Softwareprojekt
Prof. Dr. Ulrich Hoffmann

Projektbericht

Team Stadt Frankfurt

Cross Innovation Class 2022

Sven Hülsen, Robin von Berg, Celina Krug, Moritz Hillen,
Florian Bucher, Maybritt Braun, Lucas Below

15. Juli 2022

Inhaltsverzeichnis

Seite

1 Einleitung

Dieser Projektbericht ist im Rahmen der Cross Innovation Class 2022 entstanden.

1.1 Cross Innovation Class

Die Cross Innovation Class, kurz CIC, ist eine von der Hamburg Kreativ Gesellschaft organisierte Veranstaltung in Kooperation mit Universitäten und Fachhochschulen des Hamburger Umlands. Idee der CIC ist es das Studierende verschiedener Fachrichtungen unterschiedlicher Universitäten ein Semester lang in interdisziplinären Teams Projekte erarbeiten.
Teilnehmen konnten Studierende des Studiengangs Stadtplanung der Hafencity Universität Hamburg, der Studiengänge Produkt- und Interior-Designer der Akademie Mode & Design des Standorts Hamburg und der Studiengänge Informatik, Technische Informatik, Wirtschaftsinformatik, Smart Technology und IT-Ingenieurwesen der Fachhochschule Wedel.
Die Cross Innovation Class lief dabei dieses Jahr unter dem Thema Resilient Cities. In Bezug auf dieses Oberthema wurden von der Hamburg Kreativ Gesellschaft fünf Partnerunternehmen gefunden, die jeweils eine Fragestellung mit in die CIC gebracht haben.

Resilient Cities

Resilienz, ein wichtiger Faktor in vielen Lebensbereichen. Ein Attribut das Anpassungsfähigkeit und einen standhaften Umgang mit Krisen beschreibt. Neben persönlicher und ökonomischer Resilienz übernimmt Resilienz auch eine immer wichtiger werdende Rolle im Blick auf Gemeinden und Städte. Vorallem im Bezug auf Extremwetterereignisse und dem immer weiter voranschreitenden Klimawandel braucht es neue Ideen und Konzepte.

Daher gibt es viele Bestrebungen auf globaler, europäischer und nationaler Ebene dieses Thema voranzubringen. Eine Institution ist der Urban Resilience Club, der Urbane Resilienz wie folgt definiert:
„Urban Resilience - The measurable ability of any urban system, with its inhabitants, to maintain continuity through all shocks and stresses, while positively adapting and transforming toward sustainability.“[a]

[a] `https://urbanresiliencehub.org/what-is-urban-resilience/`

Partner dieses Jahr waren die Stadt Frankfurt mit der Stabsstelle Digitalisierung, die ACO Gruppe, Hamburg Marketing, das Hamburg Institute for Innovation, Climate Protection and Circular Economy (HiiCCE) und das Wald Stadt Labor Iserlohn.
Fünf Teams, jeweils bestehend aus Studierenden jeder Universität und einem Praxispartner durchliefen über knapp 12 Wochen ein Programm im Rahmen des Design Thinkings. Dieses Format stellte im Vergleich zu den sonst eher theoretischeren oder fachspezifischeren Veranstaltungen eine willkommene Ergänzung da, da wir vielen Aufgaben ausgesetzt waren, aus denen wir wertvolle Erfahrungen ziehen konnten, die uns im Arbeitsalltag begegnen werden, die wir ansonsten in unserem Studium aber nicht kennengelernt hätten.

1.2 Das Team

Unser Team bestand aus sieben Studierenden. Drei der Hafencity Universität des Studiengangs Stadtplanung, Celina Krug, Moritz Hillen und Florian Bucher, zwei Studierende der Akademie Mode & Design des Studiengangs Product Design, Maybritt Braun und Lucas Below und uns, Sven Hülsen und Robin von Berg, Informatikstudenten der FH Wedel. (Siehe Abbildung 1.1)
Aus der AMD unterstützt wurden wir ebenfalls von Annika Fröhlich, da an der AMD eine weitere interne Unterteilung in Gruppen angesetzte wurde, die gemeinsam mehrere Projekte (eins davon die CIC) bestritten haben.

1.3 Ablauf der CIC

Das Projekt wurde in drei große Phasen eingeteilt, Konzept, Entwurf und Prototyping. Neben einem KickOff zu Beginn gab es am Ende jeder Phase eine Feedback Runde mit der gesamten Class und zum Abschluss ein öffentliche Abschlussveranstaltung.
In der KickOff Veranstaltung haben sich die Praxispartner und ihre Fragestellung vorgestellt und wir haben unser Team kennengelernt.
Ergänzend wurde allen Interessierten am Ende der Class, vor der Abschlussveranstaltung ein sehr lehrreiches Pitch-Training angeboten.

Projektphasen

Termin/Phase	**Bezeichnung**
8. April 2022	Kick-Off
11. April 2022 - 21. April 2022	Analyse & Konzept Phase
25. April 2022 - 6. Mai 2022	Entwurfsphase
9. Mai 2022 - 23. Juni 2022	Prototyping & Modellbau Phase
30. Juni 2022	Abschlussveranstaltung

Abbildung 1.1: CIC Team Stadt Frankfurt
v.l.n.r.: Maybritt Braun (AMD), Annika Fröhlich (AMD), Robin von Berg (FHW), Lucas Below (AMD), Sven Hülsen (FHW), Celina Krug (HCU), Moritz Hillen (HCU), Jochen Schmitz (FES). Abwesend: Florian Bucher (HCU), Mechthild Schulze-Tenberge & Karina Mombauer (Stadt Frankfurt Stabstelle Digitalisierung)

FH Wedel JourFixe

Analog zur AMD und HCU hatten wir ein wöchentliches internes Meeting an der FH Wedel, das jeden Donnerstag um 17 Uhr stattgefunden hat. Ziel dieses Meetings war ein Statusbericht der jeweiligen Teams, um Themen zu identifizieren, die den Fortschritt des Projekts blocken, sowie ein Austausch unter den Teams bezogen auf technische Fragestellungen und die effiziente Kommunikation in den interdisziplinären Teams.

Gruppen & Praxispartner JourFixe

Zusätzlich haben wir uns intern im Team jeden Donnerstag um 8:30 Uhr getroffen um offene Fragen und anstehende Aufgaben zu besprechen. Um 10 Uhr sind unsere Praxispartner dazugestoßen, sodass wir aufgekommene Fragen klären konnten und einen Bericht über den aktuellen Stand und die anstehende Woche skizzieren konnten.

CIC CheckIn

Unregelmäßig am Mittwoch hat ein CIC-übergreifendes CheckIn stattgefunden, das unsere Projektleiterin besucht hat, auch hier ging es um ein Statusbericht und zusätzlich um anstehende Termine und Deadlines der CIC.

1.4 Unsere Praxispartner

Zu Beginn der CIC hat Mechthild Schulze-Tenberge als Ansprechpartnerin der Stadt Frankfurt am Main agiert. Frau Schulze-Tenberge übernimmt die Leitung der Stabstelle Digitalisierung und war diejenige, die uns die Aufgabenstellungen der Stadt Frankfurt präsentiert hat.
Im späteren Verlauf hat Karina Mombauer, ebenfalls aus der Stabstelle Digitalisierung, ihren Platz übernommen, da Frau Schulze-Tenberge andere Projekte verfolgen musste.

Als Unterstützung und Experte zum Thema Stadtreinigung ist Jochen Schmitz zum Projekt hinzugestoßen. Als Leiter des Innovationsmanagement der FES konnte er uns Einblicke in die alltäglichen Abläufe und Erfahrungen der FES, der internen Projekte bezüglich Sensorik und Kommunikationsansätzen geben, aber zusätzlich auch eine weitere Perspektive auf die vorhandenen IT-Projekte der Stadt Frankfurt.

Zusätzlich wurden wir von Dagmar Schöne unterstützt, die Teil des Teams-„#cleanFFM“ ist und somit die andere Seite, die Rolle der Auftraggeberin der FES übernommen hat.

1.4.1 Stadt Frankfurt am Main - Stabstelle Digitalisierung

Die Stabstelle Digitalisierung der Stadt Frankfurt am Main existiert seit 2021 und ist der Nachfolger der im Jahre 2013 gegründeten Stabstelle E-Government, die als Ziel die möglichst durchgehend elektronische Abwicklung von Verwaltungsvorgängen hatte.
Bei der Umstrukturierung zur Stabstelle Digitalisierung sind weitere Themen zum Aufgabengebiet hinzugekommen. In diesem Zuge wurde eine „Gesamtstädtische Digitalisierungsstrategie“ (**PDF**) entwickelt. Diese umfasst unter anderem eine „Urban Data Plattform“ und den flächenüberdeckenden Ausbau eines LoRaWAN Netzes.
Da uns von Frau Schulze-Tenberge drei Fragestellungen zur Auswahl gestellt wurden, die in verschiedene Fachbereiche reichen (siehe Abschnitt 2), haben wir nach unserer Wahl entsprechende fachliche Unterstützung von der FES bekommen.

1.4.2 Frankfurter Entsorgungs- und Service GmbH (FES)

Die Frankfurter Entsorgungs- und Service GmbH, kurz FES, ist das Frankfurter Äquivalent zur Stadtreinigung Hamburg. Das seit 1995 bestehende Unternehmen, vorher 100% in öffentlicher Hand, heute öffentlich privat, beschäftigt circa 1900 Mitarbeiter und bietet neben Entsorgungs- und Reinigungsdienstleistungen bspw. auch Verkehrsmaßnahmen

und ein Veranstaltungsservice an. Desweiteren hält die FES 50 Prozent der FFR GmbH, die das Müllheizkraftwerk Frankfurt am Main betreibt.

2 Projektfindung

Zu Beginn des Projekts wurden wir von unserem Praxispartner mit drei Fragestellungen konfrontiert, unter denen wir eine wählen konnten:

> Wie lässt sich der Verkehr für Einkäufe und Lieferungen reduzieren, um die Schadstoffbelastungen in der Luft zu minimieren?

> Wie lässt sich die Müllentsorgung in der Innenstadt und in den Grünflächen optimieren (z.B. Roboter, automatische Mülltrennung, Füllstandsensoren etc.)?

> Wie lässt sich Informations- und Kommunikationstechnik (IKT) (Rechenzentren, WLAN, Breitband) für eine umwelt- und ressourcenschonende Stadtgestaltung (z.B. Abwärme der Rechenzentren) nutzen?

Im Anschluss der KickOff-Veranstaltung haben wir uns für die zweite Frage, die das Thema Müllentsorgung thematisiert, entschieden, da wir dabei die größten Freiheiten bei der Gestaltung des Prototyps gesehen haben und Lust hatten uns mit der Problematik auseinanderzusetzen.
So haben wir in den kommenden Wochen Interviews mit unseren Praxispartnern geführt, nach schon existierende Maßnahmen und der aktuellen Lage vor Ort, sowie nach psychologischen Ursachen recherchiert.
So konnten wir unseren Problemraum immer größer gestalten, bis wir beim CIC Workshop vier Ideen vorgestellt haben. (**PDF der Präsentation**)

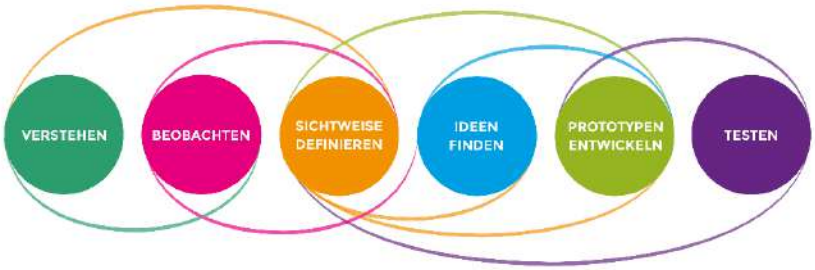


Abbildung 2.1: Der Design-Thinking-Prozess[1]

Design Thinking

Design Thinking ist ein Prozess zur Ideenfindung und -entwicklung. Dabei steht der Mensch im Mittelpunkt, der mit einem Problem konfrontiert ist.
Abbildung 2.1 zeigt die sechs Phasen, die im Prozess durchlaufen werden und visualisiert den iterativen Ansatz, das mehrmalige Durchlaufen in unterschiedlichen Kreisen im Laufe des Prozesses.

In vielen Veranstaltungen der CIC ist uns besonders das Modell des „Double Diamonds“ (Abbildung 2.2) begegnet, da der Ablauf der Cross in einem solchen Rahmen strukturiert wurde. Anhand der Fragestellung wird ein breiter Problemraum geöffnet, in dem möglichst viel Wissen aus verschiedenen Perspektiven zusammenfließt. Diese Problemdefinition wird nun konkretisiert, um einen Lösungsraum zu öffnen, der Lösungsansätze jeder Art zulässt. Anschließend werden auch diese im Team besprochen und am Ende entsteht ein konkreter Prototyp.

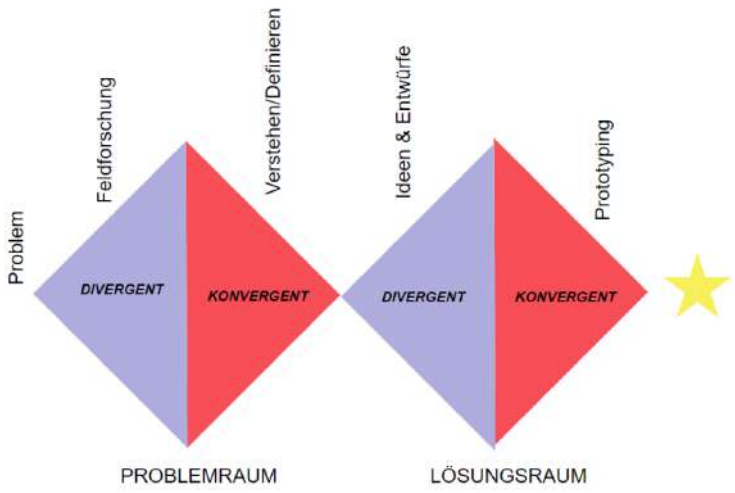


Abbildung 2.2: Denkmodell „Double Diamond“[1]

Dabei hatten wir als Team zwei Favoriten.
Die Idee, die es nicht geschafft hat, war der sogenannte „Bar-Bus“. Es sollte ein ausgebauter Bully sein, der verschiedene Konzepte vereint hätte. Darunter fällt ein als Rutsche umgesetzter Flascheneinwurf, die Möglichkeit den Bus zu beschreiben und bemalen, sowie ein Angebot Snacks in müllfreier Verpackung und Entertainment-Angebote, wie Musik. Er sollte eine Anlaufstelle für die Menschen werden, die aufgrund ihrer Kleidung, ihres Aussehens oder ihrer finanziellen Mittel oft den Zugang zu den Frankfurter Clubs verwehrt bekommen.

Problematisch an der Idee war, dass die Stadt hier als Anbieter von Waren auftritt und ebenfalls die Party-Stimmung nicht mindert, auch wenn dadurch wohlmöglich die aggressive und rücksichtslose Atmosphäre minimiert würde.
Daher haben wir im weiteren Verlauf unsere zweite Idee, einen interaktiven Mülleimer, verfolgt, den wir im nächsten Kapitel ausführlich beschreiben.

3 Projektbeschreibung

3.1 Grundlegende Konzepte

Die Idee des interaktiven Mülleimers (anfänglich als „Party-Bin“ bezeichnet) soll das Problem lösen, dass manche Feiernde den Weg zum Mülleimer nicht finden, sodass, vor allem in der Corona-Zeit, die Parks und Plätze sowie das Main-Ufer Samstag und Sonntag morgens extrem vermüllt waren.
Den ersten psychologischen Effekt den wir anwenden wollten, ist der Aspekt der Gamification. Wir wollten die Menschen dafür belohnen, dass sie ihren Müll fachgerecht im Mülleimer entsorgt haben. Dabei wollten wir erreichen dass der Mülleimer selbst auf den Mülleinwurf reagiert. Ebenfalls wichtig war uns, dass es einfach und intuitiv bleibt, keine App, kein komplexes Konzept, ein einfaches Feedback, das jeden Mülleinwurf belohnt.
Die möglichen Feedbackansätze bestanden in unseren Überlegungen aus einem auditiven Feedback, bspw. einem Geräusch, einer menschlichen Stimme oder Musik oder einem visuellem Feedback, welches per direkter oder indirekter Beleuchtung zu realisieren wäre.

Abbildung 3.1: Konzept eines Tischmülleimers, der einige der Konzepte umsetzt.

In unserem ersten Ansatz haben wir uns auf Musik fokussiert, die Möglichkeit belohnt zu werden in dem man beim Einwurf für ein nächstes Lied abstimmen kann, oder als Ansatz um die Feiernden in die Nähe der Mülleimer zu bekommen, eine Abstimmung an der jeder Mülleimer ein Lied repräsentiert und misst, wieviele Menschen um ihn herrum

stehen. Bei diesem Ansatz konnten wir ein paar große Risiken nicht ausmerzen, einerseits ebenfalls das Argument, dass diese Mülleimer die Partystimmung stärker anheizen könnten, andererseits das Thema der Rechte an der Musik. Ein Beispielentwurf ist in der Skizze 3.1 zu sehen.

Der zweite Ansatz basierte daher nun auf Licht. Ein erster Entwurf des ganzen ist die Skizze 3.2. Dieser Mülleimer reagiert beim Einwurf von Müll mit einer kleinen „Lichtshow".

In beiden Skizzen (3.1 & 3.2) zu entdecken ist ein Pfandring. Da unsere Recherchen ergeben haben, dass ein großer Teil des anfallenden Mülls auf Flaschen zurückzuführen ist und wir von Jochen Schmitz eine positive Erfahrung mit einem eingesetzten „Pfandregal" geschildert bekommen haben, wollten wir unbedingt Flaschenablagemöglichkeiten in unser Design integrieren. Eine Übersicht der In der CIC präsentierten Ideen: (**PDF der Präsentation**)

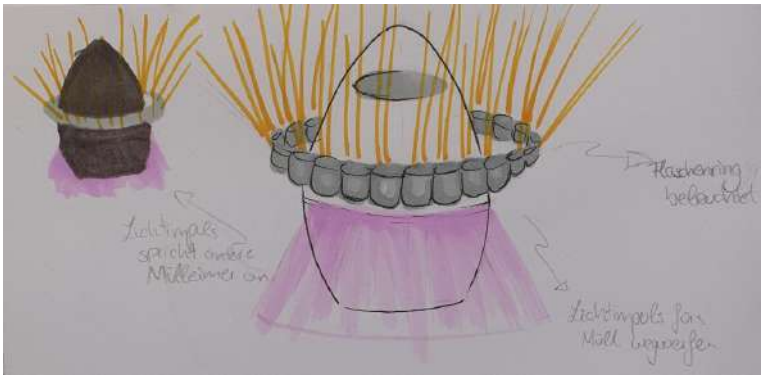


Abbildung 3.2: Konzept eines Tischmülleimers, der einige der Konzepte umsetzt.

Ausgehend aus diesen Überlegungen haben wir letztendlich ein finalen Entwurf zusammengestellt. Dieser ist minimalistischer im Design und in der Art der Beleuchtung, um sich besser ins Stadtbild einzufügen. An unserem Design optisch am auffälligsten ist der Flaschenring.

Abbildung 3.3: Das „Gerippte“.

Das Design entleiht sich dabei die Rauten am Rand des Zylinders unterhalb des Flaschenrings vom Frankfurter Gerripten, einem Apfelweinglas, das sich hoher lokaler Beliebtheit erfreut. Ein Beispielglas ist in der Abbildung 3.3 zu sehen. Neben der passenden Integration in unser Flaschenring wollen wir damit eine höhere Identifikation erzeugen. Ebenfalls konnten wir die Rautenstruktur verwenden um unsere Beleuchtung dort einzubauen, sodass sich das Modell selbst beleuchten kann und das Licht durch die Reflexion diffuser wird.

Abbildung 3.4: Skizzen des Entwurfs mit Rautenmuster

Ein weiterer Vorteil des visuellen Ansatzes ist die stärkere Präsenz im Unterbewusstsein der Feiernden. Wir Menschen reagieren Abseits unseres Fokuspunktes (der Punkt den

wir aktuell betrachten) instinktiv auf Kontraste und Bewegungen. Ein sich im Blickfeld befindender Mülleimer der dezent aufleuchtet wird unterbewusst wahrgenommen.

Abbildung 3.5: Render des Entwurfs mit Fokus auf den Flaschenring

3.2 Funkionen

Das Modell besitzt drei Grundfunktionen:

Visuelles Feedback bei Mülleinwurf

Wirft jemand Müll in den Mülleimer, so leuchten die Rauten in einer kurzen Animation auf. Wird in einem bestimmten Intervall mehrmals Müll eingeworfen, so wird die Animation intensiver. Damit haben wir eine weitere Motivation Müll zu sammeln, sowie ein dezentes aufleuchten am Tag, wenn er nur sporadisch verwendet wird.

Beleuchtung von Flaschen im Flaschenring

Als weitere Funktion sollen in den Flaschenring gestellte Flaschen von von unten beleuchtet werden. Auch hier erhält die Person ein direktes Feedback, das optisch zusätzlich sehr ansprechend aussieht. Wird ein gesamter Flaschenring verwendet, so spielt der Mülleimer ebenfalls eine Animation.

Ein mehrere Mülleimer übergreifender Lichtimpuls

In speziellen Fällen, beispielsweise bei zehn Mülleinwürfen in kurzer Zeitspanne oder bei einem gefüllten Flaschenring, soll die Animation als Impulswelle an benachbarte Mülleimer weitergeleitet werden. Dies dient als weiterer Ansporn, der einen möglichen Wett-

bewerbsehrgeiz erzeugt, falls mehrere Gruppen an Feiernden anwesend sind. Allgemein wichtig in der Realisation ist hier, dass die Animationen einen Weg zwischen nicht zu auffällig, aber wahrnehmbar finden. Wir haben diesen Grad immer als Lagerfeuerstimmung bezeichnet.

4 Aufgabenverteilung

Während der KickOff Veranstaltung haben wir Personen festgelegt die die Rollen der Projektleitung und die der Außenkommunikation übernehmen. Im Laufe des Projekts wurden diese Aufgaben aber auch von anderen Personen übernommen, sei es wegen Krankheit, anderen Abwesenheiten oder wegen der Menge an Aufgaben.

Rolle	**Person**
Projektleiterin	Celina Krug
Außenkommunikation	Florian Bucher

Tabelle 4.1: Übersicht der intern vergebenen Rollen.

Aufgabe	**Person(en)**
Löten der technischen Komponenten	Robin, Sven,Celina, Moritz
Bau der Kabel(bäume)	Robin, Sven, Moritz
Modellbau des großen Modells	May, Lucas, Annika, Sven, Celina, Moritz, Robin
Modellbau der zwei kleinen Modelle	May, Lucas, Annika, Sven, Moritz
Implementation & Kalibrierung des Füllstandssensor	Sven
Implementation & Kalibrierung des dTOF-Sensors	Sven
Implementation der Main-dTOF	Sven
Implementation der Animationen	Robin
Implementation der BLE-Kommunikation	Robin
Implementation der Main-Receiver & Main-Transmitter	Robin
Vorbereitung & Präsentieren Abschlussveranstaltung	Celina, May
Kommunikation/Absprachen mit den Praxispartnern	Robin, Florian
Teilnahme an den CheckIns	Celina

Tabelle 4.2: Retrospektiv: Welche Aufgaben wurden von wem übernommen.

5 Technische Realisation

5.1 Aufgabenbereiche

Die technischen Aspekte des Gerippten lassen sich grob in folgende Teilbereiche aufgliedern:

- Beleuchtung der Tonnenaußenwand inklusive Animation der Lichteffekte sowie Beleuchtung des Flaschenrings
- Sensorik zur Detektion von Flaschen im Flaschenring
- Detektion von Gegenständen die in den Behälter geworfen werden
- Messung des Füllstandes
- Verkabelung der Sensorik und der LEDs.
- Austausch von Informationen zwischen mehreren Endgeräten
- Verarbeitung der Sensordaten und Ansteuerung der Beleuchtung

In den nachfolgenden Kapiteln werden die zur Realisierung der verschiedenen Teilaspekte genutzten Komponenten beschrieben und diskutiert.

5.2 Mikrocontroller

5.2.1 Diskussion

Als zentrale Steuerungskomponente ist die Wahl des Mikrocontrollers von zentraler Bedeutung für die technische Realisierung des Projekts. Die wichtigste Eigenschaft des Mikrocontrollers ist in diesem Projekt die generelle Verbreitung des Mikrocontrollers, da die Einarbeitungszeit sehr kurz ist und somit eine große Auswahl von Informationsquellen entscheidende Vorteile bringt. Dadurch stechen zwei Hersteller besonders heraus. Zum einen Arduino, die eine große Auswahl unterschiedlichster Mikrocontroller mit verschiedenstem Funktionsumfang anbieten und zum anderen Espressif, deren ESP32 Mikrocontroller eher als „Alleskönner“ bezeichnet werden kann, da er im Vergleich zu den meisten Arduino Modellen wesentlich mehr und bessere Hardware bietet. Eine weitere Möglichkeit bietet die Nutzung des Einplatinencomputers Raspberry Pi, der einen weitaus größeren Funktionsumfang und wesentlich mehr Rechenleistung besitzt als ein Mikrocontroller. Die Nachteile des Raspberry Pi liegen in den hohen Anschaffungskosten

und dem ebenfalls hohen Stromverbrauch. Im allgemeinen ist ein Raspberry Pi für dieses Projekt überdimensioniert, da ein Großteil des Mehrwertes den dieser gegenüber einem Mikrocontroller bietet in diesem Projekt keine Verwendung fände.

Da eine kabellose Kommunikation der Mikrocontroller untereinander realisiert werden soll, ist es essenziell, dass der Mikrocontroller mit einem WiFi oder Bluetooth Modul ausgestattet ist. Der ESP32 bringt beides mit. Arduino bietet ebenfalls einige Modelle an, die mit mindestens einem der Module ausgestattet sind. Insbesondere die Nano 33 Reihe von Arduino bietet Geräte, die für die Verwendung in diesem Projekt die notwendigen Kriterien erfüllen. Hier sind speziell die Modelle Nano 33 BLE und Nano 33 IoT hervorzuheben. Das Modell Nano 33 IoT ist dabei speziell für die Verwendung in Internet of Things Geräten designed und bietet sowohl ein WiFi als auch ein Bluetooth Modul. Das Modell Nano 33 BLE besitzt lediglich ein Bluetooth Modul, dieses ist jedoch mit Bluetooth 5 etwas moderner als die Bluetooth 4.2 Variante, die im Nano 33 IoT Modell verbaut ist. Zudem besitzt das BLE Modell einen leistungsstärkeren Prozessor, der jedoch durch integrierte Energiesparfunktionen mindestens genauso energiesparend ist. Der Energieverbrauch stellt sich für dieses Projekt ebenfalls als wichtiger Faktor heraus, da für eine mögliche Weiterführung des Projekts eine Stromversorgung der Geräte über Solarzellen und Akkus implementierbar sein soll. Die hier diskutierten Vor- und Nachteile der 3 zur Verfügung stehenden Mikrocontrollern wird in Tabelle 5.2 nochmals in kürze zusammengefasst.

Tabelle 5.1: Vergleich der Mikrocontroller ESP32, Nano 33 IoT und Nano 33 BLE

Espressif ESP32	**Arduino Nano 33 IoT**	**Arduino Nano 33 BLE**
Leistungsstärkster Mikrocontroller	Am wenigsten Leistung im Vergleich	Leistungsstarker Prozessor
Bluetooth & WiFi	Bluetooth & WiFi	Nur Bluetooth, dafür Bluetooth 5
Vergleichsweise hoher Stromverbrauch	Geringer Stromverbrauch	Geringer Stromverbrauch

Im Kapitel 5.7 wird erläutert, dass die Kommunikation über den Bluetooth beziehungsweise Bluetooth Low Energy Standard erfolgen soll. Somit bieten die WiFi Module, die der ESP32 und der Nano 33 IoT Mikrocontroller mitbringen keine weiteren Vorteile gegenüber dem Nano 33 BLE Mikrocontroller. Da dieser außerdem einen sehr geringen Stromverbrauch im Verhältnis zu seiner Leistungsfähigkeit aufweist, wurde dieser für die Verwendung in diesem Projekt ausgewählt.

5.2.2 Platform & Code-Struktur

Als Programmiersprache kommt das systemnahe C++ zum Einsatz, die klassische Wahl zur Programmierung von Arduinos. Allerdings haben wir uns schon zu Beginn von der ArduinoIDE getrennt, da uns dort ein paar Quality of Life Features nicht zu Verfügung standen. Als Alternative haben wir uns für das PlatformIO Plugin für Visual Studio Code entschieden. Unter den benötigten Features fällt die Möglichkeit Environments zu definieren und damit die Möglichkeit mehrere Main-Files für verschiedene Arduinos in der selben Codebasis zu halten. Je nach ausgewähltem Environment wird dann ein spezifizierte Main verwendet indem alle anderen exkludiert werden. Auch konnten wir Makros an den Compiler übergeben um Arduinos IDs zuzuweisen, die verwendet werden um unterschiedliche/einzigartige UUIDs zu verwenden. Im Codeausschnitt 1 ist beispielhaft das Environment „transmitter“ zu finden, welches das Makro ARDUINO_ID=3 definiert und die Dateien main_receiver.cpp, main_dtof.cpp und main_rest.cpp beim kompilieren rausfiltert. Zusätzlich bietet PlatformIO eine Definition von Abhängigkeiten auf Projektebene, so hatten wir immer die selben Versionen der Bibliotheken und zusätzlich werden diese Abhängigkeiten sowie die benötigte Software zum Arbeiten mit Nano 33 BLE Arduinos automatisch heruntergeladen.
Die Strukturierung des Codes ist aufgeteilt in jeweils hpp- und cpp-Datei für jede Komponente unseres Modell. So führen wir eine weitere Abstrahierungsebene ein und können unsere Main-Dateien übersichtlicher und vorallem kompakter gestalten, in dem die dort verwendeten Funktionen auf unsere Aufgaben angepasst sind. Dabei ist die grundlegende Struktur einer Klasse, oft aufgeteilt in einen Konstruktor zum Setzen der benötigten Instanzvariablen, einer init()-Methode die im Setup aufgerufen wird, mehr oder wenig abstrahierte Methoden, wie beispielsweise BLEPeripheralWrapper::publishEvent(u_int8_t val) die den Wert einer BLE-Characteristic aktualisiert und nach einem gewissen Intervall wieder zurücksetzt oder u_int8_t LevelMeter::getFillLevel(), die den aktuellen Füllstand zurückgibt, sowie in manchen Fällen update()-Methoden. In diesen wurden alle Aufrufe vereint die periodisch erfolgen müssen.

5.3 Beleuchtung

5.3.1 Diskussion

Um die Flaschen des Flaschenrings und die Rauten zu beleuchten benötigen wir LEDs. Es eröffnet sich herbei ein großer Raum von Optionen. LEDs sind zu unterscheiden in Größe und Form, in Helligkeit und der Art der Ansteuerung und zwischen RGB- und RGBW-LEDS. Eine der Kategorien konnten wir mit dem Rat von Prof. Hoffmann und Simon Schröder zu Beginn klären. Um die Menge der benötigten LEDs individuell über ein Arduino ansteuern zu können, benötigen wir LEDs mit einem integriertem Driver-Chip. NeoPixel von Adafruit, oder die vielen äquivalente Optionen auf dem Markt ermöglichen diese Funktionalität in dem sie als 24 oder 32 Bit Shift-Register agieren. Dabei wird die Farbintensität jeder der drei oder vier Farben (RGB oder RGBW) als ein 8-Bit Wert angegeben und von LED zu LED weitergeleitet. Wird ein RESET-Signal gesendet, wird

```
[env]
build_flags = -Wall
lib_ldf_mode = deep+
platform = nordicnrf52
board = nano33ble
framework = arduino
monitor_speed = 115200
build_src_filter = +<*> -<.git/>
lib_deps =
  adafruit/Adafruit NeoPixel@^1.10.4
  sparkfun/SparkFun Qwiic TMF882X Library@^1.0.0
  arduino-libraries/ArduinoBLE@^1.2.2
  thijse/ArduinoLog@^1.1.1

[env:transmitter]
build_flags =
    ${env.build_flags}
    -DARDUINO_ID=3
build_src_filter =
    ${env.build_src_filter}
    -<main_receiver.cpp>
    -<main_dtof.cpp>
    -<main_test.cpp>
upload_port = /dev/ttyACM0
monitor_port = /dev/ttyACM0

...
```

Listing 1: Auszug aus der platformio.ini Datei

der zuletzt gespeicherte Wert angezeigt.
Weitere Überlegungen mussten wir uns besonders hinsichtlich der Größe und Form machen. Für die Animationen der Tonne wollten wir jede Raute an den oberen Kanten, dessen Längen jeweils ca. 10cm betragen, mit LEDs bestücken. Hierfür haben sich LED-Strips angeboten, wobei wir möglichst schmale benötigen, damit sie, wenn eine Person den Mülleimer von oben anguckt, in der Rautenstruktur verschwinden. Als alternativen Ansatz hatten wir vorher überlegt die Rauten mit einzelnen LEDs zu beleuchten, doch durch den enormen Aufwand der Verkabelung haben wir diesen Ansatz im Laufe der Entwurfsphase wieder verworfen.
Für die LEDs des Flaschenrings wiederum haben wir vorallem eine weitere Anforderung: Die Helligkeit. Dadurch dass das Licht durch die Flaschen gefiltert wird, müssen wir sichergehen, dass die Lichtintensität hoch genug ist. Bei der Recherche haben wir zwei Optionen gefunden. Einerseits ebenfalls die NeoPixel, hier als einzelne LEDS im RGBW Format um eine höhere Helligkeit zu erzielen, andererseits der „große Bruder“, die Adafruit NeoPixel Pixies. Diese LEDs laufen statt mit 0.2 Watt mit 3 Watt, ein extremer Unterschied, aber dadurch, dass wir 15 Flaschenringfächer haben, mussten wir uns aus Budgetgründen für jeweils zwei LEDs der kleineren Variante entschieden, die mit den Breakoutboards bei 75ct/Stück liegen während sich die Pixies bei 14.95€/Stück außerhalb des Rahmens befinden. Ein weiterer wichtiger Faktor den man hätte beleuchten müssen ist die Hitzeentwicklung der großen LEDs, doch so mussten wir uns nicht mit den Details beschäftigen.

5.3.2 Implementation

Hardware

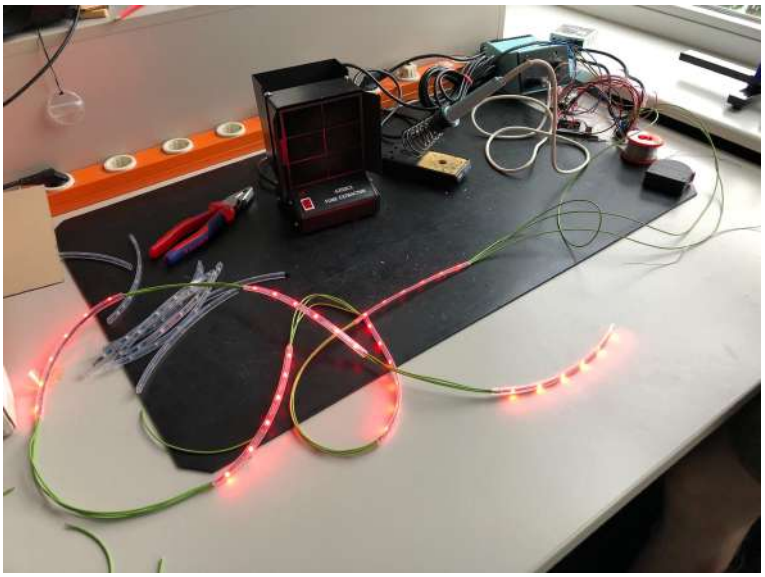


Abbildung 5.1: Eins der drei Segmente des Rauten LED-Strips.

Wir haben uns bei der Rautenbeleuchtung für 7 Meter Adafruit Mini Skinny NeoPixel Strips entschieden die mit 30 RGB LEDs pro Meter bestückt sind. So konnten wir den Strip in jeweils 20cm Segmenten unterteilen und jede Raute mit insgesamt 6 LEDs beleuchten. Der Strip ist mit 1 Ampere Verbrauch pro Meter angegeben, was die Wahl des 10 Ampere Netzteils begründet. Die Strip-Segmenten haben wir dann wieder verbunden, in dem wir die drei Kontakte per Kabel an das nächste Segmenten weitergeleitet haben. (Siehe Abbildung 5.1) Die Abstände zwischen den Strips und damit die Kabellängen sind abhängig von der Position am Modell. Diese Aufteilung wird in Abbildung 5.2 skizziert.

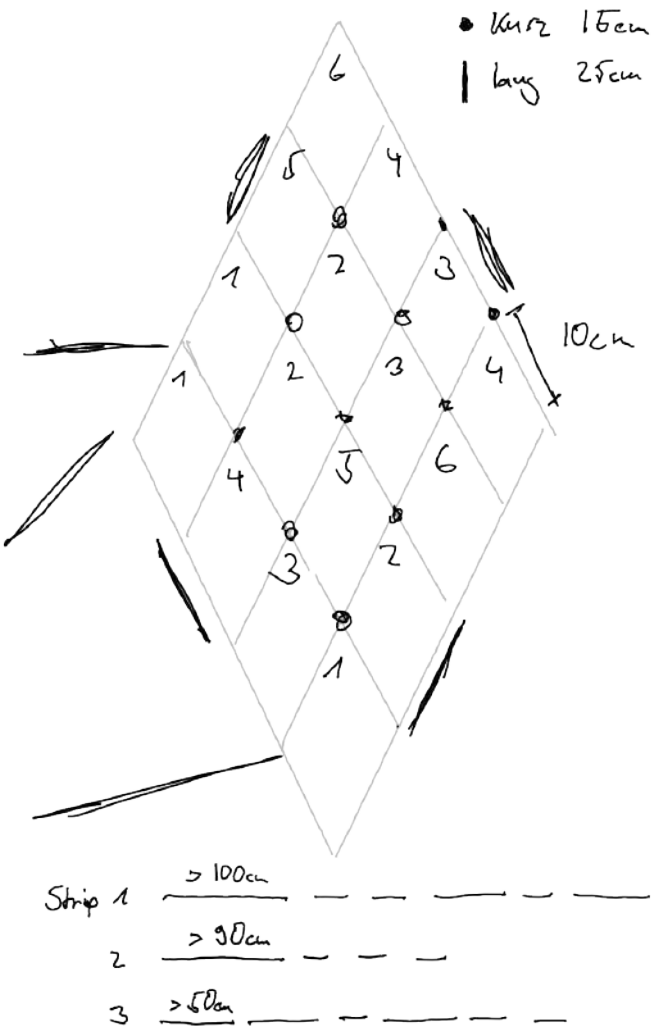
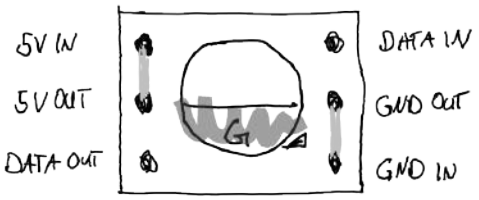
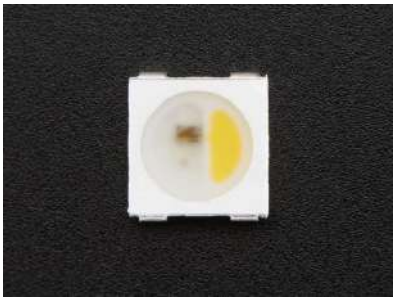


Abbildung 5.2: Anordnung der LED-Strips in der Raute am Modell

Dort zu erkennen ist, dass wir nur ein Teil der Rauten beleuchtet haben, da es uns an LED-Strips fehlte. Eine aus Rauten bestehende große Raute, die wir aus drei Kabel-Strip-Strängen zusammensetzen. So laufen am Ende drei Stecker zum Arduino.



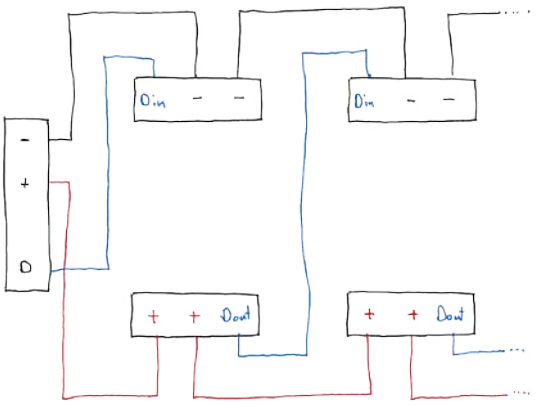
(a) Pin-Belegung auf den verwendeten 6-Pin BreakOut-Boards. Gezogene Lötbrücken sind in grau eingezeichnet.



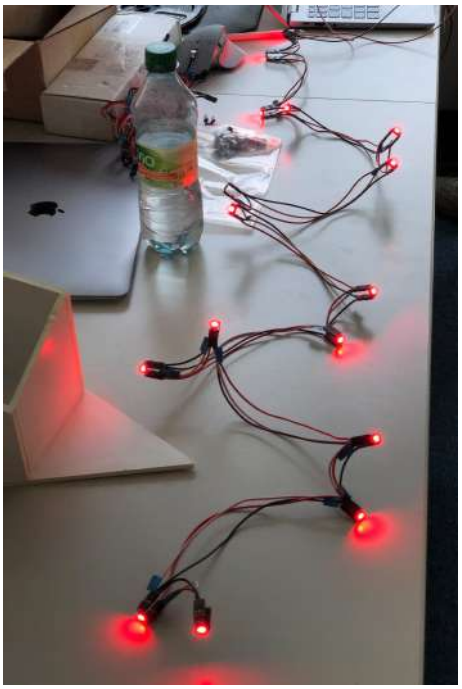
(b) Ein Pixel.

Abbildung 5.3: NeoPixel RGBW LED SK6812RGBW

Für die Flaschenring LEDs haben wir Breakoutboards beschafft, auf die wir die LEDs gelötet haben. Diese Boards haben 6 Pins, während die LEDs 4 Pins haben. In der Skizze 5.3a haben wir gezeigt wie wir diese zwei Extra-Pins dafür verwenden, über Lötbrücken Strom und Ground im Kabelbaum an die nächsten LEDs weiterzuleiten. Diese zwei Kabelbäume hat vorallem Moritz aus der HCU gebaut, wohlmöglich eine der aufwendigsten Unterfange unseres Projekts. Den Aufbau eine solchen Kabelbaums ist in Abbildung 5.4a gezeigt, der fertige Kabelbaum in Abbildung 5.4b.



(a) Skizze des Aufbaus.



(b) Fertiggestellter Kabelbaum in Betrieb.

Abbildung 5.4: Kabelbäume der Flaschenring LEDs

```
1 static uint32_t Color(uint8_t r, uint8_t g, uint8_t b, uint8_t w) {
2     return ((uint32_t)w << 24) | ((uint32_t)r << 16) | ((uint32_t)g <<
  ↪  8) | b;
3 }
```

Listing 2: Konkatenation der 4 8-Bit Werte der Farben RGBW. - Adafruit_NeoPixel

Software

Softwareseitig bieten die Neopixel eine sehr gute Integration im Arduino Universum mit der Bibliothek „Adafruit_NeoPixel“. Jeder LED-Strang benötigt nur ein Digital-Pin als Datenleitung. In der Initialisierung werden Pin-Nummer, Anzahl der hintereinander geschalteten LEDs und der Anordnung der Farbwerte in den Chips benötigt. Für die LEDs des Flaschenrings ist das GRBW für die LED-Strips GRB. Ab diesem Zeitpunkt stehen zwei essentielle Methoden zur Verfügung. Mit der Klassenmethode

```
1 void Adafruit_NeoPixel::setPixelColor(uint16_t n, uint32_t c)
```

kann einer LED über einen Index eine Farbe zugewiesen werden. Diese Farbe ist eine Konkatenation der 4 8-Bit Werten (Siehe Codeausschnitt 2). Wird kein Weiß verwendet, wird dieser Wert beim Weiterleiten an die LEDs verworfen.
Die zweite Klassenmethode

```
1 void Adafruit_NeoPixel::show(void)
```

lässt alle LEDs ihren letzten gesetzten Farbwert anzeigen.

Während die Realisation der Animation der Rauten im nächsten Kapitel behandelt wird, bedienen sich die Flaschenring-LEDs direkt den Funktionen oben erwähnten Funktionen, sodass die LEDs in jedem Schleifendurchlauf des Arduinos ihren Farbwert zugewiesen bekommen und dieser angezeigt wird. Das Neusetzen der Farbwerte muss regelmäßig in einem Interval geringer einer Sekunde erfolgen, da sonst unregelmäßige Abweichungen auftreten.

5.4 Animation

5.4.1 Diskussion

Grundlegend sind Animationen Farbwerte über Zeit. Da wir jede LED einzeln ansprechen können, haben wir die größtmögliche Freiheiten in der Gestaltung. Es stellten sich nun mehrere Fragen. In welchem Format benötigen wir die Farbwerte? Wollen wir einen Wert

für jede LED speichern, oder führen wir eine Abstraktionsebene ein. Wie lange sollen die Animationen spielen und in welcher Framerate?
Zwei Fragen ließen sich durchs Ausprobieren klären. Animationen konnten wir mit einer Dauer von 2 Sekunden so darstellen, dass sie möglichst schnell, aber in einer organischen Bewegung umsetzbar waren und als Framerate hatten wir ab 10 Frames pro Sekunde ein flüssiges Ergebnis.
Des weiteren haben wir uns als Team auf eindimensionale Animationen verständigt die nur in der Vertikalen variieren, sodass wir, wie in Abbildung 5.2 gezeigt, 21 (7 * 3) unterschiedliche Höhenwerte pro Frame benötigen.
Zusätzlich mussten noch zwei technische Aspekte bedacht werden. Animationen sollten zeitbasiert sein und nicht abhängig davon, dass das Aufrufen der Methoden in einem gleichmäßigem Rhythmus geschehen ablaufen.

5.4.2 Implementation

Um möglichst wenige Berechnung während der Laufzeit ausführen zu müssen, haben wir die Farbwerte von vornherein berechnet und in die konkatenierte 32-Bit Form transformiert. So benötigen wir für eine Sekunde 10 Frames mit jeweils 21 32 Bit Werten. Ziehen wir in Betracht, dass der Arduino Nano 33 BLE 256KB RAM besitzt wären das 38 Sekunden Animationen die wir im Speicher halten könnten, ausgenommen die RAM-Nutzung der anderen Komponenten. Auch wenn wir damit für unseren geplanten Umfang an Animationen auskommen würden, haben wir uns dazu entschieden um weitere Erweiterungen umsetzen zu können, die Animationsdaten in den Flash-Speicher des Chips zu schreiben. Dieser beträgt beim Arduino Nano 33 BLE 1MB. Konstante Daten mit dem Schlüsselwort PROGMEM in der Deklaration werden beim Flash-Prozess in den Flash-Speicher geschrieben. Um Daten aus dem Flash-Speicher zu lesen gibt es eine Reihe von Methoden die als Parameter den Pointer nehmen und unterschiedliche Byte-Größen lesen. Beispiele sind pgm_read_float() und pgm_read_word(). Alle diese Funktionalitäten werden von der Bibliothek „<avr/pgmspace.h>“ bereitgestellt.
Bei Animationen mit linearen Farbverläufen, wie wir sie verwenden, muss man zusätzlich beachten, dass ein linearer Verlauf zwischen 0 und 255 vom Menschen nicht als linear wahrgenommen wird. Hierfür gibt es die sogenannte Gamma-Korrektion, eine nichtlineare Funktion die als Array mit 256 Werten implementiert ist. Diese bildet jeden Wert auf einen korrigierten ab, sodass das Endprodukt wieder linear wahrgenommen wird. Diese Korrektion wenden wir aber ebenfalls schon beim Erstellen der Animationen an.

```
1  [...]
2  static PROGMEM uint32_t ANIMATIONS[AMOUNT_ANIMATIONS][AMOUNT_LED_LEVELS
   ↪  * FRAME_AMOUNT] =
3  {
4      {
5      512, 512, 512, 512, [...]
6      1792, 1792, 1792, 1792, [...]
7      18688, 29952, 44544, 62720, [...]
8      [...]
9      }
10 }
11 [...]
```

Listing 3: Angepasster Auszug aus der animation.hpp

Skript generate_animation.py

Das Generieren der Animationen haben wir in einem Python-Skript modelliert, welches eine Animation mit einer beliebigen Farbe in dem für das Arduino-Programm benötigte Format ausgibt. Es lässt ein Lichtimpuls von unten nach oben wandern und passt die anliegenden Werte sowie den Start und das Ende so an, dass es keine ruckartigen Bewegungen gibt.
Es ist dabei modular aufgebaut, sodass weitere Animationen hinzugefügt werden können.

Ein Ausschnitt der fertigen Animation ist im Codebeispiel 3 zu sehen.

Während wir bei den Farbwerten der Animationen mit 21 Werten arbeiten, müssen wir die 96 LEDs ebenfalls den 21 Höhen zuweisen. Das haben wir mit einer einfachen Heightmap realisiert, ein Array das für jeden Index und damit jede LED ein Höhenwert mitbekommt. So kann später eine effiziente Berechnung des benötigten Farbwerts erfolgen.

Beim Start einer Animation wird nun ein Zeitstempel mithilfe der Methode millis() gespeichert. Jeder Aufruf der im Codeausschnitt 4 skizzierten update()-Methode berechnet sich anhand dieses Wertes das Zeit-Delta seit Beginn der Animation, um den aktuellen Frame zu berechnen. Das Array mit den Farbwerten im Flashspeicher besitzt für jede Animation 21 * „Anzahl Frames“ Farbwerte, wobei die ersten 21 Werte dem Frame 0, die nächsten 21 Werte dem Frame 1 zugewiesen sind etc. So berechnet sich die aktuelle Position der Animation wie in Zeile 6 zu sehen mit der Addition mit 21 * „Aktueller Frame“. Ausgehend von diesem Pointer wird nun über jede LED iteriert, wobei die Position aus der Height-Map gelesen wird und diese auf den Pointer addiert wird, um den korrekten

```
// Einschub aus neo_animation.hpp:
const u_int8_t *height_map;

[...]
u_int8_t idx = (millis() - animation_data.ts_start) / 100;
u_int32_t* ptr = animation_data.animation_ptr + (AMOUNT_LED_LAYERS *
↪  idx);
for (uint16_t i = 0; i < num_pixels; i++)
{
    pxl.setPixelColor(i, pgm_read_dword(ptr + height_map[i]));
}
pxl.show();
[...]
```

Listing 4: Farbwerte zeitbasiert aus dem Flashspeicher lesen - Auszug aus der update()-Methode der neo_animation.cpp

Farbwert aus dem Flashspeicher zu lesen. Bevor diese Berechnung stattfindet wird, wie in Codebeispiel 5, überprüft, ob die aktuelle Animation vollendet ist.

5.5 Einwurferkennung

5.5.1 Diskussion

Ein wichtiger Punkt bei der Realisierung der Einwurfserkennung ist, dass nur Gegenstände erkannt werden sollen, die gänzlich in den Behälter geworfen wurden und nicht wieder herausgezogen werden, wie es beispielsweise beim Hereinhalten einer Hand in den Behälter der Fall ist. Um dies zu implementieren muss der Gegenstand auf mindestens

```
[...]
if (millis() > (animation_data.ts_start + animation_data.duration))
{
    animation_data.state = OFF;
    pxl.clear();
    pxl.show();
}
[...]
```

Listing 5: Animation zeitbasiert beenden. - Auszug aus der update()-Methode der neo_animation.cpp

zwei übereinanderliegenden Ebenen detektiert werden. Wird auf der oberen Ebene der Gegenstand erkannt, wird ein möglicher Einwurf detektiert. Jedoch erst wenn die obere ebene keinen Gegenstand mehr detektiert, während die untere Ebene diesen weiterhin detektiert ist der Einwurf des Gegenstandes abgeschlossen. Sollte der Gegenstand hingegen zuletzt lediglich auf der oberen Ebene detektiert werden, so wurde er wieder aus dem Behälter herausgezogen und somit wird dies nicht als Einwurf gewertet.

Die Realisierung der zwei Detektionsebenen durch zwei einzelne Sensoren oder Lichtschranken hat jedoch das Problem zur Folge, dass die beiden Sensoren miteinander interferieren, wenn sie die gleiche Lichtwellenfrequenz nutzen, was zur Folge hat, dass eine genaue Abgrenzung zwischen den Ebenen nicht möglich ist. Zur Realisierung dieser Funktion ist es deshalb notwendig Sensoren zu verwenden, die unterschiedliche Lichtwellenfrequenzen verwenden, oder es muss ein Sensor verwendet werden, der die Möglichkeit bietet Objekte in einem dreidimensionalen Bereich zu detektieren, statt nur auf einer bestimmten Ebene zu messen.

Bei der Recherche nach zur Verfügung stehenden Komponenten fiel die Wahl letztlich auf den Mini dToF Imager TMF8821 von SparkFun. Dabei handelt es sich um einen so genannten direct Time of Flight Sensor. Damit ist ein Sensor gemeint, welcher viele kurze Lichtimpulse aussendet und die reflektierten Lichtimpulse wieder detektiert. Um die Entfernung zum Objekt zu bestimmen, welches den Lichtimpuls reflektiert hat, wird die Zeit gemessen, die zwischen der Aussendung des Lichtimpulses und der Detektion des reflektierten Lichtimpulses verstreicht. Eine Besonderheit des TMF8821 ist, dass dieser einen in mehrere Felder aufgeteilten Messbereich besitzt. So kann die Entfernung zu einem Gegenstand beispielsweise in 9 verschiedenen Messbereichen, aufgeteilt auf ein 3x3 Quadrat, gemessen werden. Der Messbereich kann softwareseitig angepasst werden und somit sind verschiedene Messwinkel und Submessbereiche in den Ausprägungen 3x3, 4x4 und 3x6 möglich. Das bedeutet, dass der Messbereich des Sensors bereits in mehrere Ebenen aufgeteilt ist, die getrennt voneinander betrachtet werden können. Technisch realisiert wird dies durch Single Photon Avalanche Photodioden (SPAD), welche hinter einer speziellen Linse angebracht sind, die den Messbereich auf die Fotodioden fokussiert. Die einzelnen Photodioden werden dann den verschiedenen Messzellen zugeordnet und somit kann ein Photon, dass von einer bestimmten SPAD empfangen wird einem Bereich im Feld der Messung zugeordnet werden. Der im folgenden verwendete Begriff 'SPAD Map' bezeichnet die Konfiguration des Messfeldes durch Aufteilung in Unterbereiche mit jeweils fest zugeordneten SPADs.

5.5.2 Implementation

Um den TMF8821 softwareseitig zu implementieren wird die von SparkFun zur Verfügung gestellte Bibliothek verwendet. Diese beinhaltet alle notwendigen Methoden um den Sensor zu konfigurieren, Messungen durchzuführen und die Messergebnisse auszuwerten.

Die Voreingestellte SPAD Map des Sensors besitzt eine Größe von 3x3 Messbereichen und die Messwinkel betragen in der Horizontalen 33° und in der Vertikalen 32°. Die ersten Messungen, mit dem in der Bibliothek zur Verfügung gestellten Codebeispiel Example-01_Basic, welches alle Messewerte auf dem Seriellen Monitor ausgibt, lieferten sehr zufriedenstellende Ergebnisse. Dabei wurde eine Hand in unterschiedlich abgemessenen Abständen und an unterschiedlichen Positionen vor den Sensor gehalten und die Werte auf dem Seriellen Monitor mit Position und Entfernung der Hand abgeglichen.

Für die geplante Einwurföffnung des Gerippten ist ein Messwinkel von maximal 33° jedoch nicht ausreichend, weshalb die beschriebenen Probemessungen ebenfalls mit anderen SPAD Map Konfigurationen durchgeführt wurden. Dabei war zu beobachten, dass die Konfigurationen, mit einem 4x4 oder 3x6 Messbereich besonders im Randbereich viele unplausible Ergebnisse lieferten. Da die Fehlerursache für dieses Verhalten nicht in kurzer Zeit ausgemacht werden konnte, wurde sich aufgrund der wenigen zur Verfügung stehenden Zeit auf die Verwendung einer SPAD Map mit einem 3x3 Messbereich konzentriert. Die vorkonfigurierte SPAD Map mit der ID 6 bietet mit 52° in der Vertikalen den größten Messwinkel aller vorkonfigurierten SPAD Maps, weshalb diese letztendlich im Prototypen Verwendung fand. Das Schema dieser SPAD MAP wird in Abbildung 5.5 dargestellt.

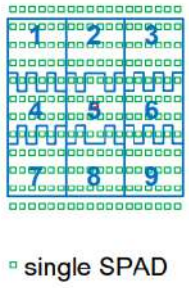


Abbildung 5.5: Schema der verwendeten SPAD Map[1]

Da bei dieser SPAD Map Konfiguration der vertikale Messwinkel größer ist als der horizontale Messwinkel muss der Sensor für einen maximal großen Messbereich um 90° versetzt eingebaut werden, sodass die Messfelder 1, 4 und 7 die obere Messreihe bilden und die Messfelder 3, 6 und 9 die untere. Da jedoch auch der horizontale Messwinkel mit 41° einen großen Messbereich abdeckt, der nach der Rotation in der Vertikalen aufgespannt ist, wird die daraufhin obere Messebene mit den Feldern 1, 4 und 7 bei der Messung nicht ausgewertet, da dieser Messbereich zu großen Teilen oberhalb der Einwurföffnung gelegen ist. Die zwei geforderten Messebenen sind somit durch die Messbereiche 2, 5 und 8 für die obere Messebene sowie 3, 6 und 9 für die untere gegeben.

Die Implementation der abstrahierten Interaktion mit dem Sensor ist in der Datei tmf8821.cpp realisiert, die entsprechende Schnittstelle dazu durch die Header Datei tmf8821.hpp ge-

geben. Durch die dort definierte Methode init wird der Sensor konfiguriert. Das bedeutet, die SPAD Map wird auf die vordefinierte SPAD Map mit der ID 6 festgelegt und der zeitliche Abstand zwischen zwei Messungen wird auf 50 ms gesetzt. Dieser Wert hat sich aus Testläufen mit unterschiedlichen Werten ergeben. Das Ziel ist diesen Wert so gering zu halten wie möglich, damit ein eingeworfener Gegenstand auf jeden Fall detektiert wird und nicht genau zwischen zwei Messungen eingeworfen werden kann. Die durchgeführten Tests haben ergeben, dass Messungen mit einem zeitlichen Abstand von unter 50 ms jedoch ungenauere Messwerte hervorbrachten. Die init Methode sollte typischerweise in der Setup Methode der main Klasse aufgerufen werden.

Die Methode start lässt den Sensor eine einzelne Messung durchführen und interpretiert zugleich das Ergebnis. Dazu werden die vom Sensor zurückgelieferten Messwert zu einem Status ausgewertet, welcher in der aktuellen Instanz des Sensorobjekts gespeichert wird. Dabei kann das Messergebnis einem von drei möglichen Status zugeordnet werden. Die Status und ihre Bedeutungen werden in folgender Tabelle erläutert.

NONE	In keiner der beiden auszuwertenden Messreihen wird ein Objekt detektiert.
INCOMING	In der oberen Messebene wurde ein Objekt detektiert.
DETECTED	Der vorherige Zustand ist INCOMING und in der unteren Messebene wird ein Objekt detektiert, während in der oberen Messebene keines mehr detektiert wird. Dieser Zustand kann nicht mehr durch weitere Messungen geändert werden, sondern wird beim Auslesen des Sensorzustandes zurückgesetzt.

Um den Zustand des Sensors auszulesen, kann die Methode getState aufgerufen werden. Diese gibt immer den aktuell gespeicherten Zustand zurück. Ein Parameter vom Typ bool gibt an, ob der Zustand auf NONE zurückgesetzt werden soll, wenn der aktuelle Zustand DETECTED ist. Diese Option dient lediglich Debug Zwecken und im Produktivsystem sollte die Methode grundsätzlich nur mit dem bool-Wert true aufgerufen werden, da ein Aufruf der getState Methode mit einem Parameterwert von true die einzige Möglichkeit ist, den DETECTED Zustand zurückzusetzen.

Während des Verlaufs der technischen Umsetzung wurde deutlich, dass die Bedienung des Einwurfsensors durch den Arduino nur dann in entsprechend schneller Taktung erfolgen kann, wenn der Arduino keine weiteren Aufgaben erfüllen muss. Somit musste für die Steuerung und Auswertung des Einwurfsensors ein eigenständiger Arduino, im folgenden Sensor-Arduino genannt, eingesetzt werden, welcher über zwei digitale Leitungen mit einem zweiten Arduino, im folgenden Haupt-Arduino genannt, kommuniziert, der die restlichen Funktionen des Gerippten steuert. Eine der Leitungen wird vom Sensor-Arduino auf HIGH gesetzt, sobald der Status DETECTED ausgelesen wird. Wenn der Haupt-Arduino diesen Status erkannt hat, setzt dieser die zweite Leitung für eine Iteration seiner main loop auf HIGH, um die Meldung zu bestätigen. Diese Bestätigung wird

wiederum vom Sensor-Arduino gelesen, woraufhin dieser die erste Leitung wieder auf LOW setzt. Somit dient der Sensor-Arduino dem Einwurfsensor als Steuergerät, welches dem Haupt-Arduino lediglich mitteilt, wann ein Einwurf detektiert wurde.

Während der letzten Praxistests, im Endzustand der Modellbauphase wurde deutlich, dass die Messgeschwindigkeit für die Einwurfdetektion noch nicht ausreichend ist, um Einwürfe zuverlässig zu detektieren. Ein großer Teil der Einwürfe wurde nicht erkannt. Jedoch wurden 'falsche Einwürfe', bei denen beispielsweise die Hand hineingesteckt und wieder herausgezogen wurde, zuverlässig als nicht vollständige Einwürfe erkannt. Da es für die Vorstellung des Prototypen von Vorteil war eine zuverlässige Einwurfdetektion zu präsentieren, statt des zuverlässigen Ausschließens von nicht vollständigen Einwürfen wurde, die Bedingung für das Setzen des Detektionssignals angepasst, sodass jedes Mal ein Detektionssignal gesetzt wird, sobald der Sensorstatus nicht mehr NONE entspricht, was der Funktionsweise einer einzelnen Lichtschranke entspricht. Durch weitere Analyse des Quellcodes im Anschluss an die Präsentationsveranstaltung fiel auf, dass in der Methode start, zu sehen in Listing 6, alle Messwerte mindestens doppelt und maximal vierfach durchlaufen werden, wenn der Sensorstatus vor der Messung den Wert INCOMING aufweist, da jeder Aufruf von checkMiddleRow oder checkBottomRow jeweils einmal über alle Messwerte der letzten Messung iteriert.

Die Auswertung der Messwerte zum neuen Status kann jedoch auch mit einer einzelnen Iteration über die Messdaten realisiert werden, indem die Methoden checkMiddleRow und checkBottomRow zu einer Methode zusammengefasst werden, die einmal durch die Messwerte iteriert und dabei zurückgibt in welchen Messebenen etwas detektiert oder eben nichts detektiert wurde. Diese Verbesserung bietet somit eine Reduktion der Laufzeit, welche möglicherweise ausreichend ist, um die Einwurfdetektion in ihrem geplanten Zustand funktionsfähig zu implementieren.

5.6 Detektion von Flaschen im Flaschenring

5.6.1 Diskussion

Bei der Detektion von Flaschen im Flaschenring ist zu beachten, dass hierfür ein Sensor verwendet werden muss, der Glas in verschiedenen Farben einwandfrei detektiert. Es zeigt sich, dass sowohl die meisten optischen Sensoren, als auch Ultraschallsensoren dazu in der Lage sind. Bei optischen Sensoren ist jedoch zu bedenken, dass die maximale Detektionsreichweite bei der Detektion von Glas kleiner sein kann, als vom Hersteller angegeben, da die Reflektivität von Glas, je nach dessen Einfärbung, nicht sehr hoch ist. Somit ist ein Ultraschallsensor für die Detektion von Flaschen die zuverlässigere Wahl.

Ein weiteres Kriterium, welches vom Sensor erfüllt werden sollte, ist eine geringe Komponentengröße, da er sonst nicht ohne großen Aufwand zwischen Flaschenring und Tonnenkörper montiert werden kann. Das Sensormaß entlang seiner Messachse ist hierbei

```
void TMF8821::start(void)
{
    sensor.startMeasuring(results);
    IntakeState curr = state;
    switch (curr)
    {
        case NONE:
            if (checkMiddleRow())
            {
                state = INCOMING;
            }
            break;
        case INCOMING:
            if (checkBottomRow() && !checkMiddleRow())
            {
                state = DETECTED;
            }
            else if (!checkBottomRow() && !checkMiddleRow())
            {
                state = NONE;
            }
            break;
        default:
            break;
    }
}
```

Listing 6: start Methode aus tmf8821.cpp ohne Kommentare und Logging Ausgaben

besonders ausschlaggebend.

Die kleinsten Ultraschallsensoren die während der Komponentenrecherche ermittelt wurden besitzen Bauhöhen von einem bis zu zwei Zentimetern, während der flachste Lidarsensor der ermittelt wurde ein Bauhöhe von 3 mm besitzt. Dabei handelt es sich um Digital Distance Sensor von Pololu. Welcher, aufgrund ebendieser geringen Bauhöhe und trotz der Einschränkung, dass die Reichweite eventuell geringer ausfällt, als vom Hersteller angegeben, für dieses Projekt ausgewählt wurde. Dieser Sensor wird in verschiedenen Ausführungen angeboten, wobei diese sich in ihrer maximalen und minimalen Messdistanz unterscheiden. Da es sich um einen Sensor mit digitalem Ausgangssignal handelt und somit nur detektiert wird, ob ein Objekt vorhanden ist und nicht wie weit es entfernt ist, muss sichergestellt sein, dass die Reichweite gering genug ist um nicht die, dem Sensor gegenüberliegende, Wandung des Flaschenfachs zu detektieren, da der Sensor seitlich am Flaschenfach angebracht wird. Da die Flaschenfächer zum Zeitpunkt der Materialauswahl nicht vollständig konzipiert waren, wurde die Sensorvariante ,mit einer Reichweite zwischen 0,5 cm und 5 cm ausgewählt und damit der Durchmesser der Flaschenfächer auf mindestens 5 cm festgelegt. Die Polou Teilenummer dieser Sensorvariante ist 4050.

Wie bereits erwähnt, wird der Sensor seitlich an den Fächern des Flaschenrings montiert. Eine Montage an der Unterseite des Fachs wurde ausgeschlossen, da dies sehr fehleranfällig ist. Beispielsweise könnte sich Dreck am Boden des Fachs sammeln und den Sensor somit blockieren. Da der Sensor zwischen Flaschenring und Tonnenkörper montiert wird und der Flaschenring eine leichte Neigung nach außen aufweist, ist der Sensor von der Horizontalen Ebene nach unten abweichend ausgerichtet und somit gegen grobe Verschmutzungen, die von oben in das Fach gelangen geschützt.

5.6.2 Implementation

Die Datei pololu_digi5.cpp implementiert die Headerdatei pololu_digi5.hpp und ermöglicht die Abstratktion der Sensorfunktion in der Software. Eine Instanz dieses Sensors wird duch den Konstruktor initialisiert, welchem der entsprechende Arduino Anschlusspin übergeben wird. Da der Sensor ein hohes Potential an seinen Ausgangspin anlegt, solange nichts detektiert wird, wird der pinmode INPUT_PULLUP verwendet, sodass das Signal am Eingangspin des Arduino ebenfalls hohes Potential besitzt, solange die angeschlossene Leitung das Signal nicht aktiv auf das niedrige Potenzial bringt.

Die Methode read liest das Signal am entsprechenden Pin aus, negiert dieses und gibt das Ergebnis der Negation als bool Wert zurück. Die Negation wird durchgeführt, sodass ein Rückgabewert von true einer Detektion entspricht und ein Rückgabewert von false bedeutet, dass sich kein Objekt im Messbereich des Sensors befindet.

5.7 Kommunikation zwischen Endgeräten

5.7.1 Diskussion

Um die Kommunikation zwischen mehreren Mikrocontrollern zu realisieren müssen wir vorallem die Faktoren der Reichweite, Kompatibilität, des Energieaufwands und der Art Kommunikation berücksichtigen. Die Datenrate ist in unserem Anwendungsfall zu vernachlässigen, da wir nur die Animationsevents übertragen müssen.

Tabelle 5.2: Vergleich verschiedener Kommunikationsprotokollen

Bluetooth	**Bluetooth Low Energy**	**Wi-Fi 4 802.11n**	**Wi-Fi Ha-Low/IEEE802.11ah**
Mittlerer Energieverbrauch	Geringer Energieverbrauch	Hoher Energieverbrauch	Mittlere Energieverbrauch
Geringe Reichweite (bis zu 30m)	Mittlere Reichweite (bis zu 100m)	Mittlere Reichweite (bis zu 90m)	Hohe Reichweite (bis zu 1km)
In vielen Mikrocontrollern integriert	In vielen Mikrocontrollern integriert	In vielen Mikrocontrollern integriert	Kaum auf dem Markt vertreten - teuer
Peer To Peer	Server - Client	Server - Client	Server - Client

Mit dieser Übersicht haben wir uns schlussendlich für Bluetooth Low Energy entschieden, was uns schlussendlich auch zum Arduino Nano 33 BLE geführt hat, einer der wenigen Arduinos mit BLE 5.0.

5.7.2 Implementation

BLE verwendet eine Server-Client-Kommunikation, wobei wir bei Servern von Peripherals reden und bei Clients von Centrals. Ein Peripheral kann mehrere Services anbieten, welche wiederum mehrere Characteristics besitzen. Metaphorisch kann ein Service als Pinnwand und die Characteristics als Notizzettel beschrieben werden, die von den Centrals gelesen und/oder beschrieben werden können. Zusätzlich können Centrals auch auf Änderungen einer Characteristic horchen.
Mit der bereitgestellte BLE Bibliothek von Arduino konnten wir unseren Mesh Ansatz leider nicht verfolgen, jeden Mikrocontroller eine doppelte Rolle, Peripheral und Central gleichzeitig, zuzuweisen. Der in anderen Implementation realisierbare Ansatz wird auf den Nano 33 BLE leider nicht unterstützt. So haben wir uns für unser Modell auf eine klassische Server-Client-Kommunikation verständigt, wobei unser Hauptmodell in diesem Fall nur Events an unsere Rauten-Modelle sendet.

```
// Einschub aus neo_animation.hpp:
BLEService service;
BLEUnsignedCharCharacteristic characteristic;

boolean BLEPeripheralWrapper::init(void)
{
    !BLE.begin()
    BLE.setAdvertisedService(service);
    BLE.setLocalName(BLE_NAME);
    service.addCharacteristic(characteristic);
    BLE.addService(service);
    characteristic.writeValue(0);
    BLE.advertise();
}
```

Listing 7: Initialisierung des BLE-Peripherals, ohne Logs.

Sender (Peripheral)

Auf der Seite des Peripherals müssen wir das BLE-Modul selbst initialisieren und ein Service und ein Characteristic bereitstellen. Service und Characteristic benötigen dafür jeweils eine UUID, eine einzigartige Adresse, über die sie in unserem Fall auch von den Centrals gefunden werden. Diese sind als Konstanten im Code hinterlegt und werden durch das Makro ARDUINO_ID je nach Environment verwendet. Dabei ist vorgegeben, dass der Peripheral die ARDUINO_ID 1 besitzen muss, da diese verwendet wird, um nach jenem Peripheral zu suchen. Das Characteristic wird auf den von uns als Standardwert definierten Wert 0 gesetzt. Dieser Ablauf ist im Codebeispiel 7 zu sehen.

Wurde nun ein Einwurf detektiert und soll ein Event in die Characteristic geschrieben werden, so wird dies nur für ein bestimmten Zeitraum getan, bevor der Standardwert 0 erneut gesetzt wird. Das ist nötig um keine erneute Animationen der Centrals zu verursachen, falls zwischenzeitlich ein Verbindungsabbruch vorliegt. Daher wird sich hier ein Zeitstempel berechnet, an dem der Reset vollzogen werden soll. Daraufhin wird der übergebene Wert publiziert. (Siehe Codebeispiel 8)

Für das BLE-Modul benötigen wir, wie in Kapitel 5.2.2 erwähnt, eine update()-Methode. Diese wird benötigt um Interaktionen der Centrals zu bearbeiten. In unserer Implementation 9 wird hier zusätzlich der Timer überprüft und im Zweifelsfall der Standardwert zurückgesetzt.

```
void BLEPeripheralWrapper::publishEvent(u_int8_t val)
{
    ts_end = millis() + BLE_ADVERTISING_DURATION_MS;
    characteristic.writeValue(val);
}
```

Listing 8: Setzen eines Wertes der Characteristic des BLE-Peripherals, ohne Logs.

```
void BLEPeripheralWrapper::update()
{
    BLE.poll();
    if (millis() > ts_end)
    {
        characteristic.writeValue(0);
        ts_end = -1ul;
    }
}
```

Listing 9: Update des BLE-Peripherals, ohne Logs.

Empfänger (Central)

Auf der Seite der Empfänger, der Centrals, wird, solange dieser nicht gefunden wurde, nach dem Peripheral gescannt. Dafür wird die oben erwähnte UUID verwendet.

```
BLE.scanForUuid(UUIDS[PERIPHERAL_ARDUINO_ID].service);
```

Ab dem Moment ab dem der Central fündig wurde, werden verschiedene Stufen des Kontakts aufgebaut, es wird der Service und daraufhin das Characteristic gesucht. Der darauf folgendene Teil ist im Codebeispiel 10 gezeigt. Hier abonniert der Central die Characteristic um die Möglichkeit zu bekommen zu überprüfen, ob sich der gesetzte Wert verändert hat. Solange die Verbindung nicht abbricht und der Wert sich nicht ändert, wird weiterhin in der Schleife verblieben. Ab dem Moment an dem ein neuere Wert, der ungleich dem Standardwert ist, gelesen wird, wird eine Animation gestartet und angezeigt.

```
[...]
    if (characteristic
                && characteristic.canSubscribe()
                && characteristic.subscribe())
    {
        while (peripheral.connected())
        {
            if (characteristic.valueUpdated())
            {
                characteristic.readValue(val);
                if (val) leds.startAnimation(ANIMATION_BLUE_UP);
                while (leds.animationActive()) leds.update();
            }
        }
    }
[...]
```

Listing 10: Subscribe-Funktionalität skizziert.

5.8 Füllstandsmessung

5.8.1 Diskussion

Zum Messen des Füllstandes soll ein Sensor, oder eventuell auch mehrere, an der Unterseite des Tonnendeckels befestigt werden. Bei der Füllstandsmessung ist zu beachten, dass er Füllstand an unterschiedlichen Positionen in der Tonne eine unterschiedliche Höhe aufweisen kann. Somit existiert kein einzelner korrekter Füllstandswert, wie es bei einer Flüssigkeit der Fall wäre. Daraus ergeben sich verschiedene Möglichkeiten für die Ermittlung eines Füllstandswertes. Eine dieser Möglichkeiten wäre, mehrere Messungen an unterschiedlichen Punkten durchzuführen und daraus einen Mittelwert zu bilden oder den Punkt mit dem höchsten Füllstand als aktuellen Wert anzunehmen. Eine weitere Möglichkeit besteht darin lediglich eine Messung an einer fixen Position durchzuführen. Diese Methode besitzt zwar eine gewisse Fehleranfälligkeit, da der Füllstand an einem anderen Punkt höher sein könnte, liegt der Messpunkt jedoch mittig, sollte in den meisten Fällen dort auch der aktuell höchste Füllstand ermittelt werden, da sich die Oberfläche des Mülls in der Tonne häufig zu einem sehr flachen Kegel auftürmt. Der klare Vorteil der Umsetzung mit einem Sensor liegt jedoch in der Einfachheit der Umsetzung dieser Methode, da nur ein Sensor benötigt wird und nicht mehrere Sensordaten miteinander verrechnet werden müssen.

Es besteht zusätzlich die Möglichkeit auch für Füllstandsmessung den, im Abschnitt Einwurferkennung beschriebenen, Sensor TMF8821 von SparkFun zu verwenden, da die-

ser jedoch vergleichsweise hohe Kosten aufweist und eine Komplexität bietet, die von der Füllstandsmessung nicht zwingend gefordert wird, soll ein simplerer Sensor eingesetzt werden. Hierfür kommen sowohl optische als auch Ultraschallsensoren in Frage, die eine Reichweite von mindestens 140 cm besitzen, da dies in etwa der Gesamthöhe des Gerippten entspricht. Die Auswahl der Sensoren bei den von uns ausgewählten Zulieferern war im unteren Preissegment nicht besonders groß, weshalb wir uns für den Sharp GP2Y0A60SZLF Analog Distance Sensor auf einem Pololu Carrier entschieden. Dabei handelt es sich um einen optischen Sensor mit einer Reichweite von 10 cm bis 150 cm. Das Ausgangssignal des Sensors ist analog.

Die Füllstandsmessung ist eine Funktion des Gerippten, die in diesem Projekt jedoch keine weitere Außenwirkung aufweisen soll, und wird lediglich implementiert um das Konzept eines smarten Mülleimers zu verdeutlichen. In einer weiteren Umsetzung können die Daten zum Füllstand dann beispielsweise an eine zentrale Steuereinheit übertragen werden. Weitere Informationen zur möglichen Weiterführung des Projekts befinden sich im Kapitel Zukünftige Entwicklungsmöglichkeiten.

5.8.2 Implementation

Die Interaktion mit dem Sensor wird im Quellcode in der Datei level_meter.cpp implementiert. Diese implementiert wiederum die Headerdatei level_meter.hpp. Um eine Instanz des Sensors zu initialisieren muss dem Konstruktor zum einen der Anschlusspin des Arduinos übergeben werden und weiterhin die Abstandsmaße vom Sensor bis zum leeren Füllstand, sowie bis zum vollen Füllstand. Dabei ist zu beachten, dass der Abstandswert zwischen Sensor und vollem Füllstand mindestens 10 cm betragen sollte, da dies die minimale Messdistanz des Sensors ist.
Des weiteren existiert die Methode calculateDistance, welche den vom Sensor ausgegebenen Wert am Arduino Pin einliest und daraus die Distanz zum detektierten Objekt errechnet. Für die Implementation dieser Methode musste jedoch zuerst ermittelt werden, wie ein Signalwert in einen Distanzwert umgerechnet wird. Dazu wurde der Sensor auf einem Breadboard befestigt und mit dem Arduino verbunden. Der Sensor wurde auf einer ebenen Fläche so ausgerichtet, dass seine Messebene parallel zum Untergrund verläuft. Danach wurde vor dem Sensor eine weißer Karton im Abstand von 10 cm platziert. Der Arduino wurde so konfiguriert, dass der Sensorwert alle 25 ms 500 mal nacheinander ausgelesen wird. Dabei wird der Mittelwert der 500 Messungen berechnet und vom Arduino auf dem seriellen Monitor ausgegeben. Dieses Vorgehen wurde mit, in 5 cm Schritten, steigenden Abständen zum Karton wiederholt, bis zu einem maximalen Abstand von 150 cm. Die gemessenen Durchschnittswerte wurden im Programm Excel von Microsoft eingepflegt. Dort wurde aus den Daten ein Diagramm, inklusive Trendlinie und Trendlinienformel erstellt. Die Ergebnisse sind in Tabelle 5.3 und Abbildung 5.6 zu sehen.

Tabelle 5.3: Ergebnisse der Abstandsmessungen

Distanz (cm)	**Messwert**	**Distanz (cm)**	**Messwert**
10	574	85	238
15	423	90	235
20	343	95	231
25	310	100	226
30	303	105	223
35	300	110	219
40	294	115	216
45	287	120	214
50	279	125	211
55	273	130	209
60	265	135	207
65	259	140	204
70	253	145	203
75	249	150	202

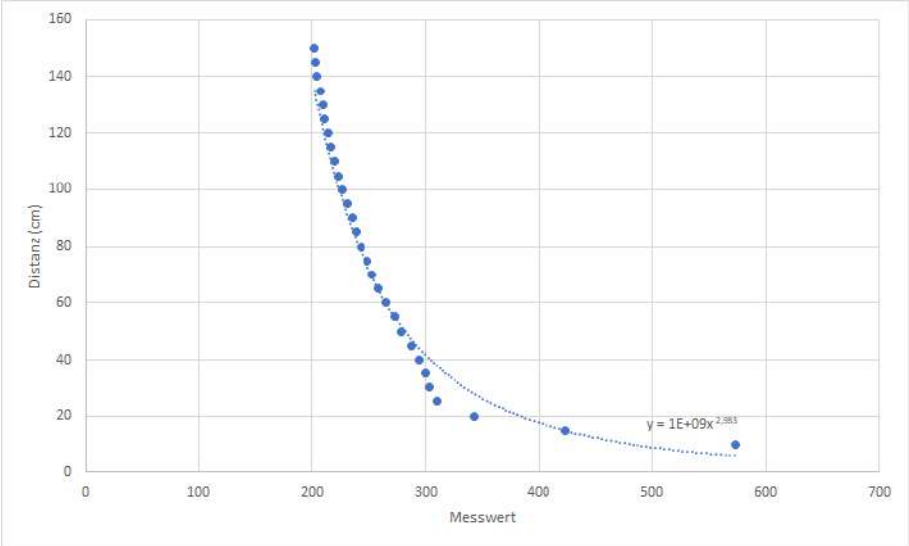


Abbildung 5.6: Von Microsoft Excel generierter Graph auf Basis der Messdaten

Es zeigt sich, dass die Trendlinie, insbesondere bei den kurzen Distanzen, den Messpunkten sehr gut folgt. Bei den mittleren Distanzen gibt es leichte Abweichungen, da dort auch die Messpunkte keinem klar erkennbaren Muster folgen. Das führt dazu, dass die Messungen, insbesondere in diesem Bereich, kein sehr genaues Ergebnis liefern werden. Jedoch werden die Abweichungen vom Ist-Zustand auch nicht so groß sein, dass die Mess-

werte für eine Auswertung, durch beispielsweise die FES, unbrauchbar sind. Der größte Nutzen der Füllstandsmessung, das Detektieren einer vollen Tonne, ist einwandfrei gegeben, durch die Genauigkeit der Trendlinie bei kurzen Distanzen.

Die für den Füllstandssensor implementierte Methode getFillLevel ruft die bereits angesprochene Methode liest den Messwert des Sensors aus und lässt anhand der bereits angesprochenen Methode calculateDistance die gemessene Distanz berechnen. Aus der gemessenen Distanz, der Distanz für den vollen Zustand und der Distanz für den leeren Zustand wird dann der Füllgrad berechnet und als abgerundeter, ganzzahliger Teil von 100 zurückgegeben.

5.9 Stückliste

Menge	**Bezeichnung**
4	Arduino Nano 33 BLE
1	Sparkfun Qwiic dToF Imager - TMF8821
3	Pololu 5cm Digital Distance Sensor
1	Sharp GP2Y0A60SZLF Analog Distance Sensor 10-150cm
30	Adafruit Neopixel RGBW 0.2W
1	Adafruit NeoPixel RGB LED Strip 30 LEDs/m 5m
2	Level Shifter, Joy-IT TXB0104
1	Netzteil POS-50-5-C 5V 10A
5	Solderable Breadboards
40 Meter	0.75mm^2 Kupferkabel - verschiedene Farben
1 Meter	1.5mm^2 Kupferkabel - verschiedene Farben
ca. 170	Crimps
ca. 50	Stecker

5.10 Schaltpläne

In den folgenden Grafiken wird die Verschaltung der Arduino Mikrocontroller mit den Sensoren und Leuchtmitteln dargestellt. Dabei ist in Abbildung 5.7 die Verschaltung der Komponenten der großen Gesamtmodells zu sehen, während Abbildung 5.8 den Schaltplan des kleinen Rautenmodells zur Präsentation der Bluetoothkommunikation zeigt.

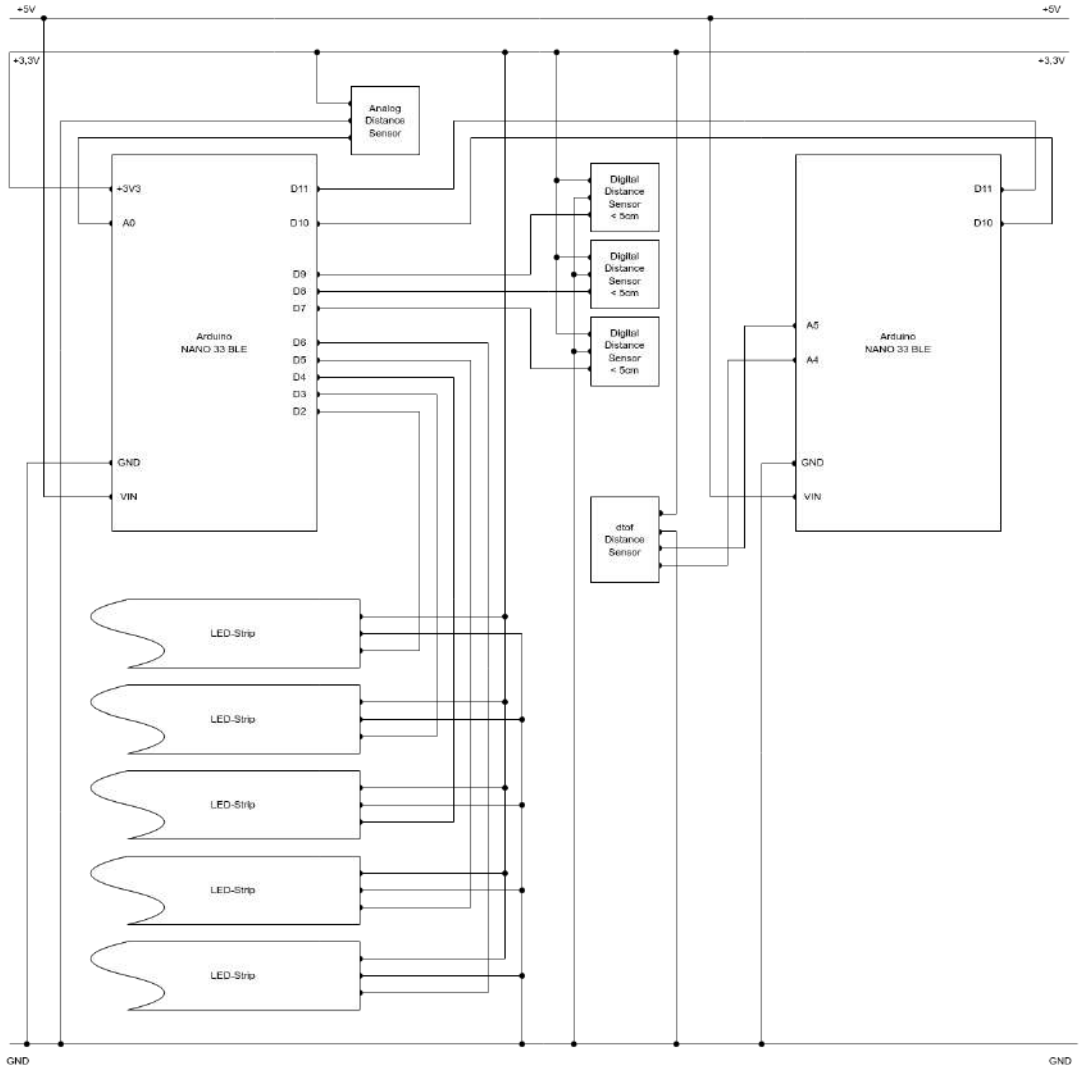


Abbildung 5.7: Schaltplan des Hauptmodells

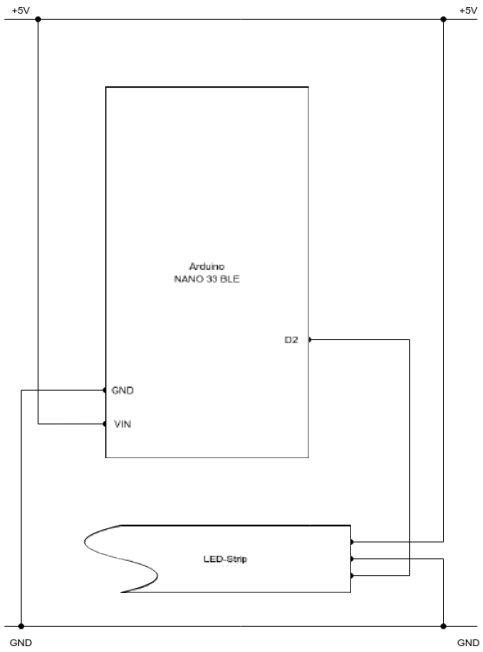


Abbildung 5.8: Schaltplan der zwei Rauten Modelle

5.11 Zusammenführung von Modell und Technik

Der Maschinenraum

Für die zwei Arduinos, das Netzteil und die Breadboards für die Steckverbindungen unserer Sensoren und LEDs brauchen wir im Mülleimer einen geschützten Ort. Wir haben uns dafür entschieden, diesen „Maschinenraum“ im Boden der Tonne zu platzieren, und einen um 10 Zentimeter erhöhten Doppelte-Boden einzubauen, auf dem dann die innere Mülltonne steht. So kann man die innere und danach die äußere Tonne nach oben rausheben, da beide Böden nicht fest mit der Tonne veranker sind. Damit kommen wir immer an unsere Technik. Dann habe wir am unteren Rand der Tonne noch ein Loch für unsere Stromverkabelung geschnitten, sodass das das einzige Kabel ist, was von außen sieht.

Verkabelung

Die restlichen Kabel sind an der Innenwand der äußeren Tonne nach oben gezogen und dann dort durch kleine Löcher nach Bedarf nach außen geführt.

Technik im Flaschenring

Die LEDs der Fächer im Flaschenring haben wir von unten außen an die Konstruktion geklebt und vorher passende Löcher in die Fächer geschnitten. Analog dazu sind wir bei

den 5cm Lidar Sensoren an der Rückseite der Fächer vorgegangen. (Zwischen Tonnenwand und Flaschenring.)

LED-Strips

Da die Rauten unterschiedliche Kantenlängen zwischen 8 und 10,5cm besitzen, die LED-Strips aber parallel zum Modellbau gelötet werden mussten, brauchten wir einen Ansatz der einen gewissen Spielraum ermöglicht. Ebenfalls unbekannt war, inwiefern die Rautenstruktur aus Schaumstoff anpassbar waren, da wir zu dem Zeitpunkt darüber noch keine Informationen bekommen hatten. Wir haben daher zwischen den geteilten LED-Strips längere Kabelverbindungen als nötig verwendet, denn während wir unter perfekten Umständen die Abstände der Übergange zwischen zwei Rauten hätten ausmessen können und wir die Kabel damit unter oder an den Rauten hätten verlegen können, müssen wir so die Länge der Kabel an jedem Übergang zweier Strips variieren können. Umgesetzt wurde das, in dem wir an jeder dieser Übergänge ein etwa 8mm dickes Loch in die Tonne gebohrt haben, um das übrige Kabel als Schlaufe in die Tonne zu führen. Die benötigte Drehung des LED-Strips um die Lötpunkte und Kabel in die Richtung des Lochs zu drehen, haben wir vorsichtig am LED-Strip selbst gebogen. Die langen Strecken, an denen das Kabel auf eine andere Höhe gebracht werden mussten (zu sehen in Abbildung 5.2) hatten wir ursprünglich geplant außen an der Raute langzuführen und an beiden Enden der Kante jeweils eine Schlaufe in die Tonne führen zu lassen, sodass wir die LED-Strips einfacher hätten abmontieren können. Unter der Voraussetzung, dass wir die Strips an dem Schaumstoff der Rauten festgeklebt wurde, haben wir uns aus ästhetischen Gründen dazu entschieden die Kabel an den langen Kanten in der Tonne langzuführen.

6 Ergebnis

6.1 Der Gerippte

Durch eine intensive Schlussphase konnten wir unsere drei anfänglich beschrieben Funktionen fertigstellen. (Siehe 3.2)
Wir haben drei unserer Flaschenringfächern mit Sensoren ausgestattet um die Funktionalität vorzuführen, während alle Fächer mit jeweils 2 LEDs ausgestattet wurden. So konnten wir ebenfalls die ästhetische „Flaschen-Krone“ präsentieren. Die weiß beleuchteten Flaschen haben den von uns erwünschten Eindruck erfüllt und es war ein belohnendes Gefühl den Flaschenring mit Flaschen zu füllen.
Ebenfalls fertiggestellt haben wir den Einwurfsensor, welchen wir kurz vor der Präsentation in der AMD im vollständig zusammengesetzten Zustand erfolgreich testen und dem anwesenden Prof vorführen konnten. Als Antwort auf einen Einwurf konnten wir eine Animation über die gesamte Raute abspielen. Dabei ist uns aufgefallen, dass der obere der drei Kabelstränge beim Einbau trotz Heißklebepunkten auf den Kontakten ein Wackelkontakt entwickelt hat, der die drei Farbwerte aus der Synchronität bringt. Da wir den Ursprung in der verleibenden Zeit nicht finden konnten, mussten wir uns entscheiden, nur eine der drei Farbkanäle anzusteuern. So haben wir uns auf für die Präsentation blaue Animationen geeinigt.
Ein weiteres Problem ist beim Abbau der Tonne für den Transport aufgetreten. Beim herausnehmen der inneren Tonne gab es erneut Probleme mit den LED-Strips. Diesmal gab es im unteren Kabelstrang Kontakt zwischen Strom und Erdung eine Verbindung, wodurch der LED-Strip extrem heiß wurde. Auch hier konnten wir leider keine Fehleranalyse mehr durchführen und mussten den unteren Teil für die anstehende Präsentation ausstecken.
Größtenteils erfolgreich umgesetzt haben wir die kleinen Rauten-Modelle, die andere Mülleimer simulieren sollten. Hier haben wir die Kommunikation zwischen dem Hauptmodell und einer Raute und parallel zwischen dem Hauptmodell und einem Handy, das die zweite Raute simuliert hat, mit dem in 5.7 beschriebenem Ansatz getestet. Wir haben es zu dem Zeitpunkt noch nicht hinbekommen, beide Rauten anzuspielen. Den genauen Grund konnten wir uns, ebenfalls wegen der kurzen Zeit zur Abschlussveranstaltung, dort noch nicht rekonstruieren.

Abbildung 6.1: „Der Gerippte“ bei der Abschlussveranstaltung. Auf dem Deckel die beiden Rauten.

6.2 Abschlussveranstaltung

(a) Unser Rauten-Modell in Aktion.

(b) Beleuchtung der Flaschen im Flaschenring.

Abbildung 6.2: Eindrücke aus dem Pitch der Abschlussveranstaltung

Die im betahaus stattfindende Veranstaltung hat am 30. Juni 2022 um 18 Uhr stattgefunden. Hier haben alle 5 Teams ihre Modelle und Konzepte in einem sieben minütigem Pitch vorgestellt. Für unser Team haben Celina Krug und Maybritt Braun einen Pitch

mit Unterstützung des Teams vorbereitet und dort hervorragend vorgetragen. Und das trotz des unglücklichen Umstands, dass der Einwurfsensor, nachdem er vorher Vorort getestet wurde, während der Vorstellung dauerhaft ausgelöst hat. Das hat einerseits nicht vollkommen mit dem Vorgetragenen überingestimmt, andererseits unsere Funktionalität leider viel einfacher dargestellt hat, als sie mit einer längeren Test-Kapazität fähig wäre. Der Grund der Fehlauslösungen liegt höchstwahrscheinlich in der Konfiguration des Sensors, denn auch wenn der maximale Abstand geringer eingestellt war, als die Kantenlänge des Sensorbereichs, kann es sein, dass diese Konfiguration im Sensor nicht fein genug umgesetzt werden kann und wir eine geringere Reichweite wählen müssen.

7 Fazit

7.1 Was lief gut

Hervorzuheben ist, dass die initiale Zusammenarbeit im Team hervorragend funktionierte. Die unterschiedlichen Kompetenzen wurden klar kommuniziert und berücksichtigt, woraus sich ein sehr harmonisches und produktives Arbeitsumfeld ergab. Im Ideenfindungsprozess wurde großer Wert auf die interdisziplinäre Zusammenarbeit gelegt, so konnte jeder seine Kompetenzen in die Bereiche der anderen Studiengänge erweitern und selbst ein Teil seines Wissens weitergeben, was zu einer guten gemeinsamen Wissensbasis führte. Dadurch gab es im gesamten Projektverlauf wenige fachliche Kommunikationsbarrieren und wenn welche auftraten konnten sie leicht beseitigt werden.

Des weiteren war die Ernennung einer geeigneten Projektmanagerin von großem Vorteil. Celina hat die Gruppe geführt, wenn Führung benötigt wurde, ohne dabei eine dominante Position einzunehmen. Insgesamt ist ihr Führungsstil sehr freundlich und offen, was ebenfalls sehr stark zum angenehmen Arbeitsklima in der Gruppe beigetragen hat. Zusätzlich zur hervorragenden Projektleitung hat das eingeführte wöchentliche Meeting inklusive kurzem Chek-In zu Beginn jedes Meetings, in dem jeder ein Update zum persönlichen Befinden geben konnte, das Gruppenklima positiv beeinflusst und dazu beigetragen, dass die Gruppenmitglieder sich, während längerer Phasen ohne Präsenztreffen, nicht aus den Augen verlieren.

Ebenfalls positiv hervorzuheben ist die Zusammenarbeit mit der FES, beziehungsweise genau genommen mit Herrn Jochen Schmitz. Jochen hat eine Begeisterung an den Tag gelegt, die das Team motiviert und bei Laune gehalten hat und stand immer für etwaige Fragen zur Verfügung.

7.2 Was haben wir gelernt

Das Ernennen von Verantwortlichen für bestimmte Teilbereich, insbesondere in der Prototyping Phase, hätte einige Vorteile mit sich gebracht. Der Verantwortlich sollte dabei jedoch nicht zwingend derjenige sein, der die Aufgabe selbst oder im Alleingang erfüllt, sondern das Kontrollorgan darstellen und überwachen, das bestimmte Teilaspekte zu bestimmten Deadlines erfüllt sind. Da solche Verantwortliche nicht benannt wurden, kam es oft zu Missverständnissen, was die Zeitpunkte für die Fertigstellung einzelner Komponenten anging.

Zudem sind die Einblicke in die Denkweise fachfremder Personen in einem solchen Projekt ein großer Erkenntnisgewinn. Das Projekt hat gezeigt, dass das Schaffen gemeinsamer fachlicher Grundlagen essenziell ist für eine erfolgreiche Kommunikation im interdisziplinären Kontext.

Weitere Lernaspekte liegen insbesondere im technischen Bereich, da wir beide Informatik studieren und mit der Einbindung von Sensorik und Aktoren in Software bisher keine Berührungspunkte hatten. Dadurch, dass wir in diesem Semester die Vorlesung zur Veranstaltung Interface-Technologie hören, konnten wir bereits erstes theoretisches Wissen in Bezug auf Sensorik sammeln und dieses nun durch praktische Anwendung in diesem Projekt vertiefen.

Zusätzlich hat das Projekt verdeutlicht, dass die Komponentenauswahl für einen solchen Prototypen weitaus besser geplant werden kann und viel Zeit in Anspruch nimmt, da es mit der einfachen Auswahl aus einem Produktkatalog nicht getan ist. Es ist durchaus legitim verschiedene Komponenten für die gleiche Aufgabe in der Praxis zu testen, um Vor- und Nachteile auch in der Anwendung der Komponenten ausfindigzumachen. Eine Auswahl nur anhand von Daten aus einem Datenblatt zu treffen kann in der praktischen Verwendung der Komponenten durchaus zu ungeahnten Komplikationen und Mehraufwand führen.

Zuletzt ist noch festzustellen, dass das Überwachen des Budgets unmöglich ist, wenn es zwar eine zentrale Stelle zur Überwachung des Budgets gibt, diese aber keine Information über voraussichtliche oder tatsächliche Kosten von Bestellungen erhält. Daraus resultiert, dass es wichtig ist Bestellprozesse vom Anfang bis zum Ende zu planen und dabei den korrekten Informationsfluss im Auge zu behalten.

8 Bewertung

Unsere Bewertung der Projektidee und des Prototypen fällt überwiegend positiv aus. Wir denken, dass das Konzept eines interaktiven und smarten Mülleimers an bestimmten Orten mehr Aufmerksamkeit auf sich ziehen und somit zur Reduzierung von Littering beitragen kann. Auch die Verwendung von Designaspekten die den Menschen im täglichen Leben begegnen und mit denen sie sich identifizieren können, trägt dazu bei, dass die Menschen ein solches Konzept annehmen werden. Zudem bietet der Designaspekt im Fall unseres Projekts ein Alleinstellungsmerkmal für die Stadt Frankfurt und unterstützt somit zugleich das Stadtmarketing.

Der vorgestellte Prototyp wurde jedoch keinem öffentlichen Praxistest unterzogen. Um die getätigten Aussagen zu untermauern muss der Prototyp zu einem vollständigen Produktprototyp weiterentwickelt werden, welcher in einem Frankfurter Park, oder auf dem Frankfurter Opernplatz aufgestellt werden kann um die Verwendung und Wirkung in der Praxis zu beobachten.

Zudem ist Anzumerken, dass interaktive und smarte Mülleimer nur an bestimmten Orten eingesetzt werden können und sollten. Dazu zählen insbesondere öffentliche Plätze und Parks. Zudem ist uns bewusst, dass unser Produkt nur einen geringen Teil zur Reduzierung des Müllproblems leisten kann. Eine wesentlich wichtigere Zielsetzung sollte es sein, die Menschen, dazu zu bringen weniger Müll zu produzieren und ihnen klar zu machen, wie wichtig fachgerechte Müllentsorgung ist. Dies kann durch Bildung und Aufklärung erreicht werden und dabei kann unser Gerippter bestenfalls unterstützend wirken.

9 Zukünftige Entwicklungsmöglichkeiten

Wie im Kapitel 6 erwähnt, wurde die Implementation eines Füllstandssensors vorbereitet, dieser jedoch aus Zeitgründen nicht in das Endprodukt integriert. Hier setzt sogleich die erste Möglichkeit zur Weiterentwicklung an. Der Füllstandssensor kann in den Mülleimer integriert werden und zusätzlich kann der Mülleimer mit dem LoRaWAN Netz der Stadt Frankfurt verbunden werden um darüber die verschiedenen Sensordaten zugänglich zu machen.

Ein weiterer wichtiger Schritt zum ausgereiften Produkt ist die Integration einer netzunabhängigen Stromversorgung durch Akkus und Solarzellen, damit der Mülleimer an den vorgesehenen Standorten autark installiert werden kann, ohne vorher größere Umbaumaßnahmen an den Standorten vorzunehmen.

Die wichtigste Weiterentwicklung ist jedoch vorerst die Entwicklung eines Modells aus beständigen Materialien, die in der Praxis eingesetzt werden können. In der Vorstellung des Projektteams lag dabei ein Rauten-Skelett aus lackiertem Metall, wobei die Rauten durch Plexiglas oder ein ähnliches Material ausgefüllt werden, wie es im kleineren Modell zur Präsentation der Bluetoothkommunikation angedeutet wurde.

10 Repository

Unser gesamter Codebasis, unsere Projekttagebücher, dieser Projektbericht und dessen Quellcode sowie weitere Materialien sind unter folgendem Repository zu finden: `https://github.com/RobinVonBerg/cic_ffm`

Abbildungsverzeichnis

[1] Quelle: Design Thinking Workshop CrossInnovationClass
[2] Quelle: https://cdn.sparkfun.com/assets/learn_tutorials/2/2/8/9/TMF882X_DataSheet.pdf