TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Elektro- und Informationstechnik Institut für Automatisierungs- Mess und Regelungstechnik

Projektarbeit

Hauptseminar AMR

Human Machine Interface

Sebastian Zarnack

Matrikelnummer: 4677584

Betreuer

Julian Rahm

Eingereicht am: 19. Januar 2020

Inhaltsverzeichnis

1. Analyse der Aufgabenstellung

1.1. Detaillierte Anforderungsbeschreibung

Ziel des Hauptseminars ist es, eine Mensch-Maschinen-Schnittstelle zur Bedienung, Visualisierung und Beobachtung der Aktionen des NXT-Roboters auf dem Spielfeld zu programmieren. Dies sollte über eine Android App ermöglicht werden. Die Kommunikation findet mittels einer Bluetoothverbindung mit dem NXT-Roboter statt. Als Endgerät wurde dabei ein 7 Zoll Android Tablet mit Android 4.4 gegeben. Die Schnittstelle sollte folgende Funktionen ermöglichen:

- Beobachtung: Anzeige der Spielfläche mit der aktuellen Position des NXT-Roboters, dessen vergangenen Pfad, den entdeckten Parklücken und Telemetriedaten des Roboters.
- Bedienung: Auswählen der Parkflächen sowie Aktivierung des vollautomatischen Ein-/Ausparkmanövers mittels der angezeigten Karte. Versetzen des Roboters in die vier verschiedenen Modi PAUSE, SCOUT, PARK-THIS, DISCONNECT.

Die Umsetzung der Schnittstelle soll sich dabei an den Anforderungen der Android Design Principles und Guidlines halten um eine stabile, nutzerfreundliche und gebrauchstaugliche Steuerung zu gestalten, ohne Einbußen bei der Funktionalität hinzunehemen.

1.2. Geplantes Vorgehen

Das Wichtigste um eine App zu designen, ist die anzusprechende Zielgruppe. Ohne eine Zielgruppe kann eine App nicht dem Nutzer angepasst werden, in etwa so wie ein Schneider wissen muss, welche Maße der Kunde hat, um ihm eine passende Hose zu nähen. Da uns keine genauen Spezifikationen gegeben waren, musste ich mir die Kriterien dafür selber zusammenstellen.
Außerdem ist das Grunddesign einer App wichtig. Dies sollte zu Beginn festgelegt werden, da es alle anderen Aspekte (Funktionalität, Bedienbarkeit...) miteinander vereint. Am Ende sollte ein erstes grobes MockUp der Nutzeroberfläche herauskommen. Zusammen mit dem MockUp wollte ich mir die Bedienbarkeit und die Hauptfeatures der App überlegen.

Um zu bestimmen welche Eigenschaften die App haben sollte und wie man diese Umsetzen könnte, plante ich mir die gegebene Beispielapplikation anzuschauen und auf dieser aufzubauen. Ich teilte die verschiedenen Aufgaben in kleine 'Untermodule' auf, die ich sukzessiv abarbeitete. Dies wollte ich durch einen iterativer Prozess von Hinzufügen einer Funktion, sowie Testen und Bewerten der Ästhetik und Funtktionalität erreichen. Durch leichte Parallelisierung dessen, erhoffte ich mir außerdem eine hohe Agilität und die Möglichkeit die verschiedenen 'Untermodule' im Zusammenspiel zu bewerten.
Abschließend sollten alle Funktionen erneut zusammen beobachtet und perfektioniert werden.

1.3. Schnittstellen und Zusammenarbeit

Als Schnittstelle zu den anderen Modulen ist ein Beispielprogramm gegeben, welches die vollständige Implementierung aller Methoden beinhaltet, um alle benötigten Informationen zur Erfüllung der in 1.1 genannten Anforderungen zu ermöglichen. Lediglich das Abfragen des Status und Setzen der Modi soll mit dem Verantwortlichen des Guidancemoduls abgesprochen werden. Mit allen anderen Modulen interagiere ich nicht direkt und lasse mir nur von diesen Daten liefern. Die Schnittstelle stellt für alle anderen Module die Möglichkeit bereit, ihre Arbeit durch die angezeigten Information zu validieren und Fehler einfacher zu finden.

2. Entwurf der Mensch-Maschine-Schnittstelle

2.1. Entwurfsprinzipien

Bezüglich der Zielgruppe entschloss ich mich dazu, die Applikation vorwiegend für Smartphones zu entwickeln. Diese sind heutzutage das wichtigste mobile Endgerät und quasi jeder besitzt - im Gegensatz zu einem Tablet - eines.
Des Weiteren ist ein Handy wesentlich handlicher und mobiler als ein Tablet, dass man nicht in der Hosentasche mit sich herum tragen kann. Die Zielgruppe grenzte ich weiter auf Nutzer mit Smartphones der jüngeren Generation ein. Diese besitzen modernere Androidversionen, welche mir ermöglichen die modernere Android-API zu nutzen und die App zeitgemäßer zu gestalten.
Genauer gesagt besitzt die App ihre volle Funktionalität nur für Androidversionen ab API Level 21, dies entspricht Android 5.0 (Android Lollipop). Damit schließe ich weltweit etwa 10,7 Prozent aller Androidnutzer aus [1]. Dies ist eine für mich vertretbare Zahl, da ich mich zusätzlich dazu entschloss, dass die App auch eine Basisfunktionalität auf Geräten mit einem API Level kleiner 21 zu gewährleisten hat.
Da ein Smartphone im Vergleich zu einem Tablet wesentlich weniger Bildschirmfläche bereitstellt, musste ich einen Weg finden, die Fülle an Informationen sinnvoll und übersichtlich darzustellen.
Hierbei entschied ich mich dafür, die App nur im Portraitmodus zu betreiben. Dies erachte ich als sinnvoll, da meiner Meinung nach ein Handy abseits von Spielen größtenteils nur im Portraitformat genutzt wird. Der Portraitmodus stellt bei kleinen Bildschirmen die beste Möglichkeit dar viele Informationen auf einmal übersichtlich anzuzeigen. Dies ist daran ersichtlich, dass beliebte Nachrichten oder Kommunikationsapps wie z.b. Google News oder Telegram für das Hochformat designt sind.
Als Designprinzip verwendete ich das Material Design, welches 2014 von Google vorgestellt wurde [2]. Das Material Design hat sich als Ziel gesetzt, eine klare, intuitive und moderne Designsprache durch Richtlinien, Tools, vorgefertigten Komponenten und Open Source Code zu erschaffen [3].
Ein wichtiger Aspekt des Material Designs ist die in der App verwendete Farbpalette. Diese setzten sich grob aus 3 verschiedenen Farben zusammen:

- die Primary Color, welche die Hauptfarbe der App darstellt und damit die gesamte Farbgebung bestimmt
- die Accent Color die bestimmte Bedienelemente oder kleinere Abschnitte hervorhebt und Akzente setzt
- die Primary Text Color, welche die Schriftfarbe ist

Weitere Farben sind die Dark-/Light PrimaryColor, Divider-, Primary Text-, Secondary Text, Iconcolor. Diese sind dunklere oder hellere Varianten der PrimaryColor.
Die Farbpalette und deren Farben/Farbkombinationen können nicht beliebig gewählt werden, sondern sind durch die Material Design Richtlinien vorgegeben. Diese Vorgaben vermitteln, welche Farben gut miteinander harmonieren und ein stimmiges Gesamtbild ergeben.
Alle Icon- und Buttonelemente, sowie die Interatkion mit diesen, wurden nach den Richtlinien des Materialdesigns erstellt [4].

2.2. Funktionen des HMI

Die Hauptfunktion des HMI ist es, eine möglichst angenehme Bedienung des Roboters zu erlauben und gleichzeitig den Benutzer mit aktuellen Informtionen des Roboters zu versorgen, ohne ihn dabei abzulenken oder zu überfordern.
Da ich mich außerdem für ein Smartphone als Hauptendgerät entschieden habe, musste ich Lösungen finden, viele Informationen auf kleinem Platz darzustellen. Eine beliebte Lösung, wie sie auch von Google Maps oder Nextbike genutzt wird, ist ein Aus- und einklappbares Panel. Dadurch können die für die Bedienung zweitrangigen Funktionen/Informationen ausgeblendet werden.
Im zugeklappten Panel zeigt es nur die wichtigsten Bedienelemente an, im aufgeklappten kommen die Telemetriedaten hinzu. So kann der Nutzer selber entscheiden wann er mehr Informationen bekommen möchte. Im zugeklappten Zustand wirkt die App so sehr übersichtlich und aufgeräumt und kann dennoch alle in 1.1 genannten Anforderungen erfüllen.

3. Implementierung der Mensch-Maschine-Schnittstelle

3.1. Implementierung des visuellen Designs

3.1.1. Vorgehen

Ich plante meine App in drei Bereiche und zwei Fenster aufzubauen.

- Das Hauptfenster um den Roboter zu kontrollieren mit
  - der aktuellen Statusanzeige im oberen Bereich
  - der Karte des Spielfelds
  - dem Panel mit der Bedienung des Roboters sowie optionaler Anzeige weiterer Daten
- Das Bluetoothfenster zur Auswahl der Bluetoothverbindung mit
  - der aktuellen Statusanzeige im oberen Bereich
  - allen gekoppelten Geräten in der Mitte
  - sonstigen Bedienelemente

Bei der weiteren Konzeptphase nutzte ich ein Tablet mit Stifteingabe. Das Endergebnis mit allen Möglichen Panelzuständen ist in Abbildung 3.1 ersichtlich:

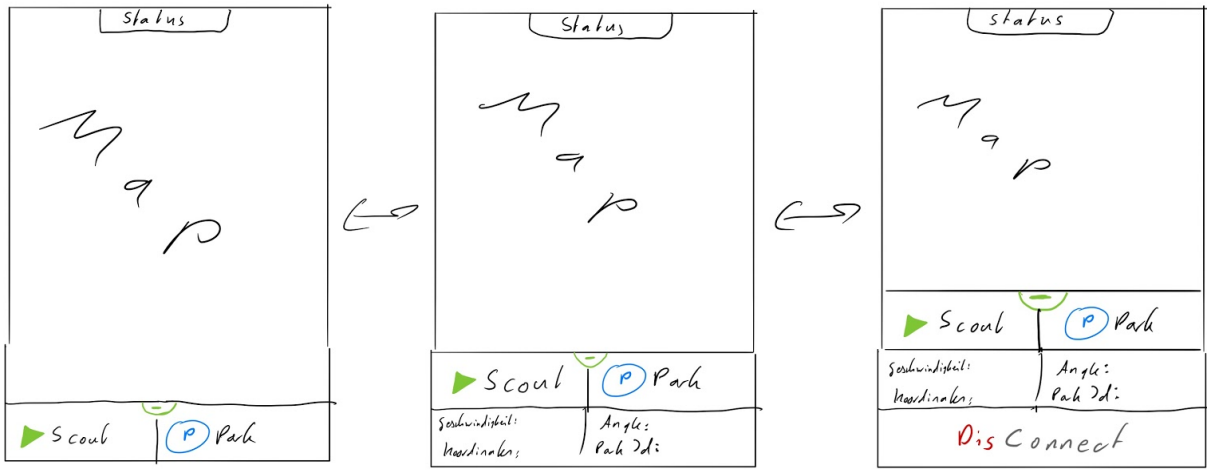

Abbildung 3.1: Erstes Sketch-Up

Anschließend übertrug ich dieses MockUp mittels Photoshop in eine realitätsnahe Version. Diese beinhaltet auch ein erstes Farbkonzept. Um die richtige Kombination an Farben zu finden, nutzte ich die Website 'materialpalette.com' [5] und erstellte mehrere Versionen mit unterschiedlichen Farbgebungen, die ich anschließend von mehreren Freunden bewerten ließ. Das Ergebnis mit der beliebtesten Farbkombination ist in Abbildung 3.2 ersichtlich.

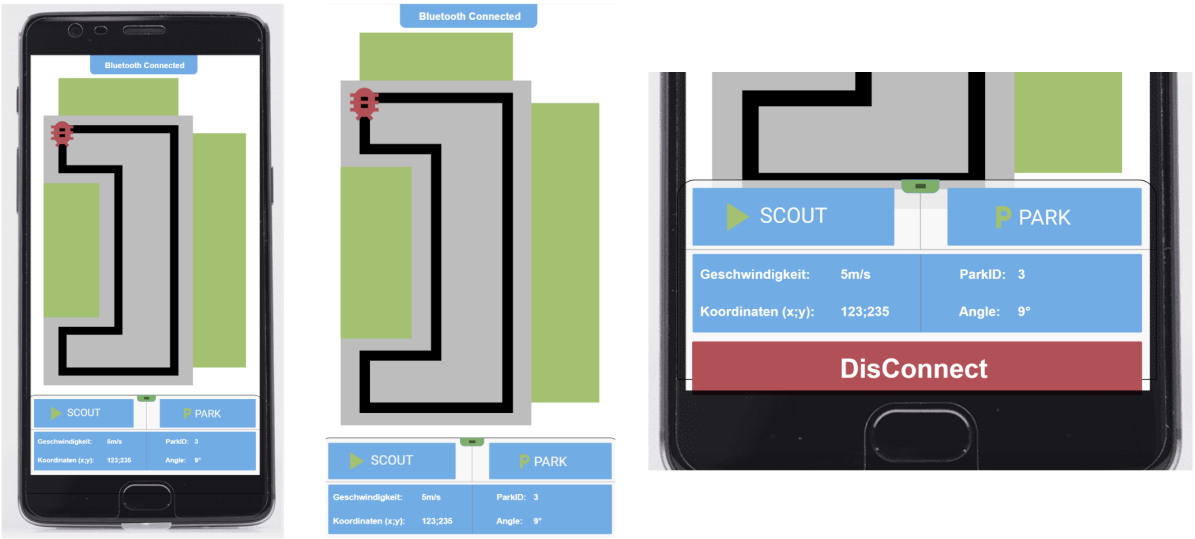

Abbildung 3.2: Erstes Sketch-Up im Appdesign

Ich entschied mich anfangs dafür ein dreiteiliges Panel anzufertigen, welches im eingeklappten Zustand nur zwei Buttons anzeigt.
Diese beide Knöpfe hatten mehrere Funktionen. Der linke dient alleinig zur Bewegungssteuerung des Roboters. Über ihn sollte der Roboter vom aktuellen Modus in den Pausemodus und vom Pausemodus in den vorherigen Zustand gewechselt werden.
Der rechte Button dient für die Steuerung der Parkvorgänge. Über ihn sollte der Roboter in den Ein-/Ausparkmodus gewechselt werden.
In der zweiten Panelstufe sollten nur Telemetriedaten angezeigt werden.
Die dritte Stufe diente als 'indirekter Disconnect Button'. Hiermit sollte über einen 'Pull and Release' Mechanismus des Panels die Bluetoothverbindung getrennt werden. Des Weiteren besitzt das Panel einen kleinen visuellen Balken in der Mitte um einen Angriffspunkt für die Bewegung des Panels zu bieten und im eingeklappten Zustand zu verdeutlichen, dass dieser Bereich beweglich ist.
In der Mitte der App sollte die gesamte Zeit die Karte präsent sein. Sie wurde um 90 Grad gedreht, um sie möglichst großflächig darstellen zu können.

In der oberen Hälfte ist die Statusleiste um den aktuellen Status der App und des Roboters zu präsentieren. Sie ist einer Notch nachempfunden, die inzwischen bei vielen Smartphones der neueren Generation präsent ist und damit nicht allzu disruptiv wirken sollte.
In den MockUps ist nicht dargestellt, dass die Bluetoothverbindung über einen extra Button in der oberen rechten Ecke erfolgen sollte. Außerdem war geplant, die Parklückenauswahl durch zur Laufzeit auf der Karte generierte Parkbereiche zu ermöglichen.

3.1.2. Ergebnis

Als erstes veränderte ich die dem Android Studio zugrunde liegenede Support-Library von Appcompat-V7 zu AndroidX. Diese ist wesentlich moderner und damit zukunftssicherer. Außerdem ermöglicht sie mir, alle Bedien- und Anzeigeelemente im Material Design zu integrieren. Dies ermöglichte unter anderem ein modereners Design der Buttons inklusive einer Anklickanmitation ('Ripple Effect'). Diese liefert ein viseulles Feedback durch ein über den gedrückten Button laufenden Schatten.
Des Weiteren werden alle Knöpfe ausgegraut und die Klickanimation deaktiviert, wenn sie nicht verwendet werden sollen
Bei der Farbwahl entschied ich mich schlussendlich als Primary Color ein blaugrau (607D8B), als Accent Color ein Rotorange (FF5722) und als Text Color die Farbe weiß (FFFFFF) zu wählen.
Insgesamt habe ich versucht die Farbgebung der Karte der Realität anzugleichen, also deckende graue Töne mit hellen Farben als Kontrast. Um etwas Farbe in die App hereinzubringen und so die wichtigen Elemente visuell abzuheben, entschied ich mich für das rotorange. Dadurch entsteht eine Farbpalette die nicht aufdringlich wirkt, aber trotzdem die wichtigsten Elemente hervorhebt und herausstellt.
Alle anderen, hier nicht angegebenen Farben, sind von dieser Farbpalette abgeleitet und gehen mit dem vom Material Design vorgegeben Richtlinien konform.
Der Hintergrund der gesamten App ist weiß, da dies eine einfache nicht ablenkende Farbe ist und einen guten Kontrast zu allen anderen verwendeten Farben bietet. Der Panelhintergrund ist die helle Version der Primary Color und grenzt sich so angenehm von dem weißen Hintergrund ab.
Die Implementierung der Karte gestaltete sich als aufwändig aber nicht allzu kompliziert. Ich nutzte als Basismaterial ein Screenshot der Karte aus der PowerPoint Präsentation der Aufgabenstellung. Diese Karte teilte ich in drei Unterkarten auf und färbte sie in Android Studio jeweils mit einer anderen Farbe ein:

- die zu folgende Linie - Farbe: schwarz
- die unmittelbare Umgebung der Linie - Farbe: grau (Divider Color)
- die möglichen Parkflächen - Farbe: Primary Color

Dadurch erreichte ich eine hohe Flexibilität bei der späteren Farbwahl der einzelnen Flächen. Des Weiteren glättete ich die Kanten der Karten und entfernte störende Objekte. Anschließend ordnete ich sie in Android Studio in einem einzelnen RelativLayout zueinnander an und färbte sie wie gerade beschrieben ein.
Anschließend kümmerte ich mich um das auf und zuklappbare Panel. Hierzu nutzte ich eine Vorlage von dem Gitnutzer umano [6]. Während der Implementierung entschied ich mich dafür, dass die Karte mit der Höhe des Panels mitskaliert. Zuallererst wollte ich die Skalierung in X Richtung fixieren und nur die Y-Richtung skalieren/verzerren. Das Ergebnis sah allerdings nicht sehr vielversprechend aus, sodass ich mich für eine gleichmäßige Skalierung entschied. Dies lieferte ein gutes finales Ergebnis.
Beim bestücken des Panels stellte sich der 'Pull and Release' Mechanismus zum Trennen der Bluetoothverbindung als ein für mich nicht lösbares Problem heraus, weshalb ich diesen verwarf. Den Rest des Panels behielt ich wie in der Konzeptzeichnung gezeigt bei. Die Buttons erhielten vor ihrer Beschriftung kleine Icons um ihre Funktion zu verdeutlichen. Die Anzeige der Telemetriedaten gestaltete ich in Ihrer Formatierung etwas anders um einen besseren Überblick zu ermöglichen.
Eine weitere Änderung fand bei der Statusanzeige im oberen Bereich der App statt. Eine Notch und die angrenzenden Bedienelemente zur Steuerung der Bluetoothverbindung erwiesen sich als nicht sehr praktikabel. Stattdessen verwendete ich eine Toolbar. Diese ist - im Gegensatz - zur Notch ein fester standardmäßiger Bestandteil von Android und ist - aus persönlicher Erfahrung heraus - in sehr vielen Androidapps anzufinden. Über diese konnte ich die Statusanzeige, sowie den aus dem Panel verschwundenen Bluetoothknopf mittels eines Icons realisieren. Die Toolbar bietet zudem ein sogenanntes Overflow Menü, in dem über ein Drop Down Menü weitere Auswahlmöglichkeiten versteckt werden können. Das Icon des Bluetoothknopfes ist das Bluetoothsymbol, dass sein Aussehen leicht verändert, wenn eine Verbindung hergestellt wurde (je ein Punkt links und rechts des Bluetoothsymbols).
Um den Standort des Roboters anzuzeigen, entschied ich mich für ein simples Icon, welches ein 'Bug' (Wanze) präsentiert, der über die Spielfläche 'krabbelt'. Dieser Wanze folgen kleine runde Punkte, die nach einiger Zeit verschwinden, um den zurückgelegten Weg zu kennzeichnen.

Diese bestehen dabei aus einem Farbverlauf von der Akzentfarbe zu der Primary Color. Das rotorange (Akzentfarbe) ist dabei dem Roboter zugewandt um besser zu kennzeichnen von wo der Roboter kommt (bzw. wohin er gegangen ist).
Als Parkflächen entschied ich mich für ein einfaches großes 'P' mit Begrenzungen oben und unten um die Dimensionen der Parklücke erkenntlich zu machen. Dieses kreierte ich als Bild und nutzte dieses als Oberfläche für ImageButtons. Dadurch können die Parkflächen direkt auf der Karte angezeigt und ausgewählt werden.
Um die Bluetoothverbindung herzustellen, nutzte ich das vom Beispielprojekt verwendete Auswahlfenster und fügte nach den Design Prinzipien des Material Designs einen 'Cancel' als TextButton und einen 'Disconnect' als OutlinedButton in der rechten unteren Ecke hinzu.
Die Akzentfarbe besitzen alle Buttons, die Toolbar, der Roboter auf dem Spielfeld, der Hintergrund des Launcher Icons, sowie die horizontale kleine Leiste in der Mitte der oberen Begrenzung des Panels.
Die folgenden Abbildungen 3.3 und 3.4 zeigen das finale Appdesign. Im Abbildungsverzeichnis können weitere Bilder gefunden werden (A.1 bis A.6).

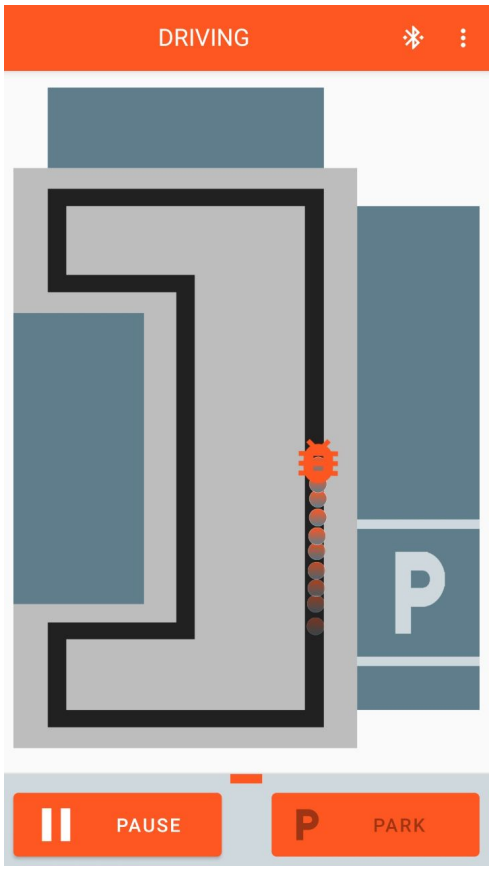

Abbildung 3.3: Finales App Design - Hauptfenster

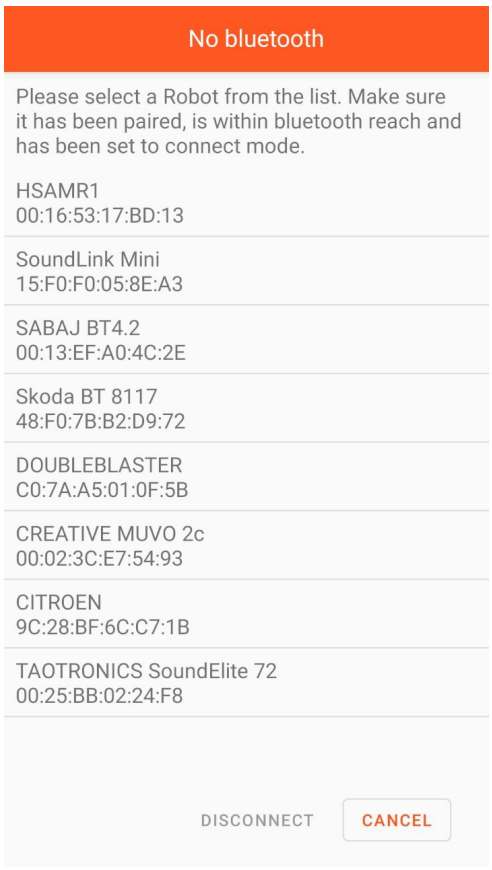

Abbildung 3.4: Finales App Design - Bluetoothfenster

Als Launcher Icon wählte ich ein Bild eines Legoroboters [7] aus. Daneben platzierten ich ein dem WiFi-Symbol ähnliches Zeichen um den Zweck der App zu verdeutlichen.
Auf dem Roboter platzierte ich ein PixelArt eines Meerschweinchen [8], da wir als Gruppe unseren Roboter (liebevoll) Meerschweinchen nennen. Dieses ist im Android Launcher kaum bemerkbar und kann deshalb als Easter-Egg angesehen werden.
Als Hintergrundfarbe verwendete ich die Akzentfarbe der App um ein einheitliches Gesamtdesign zu schaffen und das Icon im Launcher hervorstechen zu lassen. Insgesamt ist das Icon nach den Vorgaben von Android erstellt und wurde in allen von Google geforderten Formaten (rund, eckig, verschiedene Auflösungen...) angefertigt.

Abbildung 3.5: runde Version des Launcher Icons

3.1.3. Besondere HMI-Merkmale

Als besonders herausstechendes Merkmal sehe ich das Panel an. In dieser Form bietet das (meines Wissens nach) keine andere Gruppe an. Es orientiert sich an moderenen Apps wie Google Maps oder Nextbike und bietet eine ideale Lösung um viele Informationen auf einer stark begrenzten Fläche darzustellen. Es zeigt außerdem wie leicht es heutzutage ist, moderne Feature zu implementieren, ohne Erfahrung in der Entwicklung von Apps zu haben. Des Weiteren kann die App, trotz des Material Designs, ohne Funktionseinbuße auf Geräten mit einem API Level kleiner 21 und geringer Bildschirmauflösung verwendet werden. Bei Geräten mit einer Android Version unter Lollipop fehlt nur der Animationseffekt beim Betätigen eines Buttons und es kommt zu einer fehlerhaften Iconanzeige.

3.2. Implementierung der Funktionalität

3.2.1. Vorgehen

Erst nach der ersten zufriedenstellenden Implementation des Designs kümmerte ich mich um die Funktionalität.
Ich entwarf ein Flussdiagram um die Zustände der Buttons darzustellen (Abbildung 3.6).

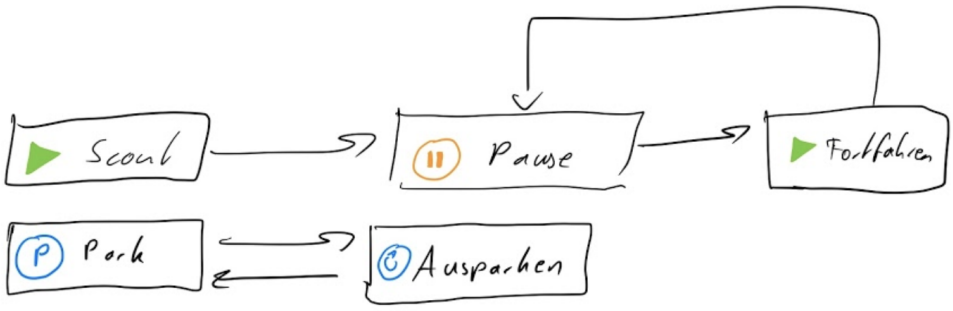

Abbildung 3.6: Flussdiagram für das Verhalten der Knöpfe

Dieses Flussdiagramm implementierte ich in das bestehende Design. Des Weiteren verknüpfte ich den Bluetooth Button um eine erste Verbindung mit dem Roboter herstellen zu können. Dadurch ermöglichte ich meiner Gruppe eine Basisfunktionalität zum Testen ihrer Module.
Danach kümmerte ich mich um die richtige Funktionalität des Panels und die dazugehörige Skalierung der Karte. Anschließend ging es um die Visuallisierung der Position des Roboters, dessen vergangener Pfad, sowie der Anzeige von Telemetriedaten im Panel. Dadurch konnte das Modul Navigation einfacher testen, ob die angezeigte Position des Roboters mit der Realität übereinstimmt. Um dies zu erreichen musste ich einen Konverter erstellen, der mir die von der Navigation in Zentimeter gelieferten Angaben in - abhängig von der aktuellen Größe der Karte - Pixel umrechnet.
Zwischendurch änderte ich immer wieder Feinheiten bei der Farbdarstellung und dem allgemeinen Design.
Die Toolbar wurde im weiteren Verlauf hinzugefügt.
Schlussendlich kümmerte ich mich um die Darstellung der Parklücken. Dies gestaltete sich schwieriger als gedacht, da die Navigation zu dem Zeitpunkt noch keine Parklücken liefern konnte. Aus diesem Grund schrieb ich mir eine eigene Testklasse über die ich mir per Knopfdruck Parklücken anzeigen lassen konnte. Da die Lücken rechts, links und oben platziert werden konnten, benötigte ich eine Fallunterscheidung, da alle Bereiche unterschiedliche Parameter aufweisen. Die Parklücken platzierte ich, indem ich zur Laufzeit einen ImageButton erstellte, diesen mit dem Parklückenbild versah (siehe Anhang A.7) und anschließend den Button auf die richtigen Dimensionen vergrößerte. Dadurch wird auch das Bild auf dem Button verzerrt und so eine visuelle Anzeige der Dimensionen der Parklücke ermöglicht. Anschließend implementierte ich die Übergabe der ausgewählten Parklücke an die Guidance.
Bisher hatte ich alles nur in einer großen Klasse mit vielen Methoden programmiert. Da dies keine gute Programmierstil ist, brach ich nach der ersten Verteidigung den bisherigen Code in vier Klassen auf (Refactoring):

- die ’Mainactivity’ über die die Activity/die App gestartet wird und alle anderen Klassen bedient werden
- die ’Map’, welche alle Aktivitäten auf der Karte selbst verwaltet
- die ’Parking’ welche alle ParkingSlots und deren Updates verwaltet
- die ’Converter’ welche zum konvertieren von Zentimeter in Pixel dient (Abhängig von der Bildschirmauflösung und der aktuellen Größe der Karte)

Das UML Diagramm kann im Anhang A.8 gefunden werden. Nach einem Hinweis von einem Kommilitonen, ersetzte ich das einzelne Runnable mit dem enthaltenden Timer durch die aktualisierte Version von Runnables, welche es ermöglichten, mehrere Runnables mit verschiedenen Pausenzeiten zu erstellen. Dadurch konnte ich die Häufigkeit der Methodenaufrufe der einzelnen Klassen priorisieren und eine bessere Performance erreichen.
Zu allerletzt verbesserte ich das Bluetoothfenster um die zwei Buttons zum Trennen der Verbindung und Abbrechen der Verbindungsauswahl. Außerdem fügte ich in der Toolbar über das Drop Down Menü die Möglichkeit hinzu den Pfad des Roboters zu de-/aktivieren. Dank der Hilfe eines Weiteren Kommilitionen, sowie meinem Betreuer konnte ich im gegebenen Backend die Möglichkeit hinzufügen, auch die von der Navigation aktualisierten Parklückendaten zu empfangen. Dies ermöglichte mir, die Dimensionen der bereits erkannten Parklücken zu aktualisieren, wenn sich diese in der Realität verändert hatten.
Zwischen all diesen Schritten veränderte ich immer wieder kleine Details am Design, wie zum Beispiel die Anordnung der Telemetriedaten im Panel.

3.2.2. Ergebnis

Insgesamt habe ich eine performante und stabile App programmiert. Zu Beginn sind alle Buttons außer der Bluetoothbutton deaktiviert. Über diesen kann in das Bluetoothfenster gewechselt werden. Hier kann eine Verbindung ausgewählt und hergestellt, ohne Aktion zum Hauptfenster zurückgekehrt oder eine bestehenden Verbindung getrennt werden. Wird eine bestehende Verbindung getrennt, kehrt auch die App in ihren Anfangszustand zurück. Das Panel kann ohne Probleme zwischen zwei Zuständen bewegt werden, die Karte skaliert dabei mit, sodass kein ungenutzer Leerraum entsteht. Die Anwendung kann die aktuelle Position des Roboters, sowie dessen Pfad (an-/ausschaltbar) anzeigen. Parklücken werden während der Laufzeit hinzugefügt und bei erneuter Vermessung ihre Dimensionen auch visuell aktuell gehalten. Es werden nur zum einparken passende Parklücken angezeigt. Sollte eine Parklücke von passend(/nicht passend) zu nicht passend(/passend) bei einem erneuten Vermessungsvorgang wechseln, wird sie auch auf der Karte nicht weiter/(erneut) angezeigt. Es kann jederzeit nur eine Parklücke ausgewählt werden, diese wird dann deutlich durch eine Farbänderung (zur Akzentfarbe) erkenntlich gemacht. Der Button "Parkingïst nur klickbar, wenn eine Parklücke ausgewählt ist. Wurde der Roboter in den Einparkmodus (PARK-THIS) versetzt, blinkt die ausgewählte Parklücke während des gesamten Einparkorgangs. Dadurch ist jederzeit erkenntlich ob und wo der Roboter einparkt. Während der Roboter sich in einem beliebigen Parkmodus befindet, können keine anderen Parklücken ausgewählt werden. Alle Buttons werden je nach Zustand des Robo-

ters/der App de-/aktiviert. So kann kein 'unmöglicher' Zustand erreicht werden. Ein Beispiel dafür ist der Disconnect Button in der Bluetoothübersicht. Dieser ist solange deaktiviert, wie keine Verbindung hergestellt wurde. Die beiden Buttons zur Kontrolle des Roboters dahingegen werden erst nach der Herstellung einer Verbindung aktiviert.

3.2.3. Besonderheiten der Umsetzung

Eine Besonderheit der App ist die Darstellung der aktuellen geschätzten Geschwindigkeit des Roboters. Dies habe ich durch ein eigenes Runnable in der Map Klasse erreicht. Diese errechnet alle 400 Millisekunden welchen Weg der Roboter in dieser Zeit zurückgelegt hat und stellt diesen Wert im Panel dar.
Auf Programmtechnischer Seite kann die App sehr einfach erweitert werden. Ich habe verschiedene Variablen so angelegt, dass sie leicht durch zukünftige Bedienelemente verändert werden können. Zum Beispiel ist die Länge des angezeigten Roboterpfades über eine einzelne Klassenvariable änderbar.
Durch die spezielle 'Converter' Klasse skaliert die App alle Elemente problemlos auf alle Bildschirmauflösungen. Außerdem läuft die App ohne Probleme und ohne sich zurückzusetzten im Hintergund.

4. Sonstiges, Anmerkungen und Verbesserungsmöglichkeiten

Nach viel Debugging habe ich es geschafft den Bluetoothverbindungsaufbau auf Geräten unterschiedlichster Androidversionen zu stabilisieren. Dazu muss ein kleiner Delay von 500ms in die run() Methode der ConnectThread.java eingebaut werden. Die Abbildung 4.1 zeigt die genaue Position.

```
50
51          // If connection can be established connectTo() returns true
52          if (connector.connectTo(hmi.nxtName, hmi.nxtAddress, NXTCommFactory.BLUETOOTH)) {
53              hmi.dataIn = new DataInputStream(connector.getInputStream());
54              hmi.dataOut = new DataOutputStream(connector.getOutputStream());
55              if (hmi.dataIn == null)
56              {
57                  hmi.connected = false;
58                  Log.e(TAG_BT_CONNECT, msg: "Connection failed.");
59              }
60              else
61              {
62                  // add delay here
63                  try {
64                      Thread.sleep( millis: 500);
65                  } catch (InterruptedException e) {
66                      e.printStackTrace();
67                      Log.e( tag: "sleep",  msg: "skipped");
68                  }
69                  hmi.bTCommunicationThread = new BTCommunicationThread(hmi);
70                  hmi.bTCommunicationThread.setName("readerThread");
71                  hmi.bTCommunicationThread.setDaemon(true);
72                  hmi.bTCommunicationThread.start();
73                  hmi.messenger = new Messenger(hmi.messageHandler);
74              }
75              hmi.connected = true;
```

Abbildung 4.1: Position des Delays in der ConnectThread.java

Mit diesem Fix läuft der Verbindungsaufbau auf allen Geräten problemlos. Getestet wurden mehrere Smartphones mit Android 9 und 6, sowie das bereitgestellte Tablet.

An der App würde ich hauptsächlich das Anzeigen der Parklücken verbessern. Das Verzerren des Bildes sieht bei sehr großen und sehr kleinen Parklücken unschön aus. Eine leichte Lösung wäre es, den transparenten Hintergrund durch einen farbigen zu ersetzen und das Park Icon zu zentrieren. Darauf habe ich jedoch verzichtet, da ich einen transparenten Hintergrund mit farbigen Begrenzungen oben und unten haben wollte. Des Weiteren ist das deaktivieren des Pfades des Roboters nicht ganz fehlerfrei. Hier hat mir allerdings die Zeit gefehlt dies zu beheben.

Zu guter Letzt verändere ich viel Text nicht mittels Stringresorucen sondern direkt über Strings. Dies ist zwar problemlos möglich, laut Android aber kein gute Methode, da es ein mögliches Übersetzten der App in andere Sprachen wesentlich erschwert.

Allgemein würde ich mir wünschen, dass das gegebene Backend für die Datenübertragung besser, beziehungsweise überhaupt dokumentiert wird, da hier doch einige Änderungen von meiner/unserer Seite nötig waren.

A. Anhang

Abbildung A.1: Startzustand

Abbildung A.2: Startzustand mit hergestellter Bluetoothverbindung

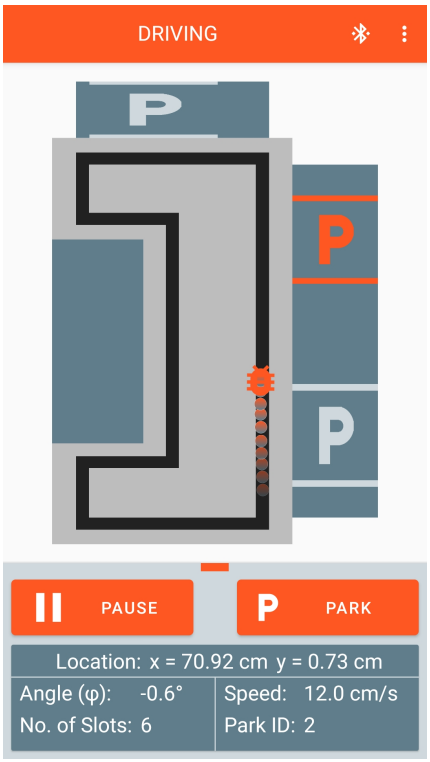

Abbildung A.3: Roboter in Bewegung mit erfassten und ausgewählter Parklücke

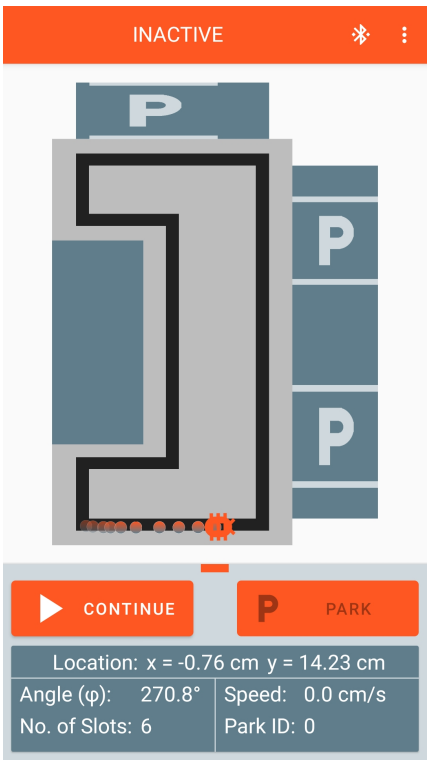

Abbildung A.4: Roboter pausiert mit erfassten Parklücken

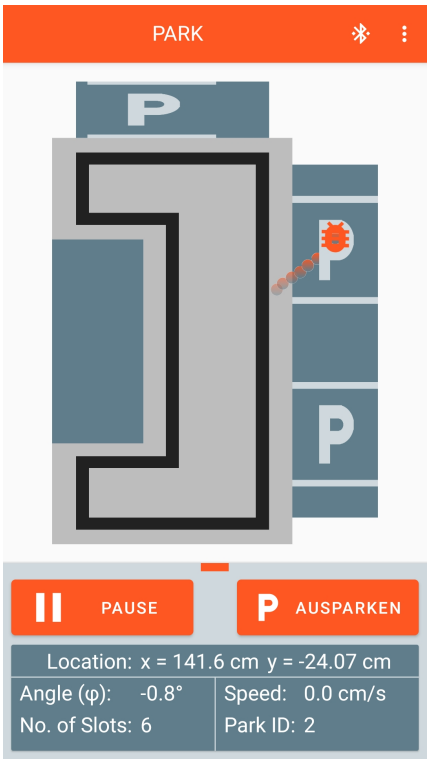

Abbildung A.5: Roboter in Parklücke geparkt

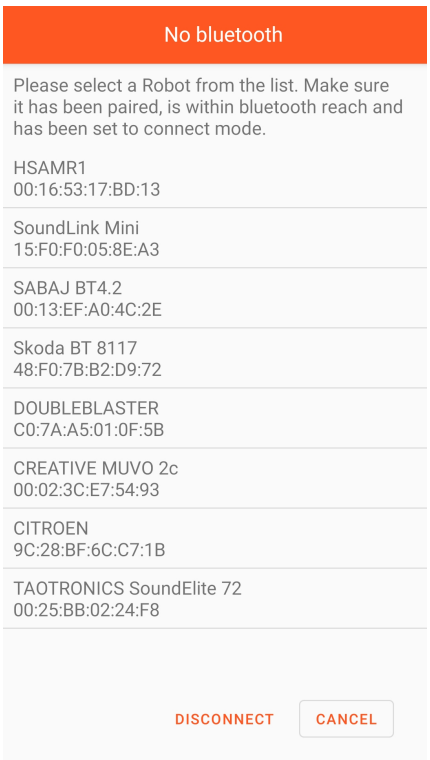

Abbildung A.6: Bluetoothfenster nach hergestellter Verbindung

Abbildung A.7: Parklückenbeispiel in Standardfarbe

Abbildung A.8: UML Diagramm

Quellenverzeichnis

[1] Androidversionen Nutzungsstatistik
https://developer.android.com/about/dashboards;
Zugriff: 10.01.2020

[2] Material Design Veröffentlichung
https://de.wikipedia.org/wiki/Material_Design;
Zugriff: 10.01.2020

[3] Ziel von Material Design
https://material.io/design/introduction/#principles;
Zugriff: 10.01.2020

[4] Design Principles des Material Designs
https://material.io/design;
Zugriff: 10.01.2020

[5] Website zur erstellung einer Farbpalette
https://www.materialpalette.com;
Zugriff: 10.01.2020

[6] Sliding Panel preset
https://github.com/umano/AndroidSlidingUpPanel
Zugriff: 10.1.2020

[7] Legoroboter
https://s3.amazonaws.com/fllcasts/content_pictures/pictures/000/001/685/7e6a60b2259227845f8580736ce79ef466097081LEGO-Mindstorms-EV3-Sterling-Lego-Submarine-Robot-Fllcasts.png?1524911539
Zugriff: 10.1.2020

[8] PixelArt Meerschwein
https://i.pinimg.com/474x/43/db/7b/43db7bf7634f161d98fb4147e2aad671--guinea-pigs-bead-art.jpg
Zugriff: 10.1.2020