Sockel von innen beleuchtet. An den anderen ESP32 ist der RFID-Reader (siehe Abbildung 8). Dieser Reader prüft, ob sich NFC-Tags in der Nähe befinden und sendet diese Information weiter. Dies macht er, indem er in einer Schleife überprüft, ob und welcher NFC-Tag in der Nähe ist:

```
1 if (mfrc522.PICC_IsNewCardPresent()
2     && mfrc522.PICC_ReadCardSerial()) {
3     //Überprüfung der Karte
4 }
5 delay(100);
```

Nach der Überprüfung der Karte, ob eine neue Karte gefunden wurde, wird in einer Tabelle nachgeschaut, welcher Vitrine dieser NFC-Tag zugewiesen wurde. Der Code ist aufs Wesentliche reduziert:

```

1 //Alle UID's der NFC-Tags
2 std::string UIDs[DIFFERENT_TAGS][NUMBER_TAGS] = {
3     {//A-Bus
4         "04 09 8D 15 39 6C 80",
5         "04 59 E7 14 39 6C 80",
6         "04 52 A4 15 39 6C 80",
7         "04 DE F5 05 49 6C 80",
8         "04 36 E7 14 39 6C 80",
9         "04 F3 74 15 39 6C 80"},
10    {//Baum
11        "04 4C 77 15 39 6C 80",
12        "04 23 B7 14 39 6C 80",
13        "04 BC EC 14 39 6C 80",
14        "04 B3 33 15 39 6C 80",
15        "04 95 FA 14 39 6C 80",
16        "04 7B C5 14 39 6C 80"}
17    };
18
19 for (int i = 0; i < DIFFERENT_TAGS; i++) {
20     for (int j = 0; j < NUMBER_TAGS; j++) {
21         if (s == UIDs[i][j]) {
22             showcase = i + 1;
23         }
24     }
25 }
26 switch (showcase) {
27     case 1:
28         res = "abus";
29         break;
30     case 2:
31         res = "tree";
32         break;
33     default:
34         break;
35 }
```

Das Resultat, also äbus”, „baumöder „none“ wird dann über das vom Raspberry Pi aufgesponnene Wifi-Netz über das MQTT-Protokoll an den Raspberry Pi gesendet, damit dieser die projizierten Folien ändern kann.

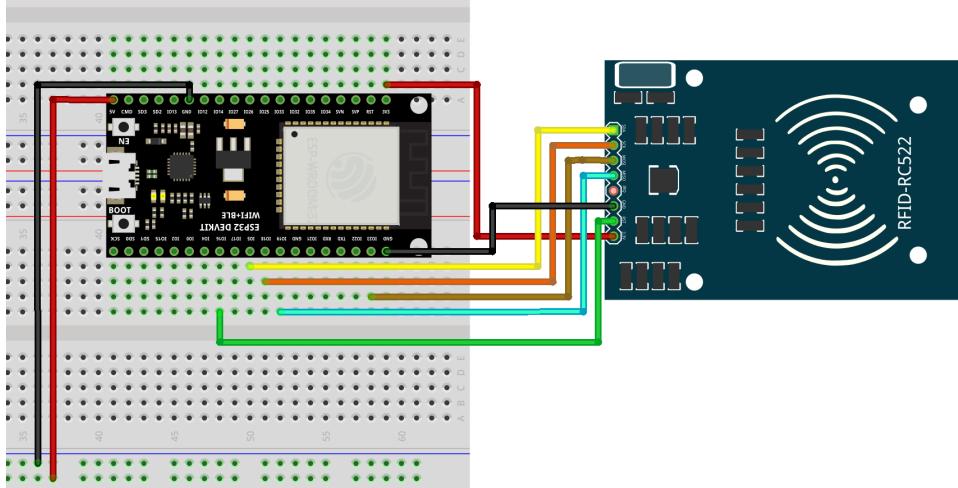


Abbildung 8: Schaltplan des Sockel-ESP32

Damit die Vitrinen, die auf den Sockel gestellt werden können, Strom bekommen, gibt es in der Mitte der Beton-Platte außerdem eine Spule zur Spannungsübertragung. In diesem Fall handelt es sich um den Transmitter. Die Spule ist mittig im Sockel platziert und der RFID-Reader so, dass der Mittelpunkt des Readers sich genau unter dem Kreis der NFC in der Base befindet, wenn diese mittig auf dem Sockel steht.



Abbildung 9: Loch in der Betonplatte mit Konstruktion für Spule und RFID-Reader

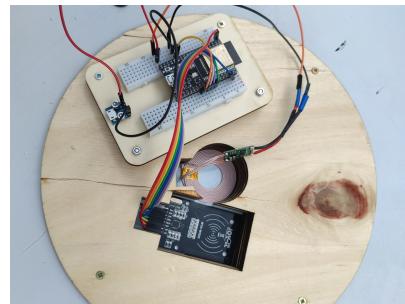


Abbildung 10: Technik unter der Betonplatte

Um die Technischen Komponenten in der Betonplatte zu befestigen bzw. unterzubringen, haben wir mit den Designern zusammen vereinbart, dass in der Mitte der Betonplatte ein Loch mit einem Durchmesser von 16cm frei bleibt. Hier haben wir nun eine mithilfe des Lasercutters hergestellte Kon-

struktion eingebaut, in der die Spule und der RFID-Reader Platz finden. Der ESP32 und die Verkabelung sind von unten an die Betonplatte angebracht. Zuletzt wurde noch eine schwarze Plexiglasscheibe mit einem Durchmesser von 16cm und einer Dicke von 1mm auf die in Abbildung 9 zu sehende Konstruktion aufgesetzt, sodass diese bündig mit der Betonplatte abschließt und die Technik "verschwinden" lässt.

4.4 Vitrinen

Die Vitrinen (siehe Abbildung 11) sind zum Interagieren mit dem Nutzer erstellt worden. Sie können auf den Sockel gestellt werden, um Informationen anzuzeigen. In diesen Vitrinen gibt es immer ein bestimmtes Modell, zu dem Informationen rechts neben dem Stadtmodell und auf dem Stadtmodell angezeigt werden können. Als Grundmodelle gibt es in unserem Projekt einen Baum und den aBUS. Bei den Informationen könnte es sich auch um Live-Sensordaten handeln, die dann auch an die Vitrinen übertragen werden können, damit dort die Live-Daten in eine Handlung umgewandelt werden.

Die Vitrinen bestehen aus einem Basisteil, in dem die Grundtechnik einer jeden Vitrine verbaut ist. Diese Base der Vitrinen hat 7 Runde Aussparungen, eine Mittig für die Spule und 6 im Kreis darum für die NFC-Tags. Die Basis ist innen hohl um Platz für die Technik der Vitrinen zu bieten, wie den ESP32 und z.B. einem Schrittmotor. Auf diese Base können dann einfach Deckel aufgesetzt werden, die je nach Modell in der Vitrine angepasst sind. Die Basis ist außerdem so designt, dass die Plexiglasröhren auf die Basis gestellt werden und unten bündig abschließen. Diese bildet so eine Ummantlung, damit der Inhalt der Vitrine geschützt bleibt.

Die Grundtechnik besteht immer aus 6 NFC-Tags, die dem RFID-Reader des Sockels als zu einer bestimmte Vitrine zugehörig bekannt sind. Dadurch kann jede Vitrine eindeutig erkannt werden, da NFC-Tags normalerweise eine eindeutige ID besitzen (kein anderer NFC-Tag besitzt die gleiche ID).

Um in der Vitrine selbst Bewegung oder andere Effekte, wie Beleuchtung bzw. Licht, zu ermöglichen, sobald diese auf den Sockel gestellt wird, benötigt sie Strom bzw. Energie. Hier haben wir uns für kontaktlose Energieübertragung durch Spulen entschieden.

Die naheliegende Alternative wären Akkus gewesen, dies würde allerdings den Arbeitsaufwand beim Programmieren, sowie die Wartung durch das ständig aufladen deutlich erhöhen. Außerdem müsste die gesamte Konstruktion darauf ausgelegt sein, dass man die Akkus laden kann. Das würde die Konstruktion zusätzlich verkomplizieren.

Unsere Anforderungen an Spule waren, dass sie genug Strom für den

ESP32, sowie z.B. einen kleinen Schrittmotor liefern muss (wir sind von ca. 1A ausgegangen) und das sie eine Übertragungsdistanz von mindestens 3mm hat. Die Wahl fiel auf die Spulen "106990017" mit dem Namen "Wireless Charging Module - 5V 1A" von Seeed Studio. Diese Entscheidung ist vor allem aus Mangel an Alternativen gefallen. Mouser.de hat als Wireless Charging Modul nur diese Spulen im Angebot. Nichtsdestotrotz erfüllt sie laut Datenblatt alle unsere Anforderungen, allerdings entspricht dieses Datenblatt keinerlei Anforderungen für Datenblätter, weswegen wir den Angaben gegenüber skeptisch eingestellt waren. In den ersten Tests, haben sie sich als unseren Anforderungen entsprechen rausgestellt und damit mussten wir uns nicht weiter nach Alternativen umsehen. Im späteren Verlauf sind uns allerdings doch noch Mängel aufgefallen, wie in 3.7 beschrieben. Die Spule in den Vitrinen ist der Receiver.

Durch diese Spule können jetzt zum Beispiel visuelle Effekte erzeugt werden, damit der Nutzer ein Feedback der Vitrine hat, dass die Interaktion funktioniert hat. Wie bereits erwähnt, habe wir uns aufgrund der Einheitlichkeit dafür entschieden, in jedem Basisteil der Vitrine einen ESP32 zu platzieren, damit dieser zum Beispiel Live-Informationen entgegennehmen kann, um diese direkt in eine Handlung umzusetzen.

4.4.1 Konstruktion

Für die Konstruktion der Vitrinen haben wir Fusion 360 genutzt. Das Repository zum Projekt enthält die konstruierten Objekte unter (<https://github.com/LucaKrause1/CICISerlohn/tree/main/3D-Komponenten>).

Wir haben uns für Fusion360 entschieden, weil wir schon mit der Software vertraut waren und es ein gut für diesen Einsatzzweck geeignet ist.

4.4.2 Baum

Das Baummodell in der Vitrine ist 3D-gedruckt. Wir haben den Baum in Blender erstellt und ihn anschließend als .obj Datei exportiert. Als nächstes haben wir das Modell in Fusion 360 importiert. Hier haben wir nun das Modell von einem Netzkörper in einen Körper konvertiert. Im nächsten Schritt haben wir die Baumkrone vom Stamm getrennt und die Baumkrone ausgehölt. Als letztes haben wir ein Kabelkanal in den Baumstamm modelliert und den entsprechenden Deckel für die Base, mit Loch an der selben Stelle erstellt. Anschließend haben wir einen WS2812b LED Streifen mit 7 LEDs zu einem Kreis geformt, die 3 Kabel dazu angebracht und dann den Ring befestigt, sodass die LEDs alle nach außen, die Innenwand des Baumes



Abbildung 11: 2 fertige Vitrinen

anstrahlen. Die Helligkeit ist hierbei für Innenräume sehr gut und die Dicke der Baumkrone ist auch gut, so leuchtet das Licht gleichmäßig

Bei den LEDs haben wir uns für WS2812b LED Streifen mit 60 LEDs/m entschieden. Dieses Streifen haben wir benutzt, da sie mit 5V betrieben werden und durch nur eine Datenleitung angesteuert werden. Diese Datenleitung sollte zwar eigentlich mit 5V betrieben werden, allerdings funktioniert es auch mit den 3,3V des ESP32 zuverlässig. Außerdem kennen wir uns gut mit den LED Streifen und der Ansteuerung aus. Gesteuert werden sie von dem ESP32 (siehe Abbildung 12), der sich in der Basis der Vitrine befindet.

Mit dieser Farbe könnte in Zukunft der Zustand eines bestimmten Baumes in der Nähe angezeigt werden, an den verschiedene Sensoren wie ein Feuchtigkeitssensor angebracht sind. Die Informationen eines solchen Sensors werden in Zukunft über LoRaWAN an den Raspberry Pi oder einen zentralen Server der Stadt Iserlohn übertragen, von dem sich der Raspberry Pi diese Informationen holen kann. Aus diesen Informationen können nun Schlüsse daraus gezogen werden, wie gut es dem Baum geht. Hat der Baum schon lange kein Wasser mehr bekommen? Ist das Wasser und die Luft,

welche er bekommt von toxischer Natur, weil viele Autos vorbeifahren, die Abgase in die Luft abgeben und Reifenabrieb produzieren, der durch das Regenwasser aufgenommen wird?

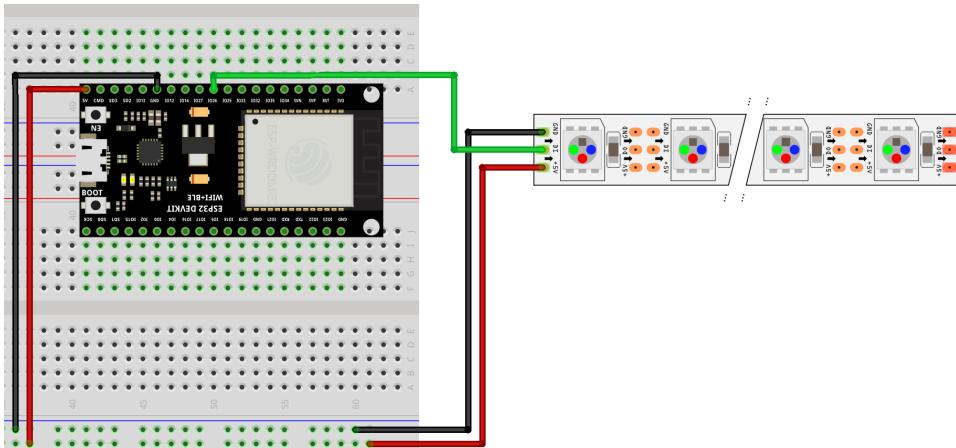


Abbildung 12: Schaltplan der Baum-Vitrine

Die zum Baum zugehörige Projektion ist ein Stadtteil der Stadt Iserlohn, in der alle Grünflächen grün und die Straßen alle in einer Farbe markiert sind. Der südlichere Teil der Stadt beinhaltet wesentlich mehr Grünflächen, weil sich hier der Wald befindet.

4.4.3 aBUS

Der aBUS als zweites Beispielmodell ist ein Modell des in Iserlohn fahrenden autonomen Busses. Dieser fährt zwischen der Fachhochschule von Iserlohn und dem Iserlohner Hauptbahnhof, weshalb diese beiden Orte auch ein Teil der Entscheidung dafür waren, den ausgewählten Abschnitt der Stadt Iserlohn zu nutzen.

Die Projektion zum aBUS ist ein Straßenbild der Stadt Iserlohn, in welcher eben diese Buslinie verstrkkt zu sehen ist. Um die Fahrt zu simulieren, haben wir mehrere Punkte, die auf der Strecke liegen, und "fahren" sie nacheinander ab.

Der ABus in der Vitrine ist 3D gedruckt. Das Modell haben wir von dem Stadt | Wald | Labor Iserlohn erhalten. Hier mussten wir nur noch eine Aussparung für die Kopplung mit dem Motor erstellen. Dies ist in Blender geschehen, da, durch die Komplexität des Modells, eine Umwandlung in einen, in Fusion 360 bearbeitbaren Körper nicht möglich war.

In der Basis der Bus-Vitrine befindet sich ebenfalls ein ESP32, welcher einen Motortreiber steuert, der wiederum einen Schritt-Motor ansteuert (siehe Abbildung 13). Der Schrittmotor dreht sich im normalen Modus ein bisschen in die eine Richtung und wechselt nach einer kurzen Pause wieder in die andere Richtung.

In Zukunft könnten auch hier Live-Daten des Busses verwendet werden, um die Fahrtrichtung durch die Drehrichtung des Busses anzuzeigen. Dafür wird noch ein Kompassmodul benötigt, um herauszufinden, in welche Richtung sich der Bus drehen muss. Zusätzlich dazu bräuchte man noch einen Drehgeber, um die Position des Schrittmotors herauszufinden.

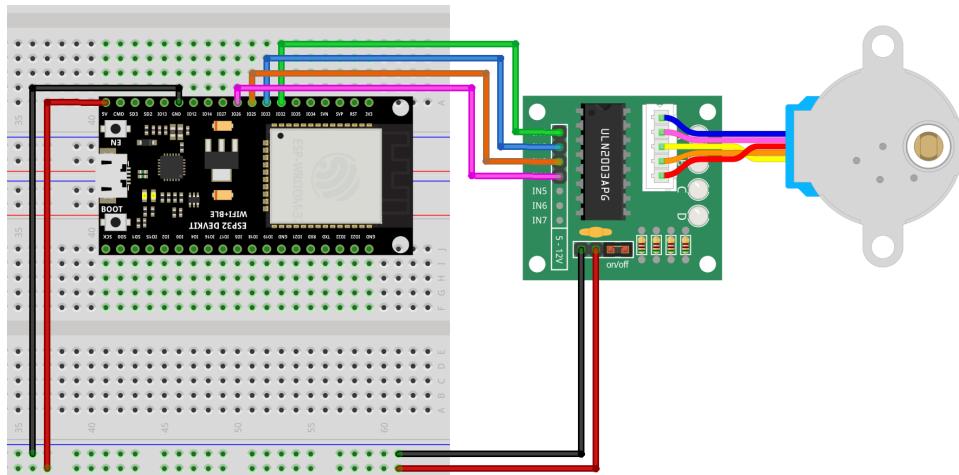


Abbildung 13: Schaltplan der aBUS-Vitrine

4.5 Projektion

4.5.1 Generierung der Inhalte

Die Generierung der Bilder erfolgt, wie bereits erwähnt, in Python mit Hilfe von osmnx, Pandas und Matplotlib. Damit man schnell und einfach Bilder generieren kann, haben wir ein Script entwickelt, welches eine Konfigurations-Datei liest. In diesen Konfigurations-Dateien sind jeweils angegeben, welche Kategorie von Straße, welche Farbe und Breite haben soll. Außerdem kann in dem Script direkt mit angegeben werden, ob die Umrisse der Gebäude auch dargestellt werden sollen und wenn ja, in welcher Farbe.

An dieser Stelle gibt es noch viele andere Möglichkeiten, was visualisiert werden kann. So haben wir für den Baum ein Bild erstellt, auf dem alle

Grünflächen der Stadt grün markiert sind. Je nach Anwendung kann also dieses Script, welches in gewisser Weise das Grundgerüst bildet, erweitert werden.

4.5.2 Kalibrierung

Um die Kalibrierung durchführen zu können, muss der Beamer erst in einem solchen Abstand vor dem Modell aufgestellt/aufgehängt werden, sodass alle Ecken der Projektion des Beamers gerade so um die zwei linken Ecken des Stadtmodells und die zwei rechten Ecken der Informationsfläche reichen. Nun kann das Kalibrierungs-Programm gestartet werden (Alle vier Ecken werden angestrahlt).

Das Programm haben wir wieder in Python geschrieben und OpenCV genutzt. OpenCV liefert auch direkt Funktionen für Perspektiven-Korrektur mit, genauer sind dies:

```
cv2.getPerspectiveTransform(oP,pL)  
  
cv2.warpPerspective(pro,M,(800, 800),flags=cv2.INTER_LINEAR)
```

Diese Funktionen führen eine Transformation wie in Abbildung 14 durch.

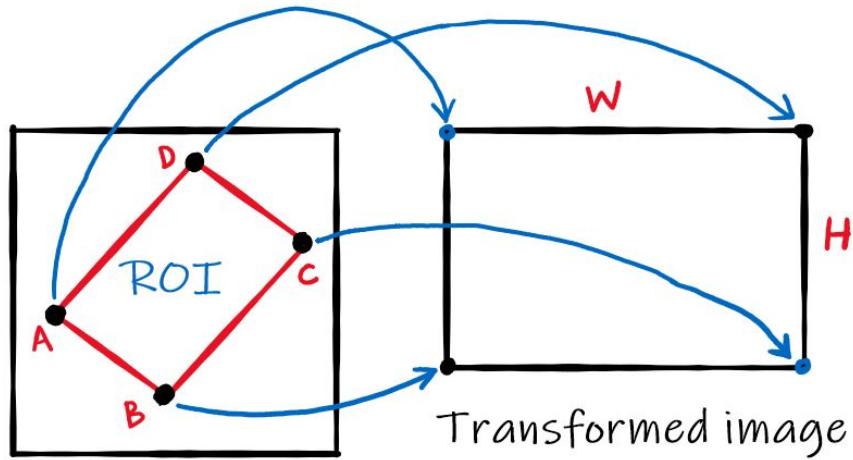
Diese Korrektur basiert auf 4 Punkten im Ausgangsbild, die auf 4 Punkte im Zielbild projiziert werden.

Das Kalibrierungs-Programm funktioniert nun wie folgt: Es sind 4 Hau-secken mit roten Punkten markiert. Es müssen nun die entsprechenden Hau-secken im 3D-gedruckten Stadtmodell in dieser festgelegten Reihenfolge angeklickt werden:

1. Rechts oben
2. Links oben
3. Links oben
4. Rechts oben

Alle gesetzten Punkte sind nun grün markiert (wie in Abbildung 15) und der jeweils letzte gesetzte Punkt lässt, wie in Abbildung 17 zu sehen, sich mit den Tasten i, j, k und l verschieben.

Per Rechtsklick lässt sich der letzte Punkt löschen. Wenn man die A Taste drückt werden in der Konsole die Koordinaten der 4 markierten Punkte



Input image

Abbildung 14: Anwendung der Transformation

https:

//i0.wp.com/theailearner.com/wp-content/uploads/2020/11/perspective2-1.jpg

ausgegeben. Wenn man die P Taste drückt wird die Transformation einmal auf das Bild angewendet und man kann überprüfen ob die Kalibrierung gut durchgeführt wurde. Das Ergebnisbild könnte wie in Abbildung 16 aussehen. Die Werte aus der Konsole (siehe Abbildung 18) müssen nun in den Code in der "MainLogic.py" kopiert werden. Wurde das Stadtmodell angepasst, werden die Daten in "calibPoints" kopiert, wurde die Infofläche angepasst, werden die Daten in "calibPointsInfo" kopiert (siehe Abbildung 19). `[[x,y],[i,j],[u,v],[f,g]]` Wichtig ist hierbei, nicht die `origPoints` zu ändern. Die Schritte müssen nun noch einmal wiederholt werden für die Info Fläche. Die Unterschiede sind hier, dass man die Projektion nicht mit der P Taste überprüfen kann, und auch die roten Punkte existieren hier nicht. Es müssen einfach die 4 Ecken der weißen Informationsfläche ausgewählt werden.

Die Kalibrierung funktioniert sehr gut, wie in der Abbildung 20 zu sehen ist. Dabei stellen die gelben Flächen immer die Grundfläche von Gebäuden dar, die sehr genau auf den Oberflächen der Gebäude des Stadtmodells sitzen. Werden die Häuserecken sehr genau ausgewählt, kann das Ergebnis der Kalibrierung perfekt auf dem Stadtmodell sitzen.



Abbildung 15: Projektion mit 8 Kalibrierungspunkten



Abbildung 16: Ergebnis der angewandten Kalibrierung

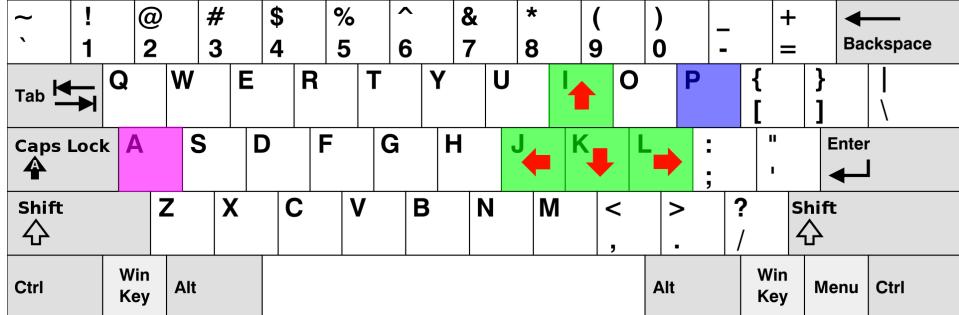


Abbildung 17: Tastenbelegung zu Kalibrierung

```
um/Semester6/CrossInnovationClass/CICIserlohn/PiScripts/Projection/calibration.py
[[706, 183], [94, 197], [77, 698], [708, 731]]
```

Abbildung 18: Werte aus der Konsole

```
38 origPoints = [[743, 139], [84, 177], [67, 710], [722, 757]]
39 calibPoints = [[754, 169], [117, 220], [99, 721], [746, 751]]
40
41 origPointsInfo = [[1279, 0], [800, 0], [800, 799], [1279, 799]]
42 calibPointsInfo = [[1222, 96], [827, 23], [884, 724], [1262, 786]]
```

Abbildung 19: Einzutragende Kalibrierungs-Punkte

4.6 Anzeige der Projektionsfläche

Die Entscheidung, was auf der Projektionsfläche angezeigt wird, läuft im Raspberry Pi ab. Dieser lauscht in einem zweiten Thread auf das Topic, in welchem vom RFID-Reader-ESP veröffentlicht wird:

```
1 def on_message(client, userdata, msg):
2     """The callback for when a PUBLISH message
3         is received from the server."""
4     print(msg.topic + ' ' + str(msg.payload))
5     global currentMode
6     if str(msg.payload) == "b'abus'":
7         currentMode = Mode.ABUS
8     elif str(msg.payload) == "b'tree'":
9         currentMode = Mode.TREE
10    elif str(msg.payload) == "b'none'":
11        currentMode = Mode.NO_MODEL
12    print(currentMode)
```

In der Hauptschleife werden diese empfangenen Vitrinen immer bearbeitet.



Abbildung 20: Ergebnis der Kalibrierung auf dem Stadtmodell

Dabei wird immer das zur Vitrine zugehörige Bild und mit seiner Informationsseite aus dem Speicher gelesen (Zeile 5 - 11) und mit der Größe des Beamers angepasst. Beim ABUS zum Beispiel gibt es, wie bei anderen Vitrinen vielleicht später auch, weil nicht viel Platz auf dem Informationsboard ist, mehrere Informationsseiten, die immer nach 100 Zyklen durchgetauscht werden (Zeile 15 - 24). Da bei dem ABus auch immer die aktuelle Position auf der Buslinie angezeigt wird, muss hier auch immer der Punkt transformiert und immer wieder in das Ergebnisbild eingebracht werden (Zeile 25 - 30). Im Anschluss daran werden die Transformationen des Stadtmodells und der Information durchgeführt, beide Bilder miteinander verbunden und an den Beamer übertragen (Zeile 31 - 36).

Der Code ist aufs Wesentliche reduziert und könnte an manchen Stellen verbessert werden, dies war aber aufgrund des Zeitmangels nicht mehr zu bewerkstelligen. Vor allem wird hier immer wieder das gleiche Bild neu geladen und transformiert, obwohl wir das Ergebnis der ersten Transformation zwischenspeichern könnten. Der Code ist auch unter <https://github.com/LucaKrause1/CICIserlohn/blob/main/PiScripts/Projection/MainLogic.py> zu finden.

```

1  while 1:
2      oldMode = currentMode
3
4      # Read the right image
5      map = cv2.imread(modeToImage[currentMode] )
6      map = cv2.resize(map, (800, 800), interpolation= cv2.INTER_LINEAR)
7
8      infoNew = cv2.imread(modeToInfo[currentMode] + str(infoNum)
9                          + ".jpg")
10     info = cv2.resize(infoNew, (480, 800), interpolation=
11                           cv2.INTER_LINEAR)
12
13     while oldMode == currentMode:
14         pro = map.copy()
15         if currentMode == Mode.ABUS:
16             infoCount+=1
17             # Change Info after x cycles
18             if infoCount > 100:
19                 infoCount = 0
20                 infoNum = (infoNum+1)%4
21                 infoNew = cv2.imread(modeToInfo[currentMode] + str(infoNum)
22                               + ".jpg")
23                 info = cv2.resize(infoNew, (480, 800),
24                                   interpolation=cv2.INTER_LINEAR)
25                 x,y = liste[idx]
26                 idx = (idx + 1)%len(liste)
27                 coords = realCoordsToPixelCoords((x,y),
28                                                 realCoordWindow,pixelCoordWindow)
29                 pro = cv2.circle(pro, coords, radius=6, color=(0, 0, 255),
30                                  thickness=-1)
31                 pro = cv2.warpPerspective(pro,M,(800, 800),
32                                           flags=cv2.INTER_LINEAR)
33                 infoDis = cv2.warpPerspective(info,Minfo,(480, 800),
34                                               flags=cv2.INTER_LINEAR)
35                 img = np.concatenate((pro, infoDis), axis=1)
36                 cv2.imshow("window", img)

```

5 Fazit

Wir haben umgesetzt, was wir im Konzept definiert haben. Ein paar kleine Probleme bestanden leider bei der Übergabe an das Wald | Stadt | Labor Iserlohn immer noch. Zum Einen das Problem mit der Baum-Vitrine,

dass die ansteigende Spannung den Mikrocontroller nicht zuverlässig starten kann, wenn die Vitrine langsamer in die Nähe der Transmitter-Spule kommt. Zum Anderen das Problem, dass der LED-Streifen im Baumstamm dauerhaft leuchtet, obwohl er dies nur tun sollte, wenn eine Vitrine auf dem Sockel steht. Außerdem hätten wir ein bisschen dickere Kabel nehmen sollen, weil die benutzen Kabel schon recht warm wurden, als die gesamte Baumstamm-Konstruktion für längere Zeit lief, was wir allerdings erst nach der Präsentation bemerkten.

Die Cross-Innovation-Class als Ganzes hat super funktioniert und auch viel Spaß gemacht. Vor allem der interdisziplinäre Austausch mit Studenten anderer Hochschulen/Universitäten/Fachbereiche hat neue Eindrücke gebracht. Es mussten Kompromisse eingegangen werden, damit alle drei Seiten Teile ihrer Vorstellungen umsetzen konnten.

5.1 Lesson's Learned

Wir haben auf jeden Fall gelernt, wie man damit umzugehen hat, wenn man mit Personen anderer Fachbereiche zusammenarbeitet. Es wird aufeinander gehört und die Komplikationen anderer und von einem selbst werden besprochen und Lösungen gesucht und zu einem großen Teil auch gefunden. Diese Lösungsfindung ist sehr wichtig, wenn man als Team gut vorankommen möchte.

Außerdem haben wir noch ein paar technische Komponenten oder Software-Konzepte kennengelernt, die wir noch nie genutzt hatten. Dazu gehört zum Beispiel das MQTT-Protokoll, der RFID-Reader und die Spannungsversorgung durch Spulen. Aber auch Dinge wie das Arbeiten mit OSM Daten und visualisieren dieser, sowie Anpassen von Projektionen

6 Ausblick

Im Großen und Ganzen ist das Modell einsatzbereit und befindet sich bereits in Iserlohn. Zum Einen können die Funktionen der vorhandenen Vitrinen erweitert werden, in dem Sinne, dass zum Beispiel die Baum-Vitrine mit einem Sensor eines echten Baumes verbunden wird, oder die Bus-Vitrine mit echten Daten des aBUS.

Zum Anderen können noch andere Vitrinen hinzugefügt werden, die zum Beispiel in Richtung Smart Home gehen, wie zum Beispiel eine LED, die über WLAN vom Raspberry Pi gesteuert werden kann.

Denkbar wäre auch eine Ampel-Darstellung, auf der auf dem Stadtmodell vereinzelte Autos zu sehen wären und immer eine perfekte Grünphase

erwischen sollen und deshalb ihre Geschwindigkeit dynamisch anpassen.

Eine weitere Verbesserung wäre es, die Benutzerfreundlichkeit der beiden Programme auf dem Raspberry Pi, vor allem der Kalibrierung, anzupassen. Es ist ein bisschen umständlich, die ausgewählten Punkte für die Transformation manuell in das Hauptprogramm zu kopieren. Um dieses Problem zu lösen, könnte man mit dem Kalibrierungs-Programm eine Konfigurationsdatei schreiben und diese dann im Hauptprogramm einlesen. Außerdem wäre es sinnvoll, dass sich die Programme über ein Desktop-Icon starten lassen könnten, um nicht die Kommandozeile benutzen zu müssen. Es ist auch einen Gedanken wert, die beiden Programme des Pi's zu verbinden und über einen bestimmten Tastendruck (z.B. Taste T) die Kalibrierung von vorher zu übernehmen oder erst durch einen Tastendruck zu aktivieren.

7 Repository

Das Repository beinhaltet den Code aller Komponenten. Zu diesen Komponenten gehören die vier ESP32 und der Raspberry Pi. Zudem sind hier die Bestelllisten und die 3D-Darstellungen der gedruckten Komponenten zu finden: <https://github.com/LucaKrause1/CICIserlohn>