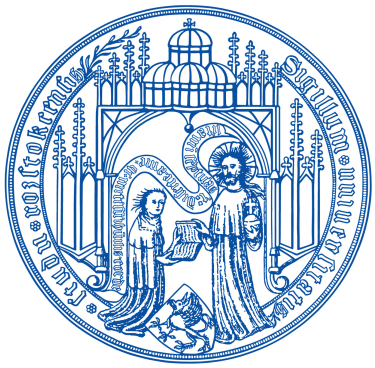
Entwicklung einer autonom fahrenden Sensorplattform auf Basis von Lego EV3 und Microsoft Kinect

Bachelorarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik



vorgelegt von: Ruven Kronenberg
Matrikelnummer: 216203798
geboren am: 1997-11-27 in Hamburg
Erstgutachter: Prof. Dr. rer. nat. habil. Andreas Heuer
Zweitgutachter: Dr.-Ing. Thomas Mundt
Betreuer: M.Sc. Hannes Grunert
Abgabedatum: 2020-03-20

Inhaltsverzeichnis

Kapitel 1

Einleitung

In dieser Bachelorarbeit wird die Entwicklung einer autonom fahrenden Sensorplattform beschrieben. Dabei wird untersucht, welche Sensorik verfügbar ist und inwiefern mit dieser Messungen vorgenommen werden können, um das autonome Fahren zu ermöglichen. Es wird ebenfalls untersucht, wie die erfassten und berechneten Daten sinnvoll gemeinsam gespeichert werden können. Zudem wird sich mit der Problematik der Routenfindung und Navigation beschäftigt. Gearbeitet wird dabei mit einem Lego EV3 als Kernelement.

1.1 Motivation

Der Traum des vollständig autonom fahrenden Automobils ist neben der Elektromobilität die größte und technisch anspruchsvollste Herausforderung der aktuellen Automobilbranche. Doch auch Technologiekonzerne wie beispielsweise Google oder Apple betrachten die Herausforderung als Sprungbrett in das Marktfeld des Autos und können insbesondere mit technischer Expertise und nahezu unbegrenzten finanziellen Mitteln glänzen. Noch hat es jedoch kein Konzern vollbracht einen technisch ausgereiften und marktfähigen Autopiloten zu entwickeln, der allen Aufgaben des vollständigen Straßenverkehrs gewachsen ist. Doch die Reihe an Vorteilen des autonomen Fahrens wirken lukrativ: die Sicherheit im Straßenverkehr könnte optimiert, das Fahrerlebnis verbessert, der Komfort erhöht und im Optimalfall irgendwann Arbeitsplätze eingespart werden.
Unabhängig vom Ausmaß der (Fahr-)Aufgaben, die von einem System automatisch durchgeführt werden können sollen, haben alle Modelle eines gemeinsam: eine große Menge an Sensorik, um die Umgebung und den Zustand des Fahrzeugs zu erfassen und angemessen auf diese reagieren zu können. Dabei geht es um Abstandssensoren, Kameras, Radar, Lidar (ähnlich dem Radar nur mit Laserstrahlen an Stelle von Radiowellen) sowie Beschleunigungs- und Verbrauchssensoren. Konstant wird eine erhebliche Menge an Daten über das Fahrzeug und dessen Umgebung erhoben. Der

Hersteller Tesla geht sogar so weit, in die aktuellen Modelle bereits ausreichend Sensorik einzubauen um damit nach eigener Aussage irgendwann in Zukunft deutlich weiter entwickeltes autonomes Fahren zu ermöglichen, wofür lediglich noch Software nachgerüstet werden müsste. Die erhobenen Daten werden dann teilweise direkt verarbeitet und für den aktuellen Fahrbetrieb ausgewertet, teilweise aber auch gespeichert oder gar an Cloudsysteme weitergegeben.
Der gewöhnliche Autofahrer setzt sich also nunmehr nicht mehr nur hinter das Steuer eines Fahrzeuges, dass ihn von A nach B befördern kann, sondern zusätzlich eine hochtechnologische Datensammelmaschine darstellt und als solche im Zweifelsfall missbraucht werden könnte. Dabei geht es zum Teil um hochsensible Daten, die zweifelsohne für den automatisierten Betrieb des Fahrzeugs unabdingbar, jedoch auch an falscher Stelle gefährlich sein könnten. Hierbei handelt es sich zum Beispiel um Standortdaten, Kamerabilder oder Audioaufnahmen. Diese Daten erfasst das Fahrzeug nicht nur über den Fahrzeugführer und seine Mitfahrer, sondern eben auch über Dritte in der Umgebung des Fahrzeuges und verarbeitet diese dann oder speichert oder gibt diese im Zweifelsfall sogar weiter.
Die Fülle an Daten wäre sicherlich in vielerlei Hinsicht auch für verschiedenste profitable Zwecke von Unternehmen von Interesse. Doch im Sinne des Datenschutzes ist auch zu hinterfragen, inwiefern es zu rechtfertigen ist, dass große Datenmengen abseits der direkten Wahrnehmung der Fahrzeuginsassen erhoben, verarbeitet und gespeichert werden. Es ist zu bezweifeln, dass das für den Nutzer nachvollziehbar oder transparent geschieht.
Dieser Tatsache entgegenwirkend soll in dieser Arbeit evaluiert werden, in wie weit es möglich ist, bereits mit einer minimalen Teilmenge an Sensordaten ein autonom fahrendes Fahrzeug zu entwickeln. Hierbei sollen alle Sensordaten ausschließlich lokal auf dem Gerät verarbeitet und ggf. gespeichert werden. Eine Herausforderung wird dabei insbesondere die Bereinigung und Fusion der Sensordaten darstellen. Im Optimalfall soll am Ende eine Abschätzung möglich sein, in wie weit die Zuhilfenahme weiterer ggf. sensibler Sensorik die Qualität des autonomen Fahrens verbessert. Als Ausgangspunkt dient hierfür ein LEGO Mindstorms EV3, da dieser mit einigen rudimentären Sensoren ausgestattet bzw. kompatibel ist.
Diese Arbeit beschäftigt sich explizit nicht mit der Entwicklung oder Optimierung autonomer Fahrsysteme als solcher. Es wird weder versucht den aktuellen Stand der Technik vollumfänglich auf ein kleineres Modell zu übertragen, noch versucht diesen in irgendeiner Weise zu verbessern. Zudem ergibt sich aus der Beschaffenheit der verwendeten Technik ein die Verwendung aber auch Nicht-Verwendung einiger Technologien.

1.2 Problemstellung

Für aktuelle Modelle des autonomen Fahrens werden große Rechnerkapazitäten und diverse Sensordaten insbesondere Kamerabilder erhoben. In dieser Arbeit wird eine Lösung entwickelt, wie autonomes Fahren mit weniger sensiblen Sensordaten, kleinerer Rechnerkapazität und ohne eine Verbindung zu Serversystemen im Rahmen eines Kleinstfahrzeuges möglich ist. Die vorliegende Arbeit bietet eine Lösung für die obige Problemstellung, indem sich mit dem Lego EV3 und der dafür verfügbaren Sensorik und Komponenten beschäftigt wird und diese hinsichtlich der Sensibilität und Nutzbarkeit für autonomes Fahren untersucht werden. Es werden Lösungsstrategien aufgezeigt, wie durch die Nutzung der untersuchten und als ausreichend befundenen Sensorik und möglichst gering aufwendigen Prozessen hinsichtlich der Rechnerkapazität ein Fahrzeug, dass sich autonom in einer nicht vollständig bekannten Umgebung zu einem gegebenen Ziel bewegen kann, erreicht werden kann. Die folgenden Zielsetzungen werden dafür definiert:

- Auswahl der nötigen Sensorik aus der gegebenen zu untersuchenden Menge und Abwägung auf welche verzichtet werden kann.
- Erarbeitung eines Konzeptes für die Umgebungsverarbeitung, welches die Umgebung in der sich das Fahrzeug befindet darstellen und zur Routenfindung nutzbar machen kann.
- Spezifikation und Erkennung von Grundlegenden Fahrsituationen, mit denen umgegangen werden muss als Ausgangsbasis für die Auswahl der nötigen Sensorik, den Aufbau des Fahrzeuges und die zu verwendende Software.
- Erarbeitung einer Strategie, bei der, unter Verwendung der ausgewählten Sensorik, ein möglichst reibungsloser Ablauf des autonomen Fahrzeuges auf einer optimalen Route zum Ziel gewährleistet werden kann.

Das Hauptziel ist ein integriertes Gesamtkonzept für eine autonom fahrende Sensorplattform mit dem Lego EV3 als Kernelement und einem Minimalansatz hinsichtlich zu verwendender Sensorik.

1.3 Abgrenzung

Ausgehend von den Zielsetzungen wurden die Schwerpunkte der Arbeit abgeleitet. So wird sich mit der Grundlegenden Umsetzung des autonomen Fahrens beschäftigt. Sie umfassen jedoch viele weitere Aspekte, die im Rahmen dieser Arbeit nicht abgedeckt werden. Es wird sich nicht mit der Interaktion mit anderen Fahrzeugen beschäftigt. Ebenfalls werden einige komplexere Fahraufgaben im realen Straßenverkehr in dieser Arbeit außen vor gelassen. Hierbei geht es vor allem um Aufgaben, wie Einparken,

Spurhaltung, Spurwechselmanöver, Überholmanöver und Erfassung beziehungsweise Anwendung der vorhandenen Rahmenbedingungen durch Verkehrsregeln, Verkehrszeichen und Signalanlagen.

1.4 Gliederung der Arbeit

In Kapitel 2 dieser Arbeit ist zunächst der aktuelle technische Stand des autonomen Fahrens mit dem Fokus auf die Sensorik am Beispiel eines aktuellen Automodells des Herstellers Tesla beschrieben. Kapitel 3 erklärt das erdachte Konzept bezüglich des Aufbaus und des Fahrzeuges und dessen Verhalten in verschiedenen (Fahr-)Situationen. Im vierten Kapitel wird die programmatische Umsetzung des Konzepts vorgestellt und auf die Inbetriebnahme des Betriebssystems für den EV3 eingegangen. Kapitel fünf beschäftigt sich mit der Evaluation der Testfahrten mit dem entwickeltem Fahrzeug und der Auswertung der Tests zur Wi-Fi-basierten Ortung. Das sechste Kapitel gibt einen Überblick über mögliche Fortsetzungen bzw. Projekte auf Grundlage dieser Arbeit.

Kapitel 2

Stand der Technik

Dieses Kapitel bietet einen Überblick über die aktuelle technische Situation des autonomen Fahrens (siehe Abschnitt 2.1). Des Weiteren wird in den Abschnitten 2.2 und 2.3 die verwendete Sensorik und die verfügbare Technik untersucht, um einen eigenen minimalen Ansatz abzubilden.

2.1 Autonomes Fahren

Im Folgenden wird der aktuelle Stand des autonomen Fahrens als solches untersucht. Insbesondere wird dabei auf die Fahrsensorik am Beispiel eines aktuellen Fahrzeuges eingegangen.

2.1.1 Allgemeines

Das autonome Fahren stellt eine aktuelle Herausforderung für die Automobilbranche dar. Die erhofften positiven Effekte reichen von Unfallvermeidung über Komfort bis hin zur Einsparung von Arbeitskräften im Personen- und Lastkraftverkehr.
Die heute allgemein gängige Einstufung (teil-)autonom fahrender Systeme wurde durch das SAE Institute 2014 vorgenommen. In [SAE18] werden Systeme in sechs Level, beginnend mit keiner und endend mit voller Automatisierung, unterteilt. Diese Level sind verkürzt wie folgt beschrieben:

0. Keinerlei Automatisierung
1. Assistenzsysteme bei entweder Längs- oder Querführung; z.B. Tempomat
2. Teilautomatisierung für Längs- und Querführung; der Fahrer hat auf Umgebung zu achten
3. Autonomes Fahren unter Eingriffsvorbehalt; das System übernimmt Umgebungserfassung

4. Hochautomatisierung, kein Eingriffsvorbehalt mehr; notfalls beendet das System die Fahrt an sicherer Stelle

5. Vollautomatisierung; das Fahrzeug kann jegliche auch durch den Menschen zu bewältigende Fahraufgabe ausführen

Dabei kommt manchmal zu dem Missverständnis[1][2][3], dass Level 5 bedeuten würde, dass ein Mensch nicht mehr selbst die Kontrolle über das Fahrzeug übernehmen kann bzw. keine Steuermöglichkeit gegeben sein dürfte. Der Standard legt zwar nahe, dass jedes Fahrzeug, bei dem ein Mensch nicht mehr eingreifen kann, in Level 4 oder 5 einzuordnen ist, jedoch wird für Level 5 nicht explizit ausgeschlossen, dass ein Mensch die Kontrolle übernehmen kann. Es ist lediglich definiert, dass das System zu nahezu keinem Zeitpunkt mehr die Notwendigkeit hat, die Kontrolle an einen Menschen abzugeben, da dieses ebenso jede, auch durch einen Menschen zu bewältigende, Fahrsituation bewältigen kann.

Aktuelle autonome Fahrzeuge im Straßenverkehr erreichen maximal Level 2. Dies ist vor allem rechtlichen Rahmenbedingungen geschuldet, da autonomes Fahren des Level 3 oder höher noch nicht in Deutschland für den Straßenverkehr zugelassen ist und es noch keine internationalen Regelungen gibt (Stand August 2019)[4]. Auch die aktuellen Modelle der Firma Tesla bewegen sich lediglich auf Level 2, denn es heißt bei [Tes19]: Die "gegenwärtigen Autopilot-Funktionen verlangen aktive Überwachung durch den Fahrer - ein autonomer Betrieb des Fahrzeugs ist damit nicht möglich". Dies wird insbesondere bei Unfällen stets betont, da der Fahrer die Verantwortung für die Umgebungserkennung hat und das System ihm lediglich assistiert. Der Audi A8 wurde laut [AUD20] als "erstes Serienautomobil der Welt" für das autonome Fahren des Level 3 entwickelt. Dieser hat einen Staupiloten, der auf zäh fließenden Verkehr von bis zu 60 km/h ausgelegt ist.

2.1.2 Aktuelle Sensorik beim autonomen Fahren am Beispiel des Tesla Model S (Hardwareversion 2)

Bei aktuellen Modellen autonom fahrender Automobile wird eine Fülle an Sensoren verwendet. Neben Sensoren für Sicherheit, Verschleiß und allgemeinen Betrieb des Fahrzeuges, soll hier insbesondere auf die Sensoren eingegangen werden, deren Zweck die Umgebungserkennung bzw. "Situationserkennung" des Fahrzeuges ist.

[1] https://www.forbes.com/sites/cognitiveworld/2019/12/08/how-autonomous-vehicles-fit-into-our-ai-enabled-future/ zuletzt aufgerufen am 2020-02-12

[2] https://www.manager-magazin.de/unternehmen/autoindustrie/selbstfahrende-autos-die-5-stufen-des-autonomen-fahrens-erklaert-a-1256773.html zuletzt aufgerufen am 2020-02-12

[3] https://www.iav.com/en/what-moves-us/on-the-way-to-level-5 zuletzt aufgerufen am 2020-02-12

[4] https://www.autonomes-fahren.de/zulassungen-fuer-das-autonome-fahren-auf-level-3/ zuletzt aufgerufem am 2020-02-12

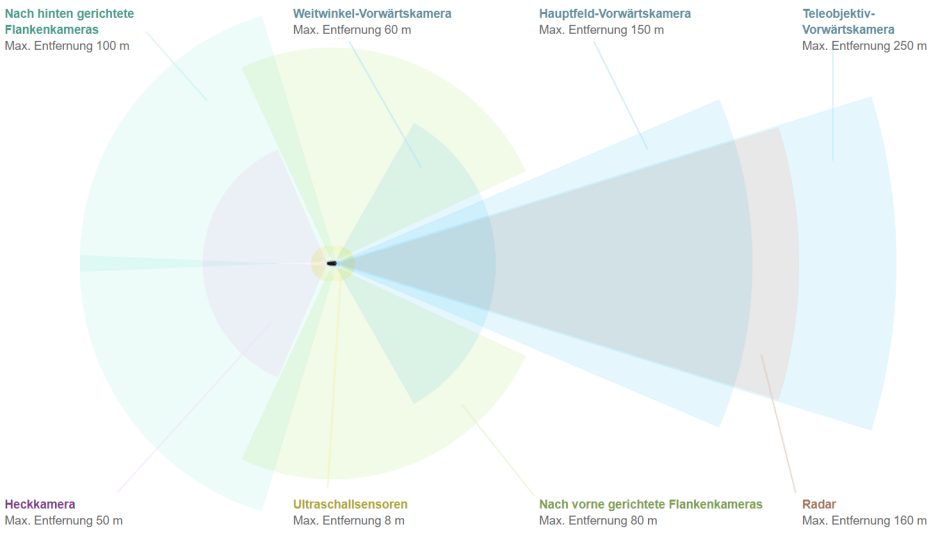


Abbildung 2.1: Sensorenausstattung laut Hersteller
(Quelle: `https://www.tesla.com/autopilot`)

Im Folgenden ist die aktuelle Sensorausstattung des Tesla Model S in der Hardwareversion 2 laut [Tes19] aufgeführt, ein Überblick ist zudem in Abbildung 2.1 zu sehen:

- **Kameras:** Acht Kameras decken insgesamt ein Sichtfeld von vollen 360° ab. Dabei gibt es:
  - Drei nach vorne gerichtete Kameras, die sich alle an der Windschutzscheibe oberhalb des Rückspiegels befinden:
    - Die Teleobjektiv-Vorwärtskamera ermöglicht die Erkennung von Objekten in bis zu 250 m Entfernung. Sie ist insbesondere für das Fahren bei hohen Geschwindigkeiten von Nöten.
    - Eine Hauptfeld-Vorwärtskamera ist für die allgemeine Erkennung des Vorfelds bis zu 150 m angedacht.
    - In einem Sichtfeld von 120° auf bis zu 60 m kann die Weitwinkel-Vorwärtskamera die vordere Umgebung erfassen. Damit eignet sie sich vor allem für das Manövrieren bei niedriger Geschwindigkeit und in Städten.
  - Vier Flankenkameras, die vor allem zur Umgebungserfassung bei eigenem Spurwechsel und dem anderer Fahrzeuge gedacht sind:
    - Zwei nach vorne gerichtete Flankenkameras, je eine links und eine

rechts in der mittleren Türsäule, die in einem Sichtfeld von 90° auf Entfernungen bis zu 80 m arbeiten.

* Zwei nach hinten gerichtete Flankenkameras, je eine links und eine rechts am Vorderkotflügel, die bis zu 100 m entfernte Objekte im toten Winkel erfassen können.

– Eine Heckkamera oberhalb des Kennzeichens erkennt auf bis zu 50 m Entfernungen Objekte hinter dem Fahrzeug. Sie ist vor allem für das Rückwärtsfahren und Einparkmanöver gedacht, wird aber auch zur generellen Umgebungs- bzw. Situationserkennung genutzt.

- **Radar:** Ein nach vorne gerichtetes Radar hinter dem vorderen Stoßfänger erkennt Objekte in bis zu 160 m Entfernung. Aufgrund der Beschaffenheit von Radarstrahlen eignet es sich besonders gut zur Umgebungserkennung bei Regen, Schnee, Nebel oder sonstigen schlechten Sichtbedingungen und kann unter vorausfahrenden Fahrzeugen hindurchschauen.

- **Ultraschallsensoren:** Zwölf Ultraschallsensoren, wovon sich je sechs an der Hinter- und Vorderseite des Fahrzeuges jeweils in den Stoßfängern befinden, ermöglichen eine präzise Rundumerkennung von nahen Objekten in einer Entfernung von bis zu 8 m. Sie werden vor allem für Einparkmanöver und zur Spurwechselerkennung anderer Fahrzeuge verwendet.

2.2 Der LEGO EV3

Der LEGO EV3 ist nach dem RCX und dem NXT der dritte intelligente Stein der Mindstorms-Serie, er verfügt laut Herstellerangaben in [LEG15] über 16 MB Flash-Speicher, 64 MB RAM und einen 300 MHz ARM9-Prozessor. Er ist ausgestattet mit einer Bluetooth-Schnittstelle, einem USB-Port, einem Mini-USB-Port, einem Micro-SD-Kartenleser und je vier RJ12-Eingabe- bzw. -Ausgabe-Anschlüssen. Die Eingabe-Anschlüsse sind dabei im nicht verbundenen Zustand des EV3 für unterschiedliche Sensoren reserviert. Wenn der EV3 mit der Steuer-App verbunden ist oder ein zuvor extern aufgespieltes Programm ausgeführt wird, können die Sensoren jedoch beliebig eingesteckt werden. Dasselbe gilt für die Eingabe-Anschlüsse. Zudem verfügt der EV3 über beleuchtete Tasten, ein Schwarz-Weiß-Display mit einer Auflösung von 178 x 128 Pixeln und einen Lautsprecher.
Zum EV3 gehören des Weiteren zwei große Motoren mit einer Leistung von 160 bis 170 Umdrehungen pro Minute und ein mittlerer Motor mit einer Leistung von 240 bis 250 Umdrehungen pro Minute. Dabei haben beide Versionen je eine Drehgenauigkeit von einem Grad.

2.2.1 Betriebssystem

Die Wahl des Betriebssystems ist für die Nutzung des EV3 maßgebend. Es ist möglich, den EV3 mit dem Standardbetriebssystem, welches er ab Werk mitbringt, zu betreiben, was die Nutzungsmöglichkeiten und insbesondere die Kompatibilität zu anderen Geräten stark einschränkt. Alternativ ist es auch möglich, den EV3 von einer Micro-SD-Karte booten zu lassen. Dabei gibt es eine Reihe an Drittanbieter-Betriebssystemen, die die Einsatzmöglichkeiten des EV3 auf unterschiedliche Art und Weise erweitern. Im Folgenden wird daher auf das native Betriebssystem des EV3 und die beiden Drittanbieter-Betriebssysteme LeJOS und ev3dev eingegangen.

Natives OS

Das native Betriebssystem des EV3 (Brick-OS) basiert laut [LEG13] auf einem Linux-Kernel der Version 2.6.33 RC4. Es ermöglicht die graphische Programmierung des Roboters sowohl am Computer als auch direkt auf dem Gerät oder per App. Die Programmierumgebung LEGO Mindstorms EV3 Home basiert auf LabVIEW und bietet dem Entwickler die Auswahl aus sechs Blockkategorien (Aktion, Ablauf-Regelung, Sensor, Daten-Operation, Erweiterter Modus, Eigene Blöcke) und damit insgesamt 37 verschiedene Blöcke. Ein Block entspricht dabei zum Beispiel einer Schleife, einer arithmetischen Operation, einer Motorbewegung oder dem Programmende. Die Blöcke werden dabei per Drag-and-Drop zurechtgeschoben und können mittels Schaltflächen und Textfeldern weiter eingestellt werden.

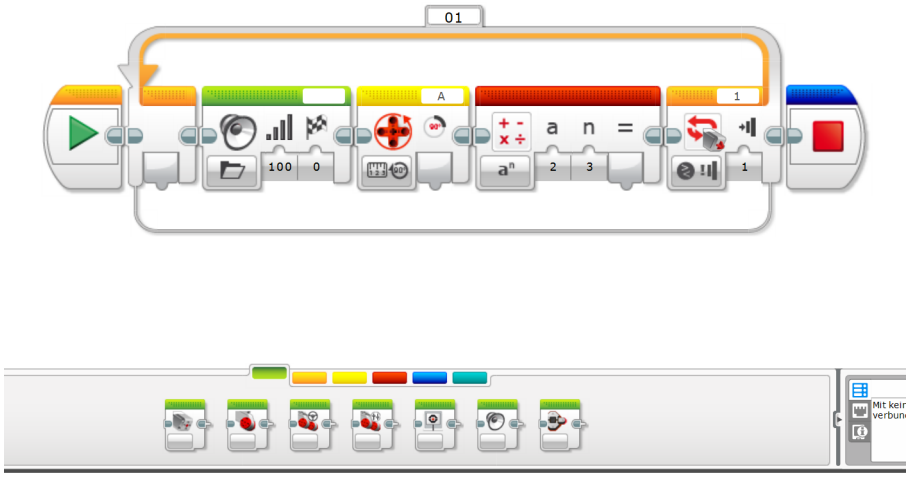


Abbildung 2.2: Screenshot der Entwicklungsumgebung LEGO Mindstorms EV3 Home

In Abbildung 2.2 ist ein Beispielprogrammablauf bestehend aus sechs Blöcken zu erkennen. Links ist der Programmstartblock zu sehen, der rechts an einen Schleifenblock angeschlossen ist, welcher wiederum an den Programmendblock anknüpft. Der Schleifenblock enthält einen Aktionsblock (Klang), einen Sensorblock (Motorumdrehung) und einen Daten-Operationblock (Mathe). Als Schleifenabbruchsbedingung ist eine Zustandsänderung des Berührungssensors angegeben. Das Beispielprogramm dient ausschließlich der Veranschaulichung und verfolgt keinen weiteren Zweck. Unten ist die Auswahlmöglichkeit weiterer Aktions-Blöcke zu sehen. Die sechs farbigen Reiter stehen für die zuvor aufgezählten Blockkategorien.

LeJOS

LeJOS ist ein Java-Betriebssystem für die Bausteine der LEGO-Mindstorms-Serie. 2013 wurde es auf den EV3 portiert und zuletzt im August 2017 aktualisiert. Es ermöglicht die Programmierung des EV3 mit Java, woraus weitere Vorteile entstehen. So wird es möglich weitere WLAN-Dongles außer dem offiziell unterstützten anzuschließen, Multithreading wird ermöglicht und Versionskontrolle vereinfacht. Auch komplexere Programmabläufe als mit der Standardsoftware, wie z.B. rekursive Programmierung, wird so ermöglicht. Zu diesem Zwecke hier in Abbildung 2.3 beispielhaft ein Hello-World-Programm, welches als Beispiel Teil des LeJOS-Downloads ist.

Im Test ließ sich eine Micro-SD-Karte mit LeJOS bespielen und der EV3 von dieser booten. Leider war es nicht möglich, den EV3 dann, wie auf der LeJOS-Seite `http://www.lejos.org/ev3.php` beschrieben, mit dem Rechner zu verbinden. Sowohl unter Verwendung von Windows 10 als auch der Verwendung von Ubuntu 19.10 konnte das Gerät zwar per Bluetooth gekoppelt, jedoch nicht verbunden werden. Per USB-Kabel angeschlossen wurde das Gerät vom System erkannt, jedoch nicht vom Eclipse-Plugin von LeJOS. Zu keinem Zeitpunkt war es möglich, den EV3 über die angegebene fixe IP 10.0.1.1 anzupingen. Auch ein Neuaufsetzen der Micro-SD-Karte brachte keinen Erfolg.

ev3dev

ev3dev `https://www.ev3dev.org/` ist ein Debian-Linux-basiertes Betriebssystem für den EV3 und andere LEGO Mindstorms kompatible Plattformen. Es wird immer noch regelmäßig gewartet (Stand Dezember 2019) und ist bereits damit im Vorteil gegenüber LeJOS. Eine Reihe von Bibliotheken stehen direkt zur Verfügung, die es erlauben Programmierung, beispielsweise in Python, Java, Go, C, C++ oder gar Prolog vorzunehmen. Dabei ist es sogar möglich, den EV3 über LeJOS zu programmieren. Da ev3dev ein auf Debian Linux basierendes Betriebssystem ist,

```java
package org.lejos.ev3.sample.helloworld;
import lejos.hardware.BrickFinder;
import lejos.hardware.Button;
import lejos.hardware.Sound;
import lejos.hardware.lcd.Font;
import lejos.hardware.lcd.GraphicsLCD;
import lejos.utility.Delay;

public class EV3HelloWorld
{
    public static void main(String[] args)
    {
        //System.out.println("Running...");
        GraphicsLCD g =
        BrickFinder.getDefault().getGraphicsLCD();
        final int SW = g.getWidth();
        final int SH = g.getHeight();
        Button.LEDPattern(4);
        Sound.beepSequenceUp();

        g.setFont(Font.getLargeFont());
        g.drawString("leJOS/EV3", SW/2, SH/2,
        GraphicsLCD.BASELINE|GraphicsLCD.HCENTER);
        Button.LEDPattern(3);
        Delay.msDelay(4000);
        Button.LEDPattern(5);
        g.clear();
        g.refresh();
        Sound.beepSequence();
        Delay.msDelay(500);
        Button.LEDPattern(0);
    }
}
```

Abbildung 2.3: Hello World für den EV3 von LeJOS

```
#!/usr/bin/env python3
from ev3dev2.sensor.lego import TouchSensor
from ev3dev2.led import Leds

ts = TouchSensor()
leds = Leds()

while True:
  if ts.is_pressed:
    leds.set_color("LEFT", "GREEN")
    leds.set_color("RIGHT", "GREEN")
  else:
    leds.set_color("LEFT", "RED")
    leds.set_color("RIGHT", "RED")
```

Abbildung 2.4: Beispielprogramm in Python für ev3dev

ist es außerdem möglich, diverse Treiber zu installieren und somit etliche Geräte anzusprechen.

Somit stellt ev3dev eine erhebliche Erweiterung der Möglichkeiten von LeJOS und umso mehr derer des nativen Betriebssystems des EV3 dar. Insbesondere die Möglichkeit, den EV3 mittels Python zu programmieren, stellt einen komfortablen Mehrwert dar. Dies wird sogar explizit aufgrund der Einfachheit für Anfänger empfohlen. Als Beispiel ist in Abbildung 2.4 ein Python-Programm zu sehen, dass die LEDs des EV3 bei Berührung des Berührungssensors die Farbe zu Grün wechseln lassen und im ungedrückten Zustand rot aufleuchten lassen.

Im Test war es problemlos möglich, eine Micro-SD-Karte mit ev3dev zu bespielen und den EV3 von dieser zu booten. Auch eine Verbindung zu einem Rechner war nun möglich. Im Folgenden wurde entschieden, den EV3 mit Python der ev3dev-lang-python-Bibliothek zu programmieren.

2.3 Verfügbare Sensoren

Verfügbar sind für diese Arbeit in erster Linie mit dem LEGO EV3 kompatible Sensoren und ebenso die des LEGO NXT. Zudem ist die XBOX Kinect verfügbar, bei der hier auch auf beide Versionen vergleichend eingegangen werden soll.

2.3.1 LEGO EV3

- **Gyrosensor:** Laut [LEG19b] ermöglicht der Gyrosensor die Erfassung von Drehbewegungen und Richtungsänderungen. Dabei erfolgt die Winkelmessung mit einer Genauigkeit von $\pm 3°$ mit einer maximalen Ausgabe von 440° pro Sekunde bei einer Erfassungsrate von 1 kHz. Der Sensor kann dabei lediglich die Drehbewegung oder Richtungsänderung in einer Dimension messen. Die Dimension, in der gemessen werden kann, wird durch zwei Pfeile auf der oberen Seite des Sensors angezeigt.
  In [San17] haben die Autoren herausgefunden, dass es auf der Unterseite der Gyrosensoren einen vierstelligen Code gibt, dessen hinterste Zahl ein Zähler des Produktionsjahr sein soll. Dies ist relevant, da sich Sensoren mit den Zählern 2 und 3 bezüglich Neukalibrierung anders verhalten als Sensoren mit den Zählern 4 bis 8. Um bei den neueren Sensoren eine korrekte Neukalibrierung zu erwirken, ist es von Nöten, die Kommunikation zum Sensor kurz zu unterbrechen und 3 bis 4 Sekunden zu warten.

- **Ultraschallsensor:** Von [LEG19d] wird der Ultraschallsensor als Sensor zur Erfassung von Entfernungen in Zentimetern oder Zoll beschrieben. Die Messung soll dabei für Entfernungen von 3 cm bis zu 250 cm mit einer Messgenauigkeit von ± 1 cm möglich sein. Die Entfernung wird gemessen, indem Ultraschallwellen frontal ausgesandt und deren Zeitpunkt bis zur Wiederkehr registriert wird. Mit dem Ultraschallsensor ist es also möglich, die Entfernung zu potentiellen Hindernissen zu messen bzw. sie überhaupt erst als solche wahrzunehmen, bevor man auf diese stößt.

- **Berührungssensor:** Der Berührungssensor wird von [LEG19a] als "einfacher, aber außergewöhlich präziser Sensor" beschrieben. Dabei wird bereits auf die Möglichkeit hingewiesen, "Start/Stopp-Steuersysteme" mithilfe des Sensors zu konstruieren. Der Sensor hat einen Druckknopf an der Frontseite der Berührungen registriert. Daher eignet er sich hervorragend, um unvorhergesehene Stöße, also das Anstoßen an ein zuvor nicht durch andere Sensoren erkanntes Hindernis, zu erkennen und die Bewegung daraufhin zu stoppen.

- **Infrarot-Detektor bzw. -Sender:** Beim LEGO EV3 gibt es nach Angabe von [LEG19c] separat Infrarotsender und -empfänger zu kaufen, um mit diesen Abstandsmessungen zwischen 50 cm bis 70 cm vorzunehmen. Hierbei ist es allerding erforderlich, dass ein Infrarotsender separat am Ort, zu dem die Entfernung zu messen ist, angebracht ist.

- **WLAN-Dongle:** Nach Angabe von [LEG19f] wird ausschließlich der Netgear N150 Wireless Adapter (WNA1100) offiziell unterstützt, um den EV3 per WLAN verbinden zu können. Dieser erfüllt nach [net08] den IEEE 802.11 b/g/n

(2,4 GHz) Standard, verfügt über einen USB 2.0 Anschluss und unterstützt Datenraten von bis zu 150 Mbps.
Dank der Nutzung von ev3dev ist es auch möglich, andere WLAN-Dongles zu verwenden. Hier soll der BENQ USB Wireless Dongle WDRT8192 genutzt werden, der nach [ben19] denselben USB- und Drahtlosstandard ebenfalls im 2,4 GHz Frequenzband erfüllt und dabei Datenraten bis zu 300 Mbps erreichen kann. Dieser hat neben seiner höheren möglichen Datenrate auch den Vorteil, geringfügig kleiner zu sein.

2.3.2 LEGO NXT

In der Produktserie LEGO Mindstorms gibt es einen vorangegangenen Steuerungscomputer sowie für diesen entwickelte Sensoren und Motoren. Glücklicherweise gibt [LEG19e] an, dass die Komponenten des alten LEGO NXT vollständig mit dem LEGO EV3 kompatibel sind. Es wird angegeben, "[...] dass alle bestehenden Sensoren, Motoren und Bauelemente mit der neuen Plattform funktionieren." Somit werden hier auch alle Sensoren des LEGO NXT in Betracht gezogen und mit ggf. ähnlichen des LEGO EV3 verglichen.

- **Infrarot-Sensor:** Der Infrarotsensor aus dem Hause Mindsensors ermöglicht laut [genb] eine Abstandsmessung zwischen 10 cm und 80 cm. Hierfür wird ein SHARP GP2Y0A21YK Sensor verwendet. Hierbei handelt es sich nicht um eine Originalkomponente von LEGO.

- **Ultraschallsensor:** Ebenso wie den eben genannten Infrarotsensor, vertreibt [genc] theoretisch (sofern Lagerbestände vorhanden wären) auch den Ultraschallsensor für den LEGO NXT. Mit diesem sind laut Angabe Entfernungsmessungen zwischen 0 cm und 255 cm möglich.

- **Berührungssensor:** Für den LEGO NXT existiert, wie bei [www16] nachzulesen ist, ebenfalls ein Berührungssensor, der im Wesentlichen mit dem des LEGO EV3 gleichzusetzen ist.

- **Gyroskopsensor:** Im Angebot von [gena] befindet sich ebenfalls eine Gyroskopsensor für den LEGO NXT, hierbei handelt es sich um ein zertifiziertes Produkt von HiTechnic, also ebenfalls keine Originalkomponente von LEGO.

2.3.3 Xbox Kinect

In [Kep13] werden technische Details über die erste Version der Xbox Kinect aus dem Jahre 2010 beschrieben. So wird erwähnt, dass diese sowohl eine RGB-Kamera als auch einen Tiefensensor, wodurch die dreidimensionale Erkennung der Umgebung möglich ist, hat. Der Tiefensensor besteht aus einer Infrarotkamera und einem

	Version 1	Version 2
Auflösung Farbkamera (Pixel)	640 x 480	1920 x 1080
Auflösung Tiefenkamera (Pixel)	640 x 480	512 x 424
Bildrate Farbkamera (fps)	30	30
Bildrate Tiefenkamera (fps)	30	30
Sichtfeld horizontal	57°	70°
Sichtfeld vertikal	43°	60°
Messinterval (cm)	60 - 500	50 - 450
Bittiefe Farbbild (Bit)	8	-
Bittiefe Tiefenbild (Bit)	11	-
Neigungsmotor	Ja	Nein

Tabelle 2.1: Vergleich technische Spezifikationen Xbox Kinect Version 1 und 2

laserbasierten Infrarotemitter. Da mit eigens emittiertem Licht im Infrarotbereich gearbeitet wird, ist die Umgebungserkennung nicht von externen Lichtquellen abhängig und somit auch in völliger Dunkelheit möglich.
Der Kinect ist es möglich, Entfernungen zwischen 60 cm und 500 cm in einem Sichtfeld von 57° horizontal und 43° vertikal zu erkennen. Dabei arbeitet sie mit einer kontinuierlichen Bildrate von 30 Bildern pro Sekunde. Die Auflösung beträgt für das Tiefen- und Farbbild gleichermaßen 640 x 480 Pixel. Das Farbbild hat dabei eine Tiefe von 8 Bit und das Tiefenbild eine Tiefe von 11 Bit.
Die Tiefenerkennung erfolgt indem der Infrarotemitter ein Punktenetz aussendet und die Infrarotkamera dieses erkennt und mittels Triangulation zwischen den Punkten den Unterschied zwischen emittierten und erkannten Punktenetz ermittelt.
Bei [Jia17] ist die zweite Version der Xbox Kinect beschrieben. Ein Vergleich der Eckdaten beider Versionen ist in Tabelle 2.1 vorgenommen. Die zweite Version wurde im Juli 2014 veröffentlicht und kann im Vergleich zur ersten Version mit einem Farbbild in 1080p Auflösung und einer Tiefenkamera, die in 512 x 424 Pixeln aufzeichnet, punkten. Dabei hat sie ein Sichtfeld von 70° horizontal und 60° vertikal. Messungen können zwischen einem Abstand von 50 cm und 450 cm sinnvoll vorgenommen werden.

2.4 Fazit

Die vorliegenden bzw. verfügbaren Sensoren sollten sich gut für die Umsetzung der Fragestellung eignen. Der EV3 ist ausreichend gut ausgestattet um kleinere Fahraufgaben zu bewältigen und durch die Möglichkeit des Einsatzes verschiedener Betriebssysteme, sollte es auch kein Problem sein komplexere Programmstrukturen zu

verwenden oder ggf. "Nicht-Lego-Sensoren" einzusetzen. Aus den untersuchten Sensoren muss nun eine ausreichend minimale Teilmenge ausgewählt werden, um mit dieser Fahrsituationen bewältigen zu können.

Kapitel 3

Konzept

In diesem Kapitel wird der grundlegend angedachte Aufbau des Fahrzeuges in Abschnitt 3.1 erläutert. In Abschnitt 3.2 wird auf die Erkennung verschiedener Situationen, die beim Fahren auftauchen können und mögliche Reaktionen darauf benannt. Auf die Routenfindung wird in Abschnitt 3.3 eingegangen. In Abschnitt 3.4 wird ein Konzept zur Ortung des Fahrzeuges erklärt und in Abschnitt 3.5 Ideen zur Speicherung von zu erhebenden Daten untersucht.

3.1 Aufbau des EV3

Aus den zuvor untersuchten Sensoren und Motoren wird sich zunächst entschieden, folgende zu verwenden:

- zwei große Motoren des EV3,
- einen mittleren Motor des EV3,
- den Berührungssensor des EV3,
- Ultraschallsensor des EV3,
- Gyroskopsensor des EV3
- und einen WLAN Dongle.

Dabei war es nicht von Nöten, die Sensoren des LEGO NXT mit einzubeziehen. Die aufgelisteten Komponenten sollten einen minimalen autonomen Fahrzeugbetrieb gewährleisten zu können. Es wurde sich zunächst explizit gegen die Verwendung der XBOX Kinect entschieden, da diese mit ihren hochauflösenden Kamera- und Tiefenbildern erheblich sensiblere Daten erhebt und somit tendenziell mehr Missbrauchspotential bietet.

Angedacht ist zunächst die Verwendung der zwei großen Motoren, wobei einer dem Antrieb und einer der Lenkung dienen soll. Der dritte Motor soll ebenfalls genutzt werden und in Verbindung mit dem Ultraschallsensor des EV3 als Hauptabstandssensor mit drehbarer Erfassungsrichtung fungieren. Der Berührungssensor soll als Bumper verwendet werden und im Optimalfall Stöße des Fahrzeuges an Front- oder Rückseite registrieren können. Der Gyroskopsensor soll ebenfalls angeschlossen werden und als Umfall-, Steigungs- und Unebenheitserkennung dienen. Zuletzt soll noch der WLAN-Dongle verwendet werden, um mittels Messungen der verfügbaren Access Points und deren Signalstärke eine rudimentäre Ortung zu ermöglichen.

Der Aufbau des Fahrzeuges erforderte drei Räder. Davon sind zwei mit jeweils einem der beiden großen Motoren über zwei Zahnräder verbunden um die Lenkung zu ermöglichen und dieselbe Achshöhe wie die des dritten Rades zu gewährleisten. Diese befinden sich hinten am Fahrzeug in einem Abstand von 11,6 cm voneinander, gut zu erkennen auf der Aufnahme der Rückseite des Fahrzeuges in Abbildung A.3. Das dritte Rad befindet sich vorne am Fahrzeug und dient lediglich als Stützrad. Die Räder haben dabei jeweils eine Breite von 2,2 cm und einen Durchmesser von 4,3 cm. Der Abstand zwischen den Achsen der hinteren Räder und des Stützrades beträgt 12 cm. Die vordere Breite am Fahrzeug ist bestimmt durch den Bumper bei 16,4 cm, die hintere Breite beträgt lediglich 16 cm, was gut auf der Aufnahme der Oberseite des Fahrzeuges in Abbildung A.4 zu erkennen ist. In der Länge misst es 26 cm von der vorderen Kante des Bumpers bis zu den hinteren Rädern, misst man die etwas lose herausragenden Kabel der Sensoren mit, so beträgt die Gesamtlänge 29 cm. An der höchsten Stelle, dem Ultraschallsensor, misst das Fahrzeug 20,3 cm. Vom vorderen Ende des Ultraschallsensors bis zum vorderen Ende des Bumpers existiert ein horizontaler Abstand von 5 cm und das vordere Ende des Ultraschallsensors ist exakt 3,5 cm von seiner Drehachse entfernt, dies ist gut auf der Aufnahme der rechten Seite des Fahrzeuges in Abbildung A.3 zu erkennen.

3.2 Situationserkennung

Die Situationserkennung soll Situationen aus den Werten verschiedener Sensoren ableiten, so dass entsprechend reagiert werden kann. Bei den hier angeführten Werten handelt es sich zunächst um Richtwerte, die sich beispielsweise aus den Sensorengenauigkeiten ergeben. Diese müssen ggf. im späteren Verlauf dynamisch eingestellt oder nach Testfahrten statisch angepasst werden, um die Qualität des autonomen Fahrens zu erhöhen. Die im Folgenden aufgezählten Situationen sollen differenziert von einander erkannt und darauf reagiert werden können. Diese sind in Abbildung 3.1 als Programmablaufplan zusammengefasst.

- **Kein Hindernis:** Gibt der Berührungssensor keine Berührung an und ist für den Ultraschallsensor in Fahrtrichtung keine Entfernung messbar oder lediglich

eine Entfernung über 250 cm messbar, so soll das Fahrzeug in voller Fahrt sein bzw. auf diese beschleunigen.

- **Fernes Hindernis:** Gibt der Berührungssensor keine Berührung an und ist für den Ultraschallsensor in Fahrtrichtung eine Entfernung über 100 cm jedoch maximal 250 cm messbar, so soll das Fahrzeug die Fahrt verlangsamen und Alternativen abwägen, also überprüfen, ob bei einer Kurskorrektur größere Abstände messbar wären.

- **Nahes Hindernis:** Gibt der Berührungssensor keine Berührung an und ist für den Ultraschallsensor in Fahrtrichtung eine Entfernung unter 100 cm jedoch maximal 4 cm messbar, so soll das Fahrzeug die Fahrt stark verlangsamen und eine Kurskorrektur vornehmen, so dass größere Abstände messbar sind. Ebenfalls muss die geplante Route, anhand der vorgenommenen Abweichung von der ursprünglich geplanten Route, angepasst werden, so dass dennoch das korrekte Ziel erreicht werden kann.

- **Übersehenes Hindernis:** Gibt der Berührungssensor eine Berührung an oder ist für den Ultraschallsensor in Fahrtrichtung eine Entfernung unter 4 cm messbar, so soll das Fahrzeug die Fahrt sofort stoppen, ein Stück rückwärtsfahren, eine Kurskorrektur vornehmen und sich die Position des nicht erkannten Hindernis nach Möglichkeit merken, um dieses in Zukunft umfahren zu können. Auch hier muss die geplante Route, anhand der vorgenommenen Abweichung von der ursprünglich geplanten Route, angepasst werden, so dass dennoch das korrekte Ziel erreicht werden kann.

- **Steigung:** Zusätzlich zu Hinderniserkennung soll das Fahrzeug auf Winkeländerungen reagieren können. Bei Winkeländerungen in Fahrtrichtung von 5° soll die Fahrt verlangsamt werden, wenn die Winkeländerung abschüssig ist oder erhöht werden, wenn die Winkeländerung ansteigend ist.

- **Unebenheiten:** Bei Winkeländerungen von 5° in horizontaler Richtung quer zur Fahrrichtung soll die Fahrt verlangsamt werden, um ein Umkippen zu vermeiden.

- **Steigung als Hindernis:** Das Fahrzeug muss ebenfalls auf zu starke Steigungen reagieren können. Würde es beispielsweise eine Rampe mit zunehmender Steigung befahren, unabhängig ob die in welcher Richtung diese Steigung zunimmt, so müsste zu einem Punkt entschieden werden, die Fahrt nicht so fortzusetzen. Da allerdings eine Steigung nicht vom Abstandssensor als Hindernis erkannt werden würde, muss das Fahrzeug soweit zurücksetzen, dass wieder eine ungefährliche Fahrt gewährleistet werden kann und dann eine Kurskorrektur vorgenommen werden. Als Richtwert für eine ungefährliche Fahrt werden

zunächst maximal 20° angenommen. Die geplante Route, anhand der vorgenommenen Abweichung von der ursprünglich geplanten Route, angepasst werden, so dass dennoch das korrekte Ziel erreicht werden kann und zudem nach Möglichkeit das, vom Abstandssensor nicht zu erkennende Hindernis, als solches gespeichert bzw. verzeichnet werden, um es bei zukünftigen Fahrten im Voraus umfahren zu können.

- **Unfall:** Bei einer Winkeländerung von 45° oder mehr zum Ausgangswinkel soll die Fahrt gestoppt werden. Es wird sich vermutlich um einen Unfall (Umkippen) handeln oder dies im letzten Moment verhindert werden können.
- **Ziel (fast) erreicht:** Ist die errechnete Fahrstrecke abgefahren oder ergibt die Ortung, dass sich das Fahrzeug am Ziel befindet, so soll die Fahrt gestoppt werden. Ist die errechnete Fahrstrecke abgefahren, die Ortung ergibt jedoch, dass sich das Fahrzeug noch nicht ganz am Ziel befindet, so soll eine Korrekturroute berechnet werden und diese abgefahren werden, bis das Ziel tatsächlich erreicht ist.

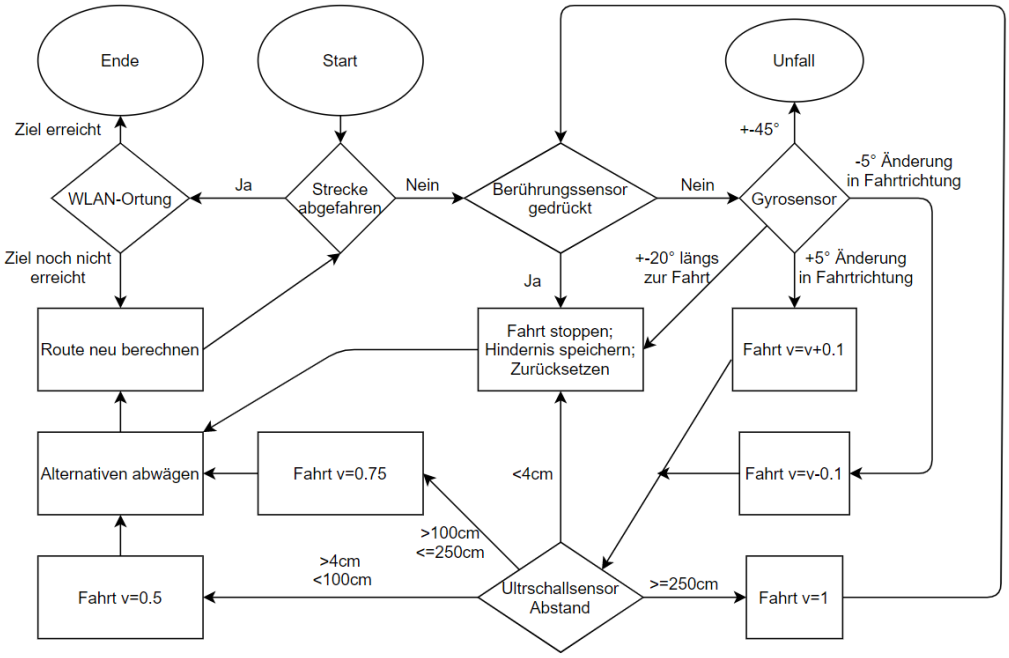


Abbildung 3.1: Programmablaufplan für die Situationserkennung und Reaktion

3.3 Routenfindung

Für die Routenfindung ist zunächst wichtig, wie das Fahrzeug die Umgebung wahrnehmen und verarbeiten soll. Naheliegend ist ein Raster von gleich großen Quadraten. Je kleiner einzelnen Quadrate also Zellen des Rasters sind, desto feiner wäre eine Routenfindung möglich. Da das Fahrzeug mit 29 cm am größten in der Länge ist, wäre das größte sinnvolle Zellausmaß 29 cm x 29 cm. Um jedoch eine feinere Navigation zu ermöglichen ohne den EV3 zu überlasten, wird ein Raster mit einer Zellgröße vo 10 cm x 10 cm verwendet. So nimmt das Fahrzeug in der Theorie, ausgehend davon, dass sich der Mittelpunkt des Fahrzeuges in einer Zellmitte befindet, stets neun Zellen ein.
Möchte man nun zellbasierte Routenfindung vornehmen, so ist es notwendig die Größe des Fahrzeuges, die die einer Zelle übersteigt zu berücksichtigen. Zu diesem Zwecke soll jede Zelle des Rasters stets einen von drei Werten enthalten. Für jeden Zellwert x soll gelten: $x \in \{0, 1, 2\}$. Dabei entspricht 0 einer nicht befahrbaren Zelle also einem Hindernis. Der Zellwert 1 hingegen steht für eine befahrbare Zelle also eine Zelle ohne Hindernis. Die 2 als Zellwert erfüllt dabei die Funktion der Abstandwahrung. Eine jede Zelle, die befahrbar ist und deren acht angrenzenden Zellen ebenfalls befahrbar sind, soll den Wert 2 enthalten. Die Ränder des Rasters werden dabei so behandelt, als würden weitere Zellen mit dem Zellwert 0 angrenzen. Es wird also davon ausgegangen, dass im Voraus die maximale Größe des Einsatzgebietes bekannt ist. Wenn sich das Fahrzeug das erste mal an einem Einsatzort befindet, so würde es mit einer initialisierten Karte arbeiten, die an allen Randzellen den Zellwert 1 und allen inneren Zellwerten den Zellwert 2 aufsetzt. Bei jeder Erfassung einer Zelle mit einem Zellwert, der der Messung widerspricht, müssen sich die Zellwerte der betreffenden Zelle und ggf. der umliegenden Zellen aktualisieren.

In der beispielhaften Karte 3.2 ist ein Einsatzgebiet von 1 m x 1 m angegeben, der zwei Hindernisse beinhaltet. Die Karte besteht daher aus 100 Zellen, die zwei Zellblöcke mit dem Zellwert 0 beinhaltet und ansonsten mit den Zellwerten 1 und 2 gefüllt ist. Das Fahrzeug muss nun lediglich die Zellen zur Routenfindung nutzen, deren Zellwert 2 ist. So können Kollisionen oder Unfälle aufgrund zuvor erkannter oder bereits bekannter Hindernisse vermieden werden. Dazu ist es notwendig, dass das Fahrzeug Karten von bereits bekannten Einsatzgebieten speichern und laden kann und die Umgebung laufend aktualisieren kann um auch veränderlichen Umgebungen gerecht werden zu können.

Für die eigentliche Suche einer optimalen und befahrbaren Route von einem gegebenem Ausgangspunkt zu einem gegebenen Zielort soll die A*-Suche genutzt werden. Die Karte wird also als Graph interpretiert, in dem jede Zelle ein Knoten ist, der Kanten gleichen Gewichts zu allen acht angrenzenden, bei Randzellen fünf

1	1	1	1	1	1	1	1	1	1
1	2	2	1	0	0	1	2	2	1
1	2	2	1	0	0	1	2	2	1
1	2	2	1	0	0	1	2	2	1
1	2	2	1	1	1	1	2	2	1
1	2	2	2	2	2	2	2	2	1
1	2	2	1	1	1	1	2	2	1
1	2	2	1	0	0	1	2	2	1
1	2	2	1	1	0	1	2	2	1
1	1	1	1	1	1	1	1	1	1

Abbildung 3.2: Beispielhafte Karte eines Raumes mit zwei Hindernissen

und bei Eckzellen drei, Zellen hat, sofern der Zellwert der jeweiligen angrenzenden Zelle 2 beträgt. Um einer veränderlichen Umgebung begegnen zu können, muss die A*-Suche nach jeder Aktualisierung der Karte erneut mit dem aktuellen Standort als Ausgangspunkt vorgenommen werden. Für die hier angegebene Beispielkarte und den gegebenen Ausgangspunkt (1, 1) und den gegebenen Zielpunkt (3, 7) ergibt die A*-Suche den Pfad, der in 3.3 rot dargestellt ist. Hierbei ist zu beachten, dass die Punkte als (y, x) dargestellt sind, wobei die oberste Zelle ganz links (0, 0) ist und die X-Achse von links nach rechts und die Y-Achse von oben nach unten ansteigt.

3.4 Ortung

Zu verschiedenen Zwecken ist es vorteilhaft, das Fahrzeug orten zu können. So entfällt beispielsweise die Eingabe der Auswahl eines bekannten Einsatzgebietes, wenn das Fahrzeug selbstständig erkennt, in welchem Einsatzgebiet es sich befindet. Zusätzlich könnte der Ausgangspunkt in einem Einsatzgebiet selbstständig ermittelt werden. Darüber hinaus hätte man die Möglichkeit, zu überprüfen, ob die Route auch wie angedacht abgefahren worden ist. Dreht sich das Fahrzeug mal nicht vollständig oder lässt aufgrund eines unzureichenden Untergrunds die Räder durchdrehen, so kann es vorkommen, dass die rein programmatisch abgefahrene Route von der tatsächlich zurückgelegten Strecke abweicht. Bei normalen Fahrzeugen würde eine Ortung mittels GPS oder Mobilfunkortung von statten gehen. Aufgrund der An-

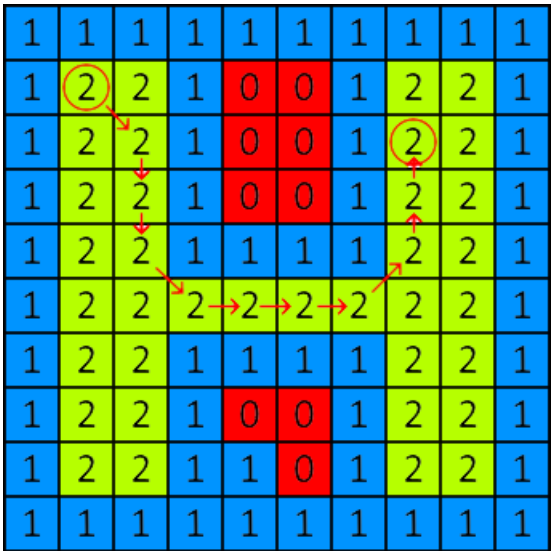


Abbildung 3.3: Beispielhafte Karte eines Raumes mit zwei Hindernissen mit rot visualisiertem optimalen Pfad laut A*-Suche

nahme, dass das Fahrzeug in Gebäuden genutzt wird, fällt die Ortung per GPS aus, auch die Ortung mittels des Mobilfunks gestaltet sich technisch als schwer umsetzbar.

Der EV3 unterstützt allerdings einen WLAN Dongle, mit diesem ist es möglich Verbindung zu Access Points in Reichweite aufzunehmen. Man kann ihn allerdings ebenfalls als Sensor interpretieren, der die verfügbaren Access Points und deren Signalstärken etc. misst. Auf diesem Wege wäre theoretisch eine rudimentäre Ortung auf Grundlage von Wi-Fi möglich. Hier gibt es zwei Varianten, an denen man eine Ortung versuchen könnte. Zum einen wäre die klassische Trilateration möglich. Bei dieser wird die Position auf Grundlage der Entfernung zu drei Punkten deren Standort im Voraus bekannt ist ermittelt. Auf Access Points bezogen könnte man die Signalstärke zur Ermittlung der Entfernung verwenden. Dies funktioniert allerdings nur unter der Annahme, dass die Signalstärke nach einem regelmäßigem Muster mit steigender Distanz abnimmt und radial gleichmäßig ist. Zudem müsste von jedem Router der genaue Standpunkt bekannt sein. Da diese Laborbedingungen nicht für dieses Projekt nachstellbar sind beziehungsweise nicht anzunehmen ist, dass diese hergestellt werden können, ist es nötig sich über die zweite Variante der Ortung mittels Wi-Fi Gedanken zu machen.

So wäre es denkbar jeder Zelle unserer Karte ein spezifisches Wi-Fi Profil zuzuordnen. Man spricht hier auch von "Fingerprinting". Der Grundgedanke ist, dass unabhängig diverser Störfaktoren, die die gleichmäßige Verteilung der Signalstärken von Access

Points unterminieren und damit eine rechnerische Positionsermittlung erschweren, die Signalstärken verschiedener Access Points am selben Ort zumindest relativ einheitlich sein müssten. Selbstverständlich ist man auch hier Störfaktoren unterworfen, jedoch sollten die einzelnen "Profile" in den Zellen über längere Zeit gleich bleiben und somit generell eine Ortung ermöglichen. Mit welcher Genauigkeit dies unter den gegebenen Bedingungen möglich ist, gilt es zu evaluieren.

3.5 Datenspeicherung und -fusion

Für den Betrieb, die Evaluation und Optimierung des Fahrzeuges beziehungsweise dessen Software sind generell zwei Arten von Daten zu erheben und zu speichern. Zum einen sind Kartendaten für die laufende Navigation und einen "Lerneffekt" beim mehrmaligem Durchfahren desselben Einsatzgebietes von Vorteil, zum anderen könne Sensordaten in Verbindung mit der Position, der Ausrichtung, des Status des Fahrzeuges und der aktuellen Zeit in Verbindung gebracht werden.

3.5.1 Kartendaten

Die Kartendaten können einfach als Textdatei abgespeichert werden. Der tatsächliche Inhalt dieser Dateien ist dann abhängig von der softwaretechnischen Darstellung der Karte. Naheliegend ist aber ein zwei-dimensionaler Array mit den jeweiligen Zellwerten. Die Kartendaten könnten mit verschiedenen Namen für unterschiedliche Einsatzorte gespeichert werden, dies würde allerdings die zusätzliche Eingabe des Einsatzortes vorab erfordern.

3.5.2 Sensordaten und weitere Daten

Die Sensordaten sind laufend zu erheben. Zur Speicherung empfiehlt sich aufgrund der Zeitabhängigkeit eine CSV-Datei. In dieser könnten zeilenbasiert, wobei eine Zeile einem Eintrag mit einem Zeitstempel entspricht, Daten gespeichert werden. Zu speichern wären dann jeweils:

- ein Zeitstempel, idealerweise ISO 8601 konform,
- der zuletzt gemessene Abstandswert des Ultraschallsensors in Zentimetern,
- der gemessene Neigungswert des Gyroskopsensors in Grad,
- der Status des Berührungssensors, der Aufschluss darüber gibt, ob der Bumper, und damit das Fahrzeug, ein Hindernis berührt,
- der aktuelle Status des Fahrzeuges wobei für den *Status* gelten soll: $Status \in$ {Ausgangspunkt, Zielort, Neuberechnung der Route, Messung der Umgebung, reguläre Fahrt, keine mögliche Route, Hindernis getroffen},

- die Position des Fahrzeuges im Format (y, x),
- die Ausrichtung des Fahrzeuges ebenfalls im Format (y, x)
- und die Ausrichtung des Ultraschallsensors ebenfalls im Format (y, x).

2020-04-10T03:27:44	106.2	78	0	Ausgangspunkt	(1, 5)	(0, -1)	(0, -1)
2020-04-10T03:27:44	113.10000000000001	78	0	Messung der Umgebung	(1, 5)	(0, -1)	(1, 0)
2020-04-10T03:27:44	120.80000000000001	78	0	Messung der Umgebung	(1, 5)	(0, -1)	(1, -1)
2020-04-10T03:27:45	106.2	78	0	Messung der Umgebung	(1, 5)	(0, -1)	(0, -1)
2020-04-10T03:27:45	255.0	78	0	Messung der Umgebung	(1, 5)	(0, -1)	(-1, -1)
2020-04-10T03:27:45	56.300000000000004	78	0	Messung der Umgebung	(1, 5)	(0, -1)	(-1, 0)
2020-04-10T03:27:52	130.5	85	0	regulaere Fahrt	(2, 6)	(1, 1)	(1, 1)
2020-04-10T03:27:52	103.60000000000001	85	0	Messung der Umgebung	(2, 6)	(1, 1)	(-1, 1)
2020-04-10T03:27:52	89.4	85	0	Messung der Umgebung	(2, 6)	(1, 1)	(0, 1)
2020-04-10T03:27:53	130.0	85	0	Messung der Umgebung	(2, 6)	(1, 1)	(1, 1)

Abbildung 3.4: Beispielhafter Ausschnitt aus einer gespeicherten CSV-Datei einer Testfahrt

Kapitel 4

Implementierung

In diesem Kapitel werden zunächst die Implementierungen der einzelnen EV3-Komponenten und die generellen Möglichkeiten der Ansprache von Sensoren und Motoren in Python für ev3dev erläutert. Dann wird die Implementierung des WLAN-Dongle mittels eines zusätzlichen Python-Pakets beschrieben.

4.1 Implementierung der EV3-Komponente

Der Ultraschallsensor bietet die Möglichkeit Ultraschallwellen auszusenden und zu empfangen. Die Bibliothek beschreibt für den Sensor folgende Möglichkeiten:

- `distance_centimeters_continuous` und `distance_inches_continuous` ermöglichen kontinuierliche Messungen, wahlweise in Zentimetern oder Zoll, so dass zukünftiges Auslesen des Sensors direkt Ergebnisse liefert.
- `distance_centimeters_ping` und `distance_inches_ping` ermöglichen Einmalmessungen, der Sensor ist nach der Ausgabe der Werte wieder im Ruhezustand.
- `distance_centimeters` und `dictance_inches` geben jeweils den zuletzt durch den Sensor gemessenen Abstand zurück.
- `other_sensor_present` gibt einen Wahrheitswert wieder, der indiziert, ob ein anderer Ultraschallsensor in der Umgebung erkannt wurde.

Aufgrund des fehlenden Bezugs des angloamerikanischen Maßsystems zum Dezimalsystem und der Tatsache, dass es kein in sich geschlossenes Maßsystem darstellt, wird im Folgenden, wie auch bereits im vorangegangenen Teil dieser Arbeit, das metrische System verwendet. Daraus resultiert, dass sich aus den eben beschriebenen Funktionen nur diejenigen eignen, welche Messwerte in Zentimetern wiedergeben. Aus den übrigen Funktionen wird hier für die Abstandserkennung

`distance_centimeters_ping` verwendet, da es so möglich ist, nach eigenem ermessen Messungen zu initiieren und auszulesen. Verwendet wird die Funktion dabei wie folgt:

```
1 from ev3dev2.sensor.lego import UltrasonicSensor
2 us = UltrasonicSensor()
3 print(us.distance_centimeters_ping)
```

Hierbei wird zunächst die Klasse des Ultraschallsensors `UltrasonicSensor` aus dem Modul `ev3dev2.sensor.lego` importiert, dann der Ultraschallsensor der Variable `us` zugewiesen und zuletzt die durch den Ultraschallsensor gemessene Distanz als Attribut `distance_centimeters` der Variable `us` ausgegeben.

Der Gyroskopsensor kann den Neigungswinkel bzw. dessen Änderungsrate wiedergeben. Die Bibliothek beschreibt für den Gyroskopsensor folgende Möglichkeiten:

- `angle` gibt den Winkel in Grad zurück, um den sich der Gyroskopsensor seit vom Auszustand aus gedreht hat.
- `rate` gibt die Änderungsrate in Grad pro Sekunden zurück, um die sich der Gyroskopsensor aktuell dreht.
- `angle_and_rate` gibt sowohl die Drehung seit dem Ausgangspunkt als auch die Änderungsrate zurück.
- `reset()` setzt den Gyroskopsensor in den Ausgangszustand zurück.
- `wait_until_angle_changed_by()` lässt warten bis sich der der Gyroskopsensor um einen als ersten Parameter in Grad angegebenen Wert gedreht hat. Dabei wird ein zweiter Parameter als Wahrheitswert erwartet, der angibt ob das Vorzeichen des ersten Parameters zu beachten ist.

Für die Unfall- bzw. Steigungserkennung ist `angle` ausreichend, da aus der Änderungsrate kein Mehrwert für die angestrebte Nutzung resultiert. Zudem bietet `angle` im Gegensatz zu den Funktionen für den Ultraschallsensor immer direkt den aktuellen Wert. Hier wird also nicht zwischen zuletzt gemessenem und aktuellem Wert unterschieden. Dies wird wie folgt eingebunden und genutzt:

```
1 from ev3dev2.sensor.lego import GyroSensor
2 gy = GyroSensor()
3 gy.reset()
4 print(gy.angle)
```

Zunächst wird die Klasse des Gyroskopsensors `GyroSensor` aus dem Modul `ev3dev.sensor.lego` importiert und dann der Gyroskopsensor der Variable `gy`

zugewiesen. Dann muss die `reset()`-Funktion verwendet werden um den Gyroskopsensor in der Ausgangslage auf 0° zurückzusetzen. Danach kann der jeweils aktuelle Winkel in Relation zur Ausgangslage mittels des Attributs `angle` der Variable `gy` ausgegeben werden.

Vom Berührungssensor kann wiedergegeben werden, ob dieser gedrückt ist oder nicht. Die Bibliothek beschreibt für den Sensor folgende Möglichkeiten:

- `is_pressed` gibt einen Wahrheitswert zurück, der angibt, ob der Berührungssensor aktuell im Zustand "gedrückt" ist.
- `wait_for_pressed()` lässt warten, bis der Berührungssensor gedrückt wird, `wait_for_released()` lässt warten, bis der Berührungssensor wieder ungedrückt ist. Beide Methoden erwarten als ersten Parameter die Angabe eines Zeitlimits, dass auf das Ereignis gewartet werden soll in Millisekunden und als zweiten Parameter eine Angabe darüber, wie lange nach dem Ereignis gewartet werden soll, ebenfalls in Millisekunden.
- `wait_for_bump()` erwartet dieselben beiden Parameter und lässt warten, bis der Berührungssensor einmal in den Zustand "gedrückt" bis zurück in den Zustand "ungedrückt" gewechselt hat.

Da für das angedachte Szenario weder Warten benötigt wird noch der vorherige Zustand des Berührungssensors von Relevanz ist, wird `is_pressed` wie folgt genutzt:

```
1 from ev3dev2.sensor.lego import TouchSensor
2 ts = TouchSensor()
3 print(ts.is_pressed)
```

Hier wird die Klasse des Berührungssensors `TouchSensor` aus dem Modul `ev3dev2.sensor.lego` importiert und daraufhin der Berührungssensor der Variable `ts` zugewiesen. Nun kann durch das Attribut `is_pressed` der Variable `ts` ein Wahrheitswert wiedergegeben werden, der angibt, ob der Berührungssensor gedrückt ist oder nicht.

Die großen Motoren und der mittlere Motor werden nahezu identisch verwendet. Zur Verwendung dieser bietet die Bibliothek eine Reihe an Möglichkeiten. Für die Klassen `LargeMotor` und `MediumMotor` können alle Methoden der dazugehörigen Klasse `Motor` genutzt werden, alle drei sind Subklassen der abstrakten Klasse `Motor`. Daraus ergibt sich eine Vielzahl an Methoden aus der Bibliothek, so unter anderem:

- `stop` die den Motor sofort anhalten lässt.
- `is_running` die ein einen Wahrheitswert zurückgibt, der angibt ob der Motor aktuell angetrieben wird.

- `on_for_rotations()` für umdrehungszahlbasierte Motorsteuerung, `on_for_degrees()` für gradbasierte Motorsteuerung, `on_to_position()` für positionsbasierte Motorsteuerung, und `on_for_seconds()` für zeitbasierte Motorsteuerung, die alle den jeweiligen Motor mit der Geschwindigkeit angegeben vom ersten Parameter bis zum Wert des zweiten Parameters als Umdrehungsanzahlen, Umdrehungsgrad, Motorposition oder Sekundenzahl drehen lassen.

- `on()`, die den Motor oder die Motorengruppe "für immer" drehen lässt. Für die Klasse `MoveTank` werden zwei Parameter erwartet, die jeweils die Geschwindigkeit des ersten und zweiten Motors angeben, für die Klasse `MoveSteering` werden ebenfalls zwei Parameter erwartet, hier gibt der erste die Lenkung und der zweite die Geschwindigkeit an; für die Klasse `Motor` werden drei Parameter erwartet, wobei der erste die Geschwindigkeit angibt.

Sie können wie folgt eingebunden und einzeln oder gemeinsam verwendet werden:

```
1 from ev3dev2.motor import LargeMotor, MediumMotor,
  OUTPUT_A, OUTPUT_B, SpeedPercent, MoveTank
2 lm_1 = LargeMotor(OUTPUT_A)
3 lm_2 = LargeMotor(OUTPUT_B)
4 mm = MediumMotor()
5 mt = MoveTank(OUTPUT_A, OUTPUT_B)
6 ms = MoveSteering(OUTPUT_A, OUTPUT_B)
7 mm.on_for_rotations(SpeedPercent(50), 2.5)
8 mt.on_for_rotations(SpeedPercent(100), SpeedPercent(50), 5)
9 ms.on_for_rotations(-100, SpeedPercent(50), 10)
```

Für die Nutzung der Motoren werden in Zeile 1 zunächst die Klassen des großen `LargeMotor` und des mittleren Motors `MediumMotor` aus dem Modul `ev3dev2.motor` importiert. Aus demselben Modul importieren wir auch die Klassen `OUTPUT_A` und `OUTPUT_B` für die beiden Motorenanschlüsse A und B des EV3, dann noch die Klasse `SpeedPercent` um Geschwindigkeit der Motoren in Prozenten angeben zu können und die Klasse `MoveTank` um zwei Motoren gemeinsam als Einheit ansprechen zu können. In Zeile 2 wird mittels `LargeMotor(OUTPUT_A)` der große Motor, der am Anschluss A angeschlossen ist der Variable `lm_1` zugewiesen. Die Angabe des Anschlusses ist notwendig, sobald mehr als ein Motor desselben Typs an den EV3 angeschlossen ist. Mit dem zweiten großen Motor wird in Zeile 3 demnach ebenso vorgegangen. Der mittlere Motor wird in Zeile 4 der Variable `mm` zugewiesen, da von diesem Motortyp nur einer angeschlossen wird, muss der Anschluss hierbei nicht angegeben werden. In Zeile 5 wird der Variable `mt` eine Motorengruppe zugewiesen, hier die beiden Motoren, die angeschlossen an den Anschlüssen A und B des EV3 sind. Das sind unsere beiden großen Motoren. Ebenso weisen wir dieselben Motoren in Zeile 6 als Motorengruppe der Variable `ms` zu. Die Methode `on_for_rotations()` erwartet in den Zeilen 7 bis

9 für einzelne Motoren zwei und für unsere Motorengruppe `mt` drei Parameter. Der letzte Parameter gibt die Anzahl an Umdrehungen, die der Motor bzw. die Motorengruppe auszuführen hat, an. Die ersten Parameter geben die Motorengeschwindigkeit an, hier für den mittleren Motor mittels `SpeedPercent(50)` auf 50-prozentige, also halbe, Geschwindigkeit gesetzt. Für die Motorengruppe müssen hier zwei Parameter gesetzt werden, da den Motoren auch unterschiedliche Drehgeschwindigkeiten aufgetragen werden können, wie hier einmal halbe Geschwindigkeit für den Motoren am Anschluss B und einmal volle Geschwindigkeit für den Motoren am Anschluss A, um eine Drehung zu erwirken. Für unsere Motorengruppe `ms` erwartet die Methode `on_for_rotations()` ebenfalls drei Parameter, wobei hier der erste Parameter ein Wert im Intervall von `-100` bis `100` zu sein hat und den Grad der Lenkung angibt. Dabei entspricht `-100` eine Lenkung von 90° nach links und `100` eine Lenkung von 90° nach rechts. Der zweite Parameter gibt wieder die Drehgeschwindigkeit und der dritte die Umdrehungsanzahl an. Hier ist also eine 90° Linksdrehung für 10 Umdrehungen bei halber Geschwindigkeit angegeben.

4.2 Implementierung des WLAN-Dongle

Der WLAN-Dongle bietet neben der offensichtlichen Konnektivität zwischen Rechner und EV3 die Möglichkeit als zusätzlicher Sensor genutzt zu werden. Mit diesem Sensor ist es möglich, die Access Points in Reichweite mit ihren Spezifikationen zu erfassen. Verwendet wird hierfür die Python-Bibliothek `access_points`[1].
Da `access_points` unter Linux-Systemen das Kommandozeilen-Programm `nmcli` nutzt, ist es notwendig das Paket `network-manager` zu installieren. Um Speicherplatz auf dem EV3 zu sparen und da der EV3 nur über eine begrenzte graphische Oberfläche verfügt, empfiehlt es sich, das Paket mittels `sudo apt-get --no-install-recommends install network-manager` zu installieren, um keine unnötigen Komponente wie zum Beispiel GUI-Elemente bei der Installation miteinzubeziehen.
Ist die Python-Bibliothek eingebunden und das Paket installiert, so kann der WLAN-Dongle wie folgt als Sensor genutzt und ausgelesen werden:

```
1 from access_points import get_scanner
2 wifi_scanner = get_scanner()
3 print(wifi_scanner.get_access_points())
```

Zunächst wird die Klasse `get_scanner` aus dem Modul `access_points` importiert und dann die Scanfunktion der Variable `wifi_scanner` zugewiesen. Danach können die aktuell verfügbaren Access Points mit ihren jeweiligen SSIDs, BSSIDs, Sicherheitsstandards und Signalstärken ausgegeben werden.

[1] https://github.com/kootenpv/access_points, zuletzt aufgerufen am 2020-04-10

Die Ausgabe ist dabei eine Liste aller verfügbaren Access Points nach dem Muster in Abbildung 4.2. Hierbei ist das erste Tupelelement des AccessPoint-Tupels stets die

```
[AccessPoint(ssid=AccessPointA, security=WPA2, quality=75,
bssid=1A:1B:1C:1D:1E:1F), ...,
AccessPoint(ssid=AccessPointZ, security=WPA1 WPA2,
quality=50, bssid=99:77:55:33:11:FF)]
```

Abbildung 4.1: Ausgabe von Access Point Metadaten

SSID, das zweite Tupelelement der Sicherheitsstandard, das dritte Tupelelement die Signalstärke als Prozentwert bestehend aus zwei Ziffern, und das vierte Tupelelement die BSSID mit den sechs Bytes in Hexadezimaldarstellung. Die Liste der Tupel wird intern als JSON-Objekt gehandhabt, dadurch ist es auch möglich, einfacher auf die Messdaten zuzugreifen.

4.3 Umgebungskarte

Die Umgebungskarte, die den Einsatzort definiert, in dem sich der EV3 bewegt wird in Python als Liste von Listen definiert. Mit der Funktion `new_room(y, x)` aus der Datei pathfinding.py wird eine neue Karte mit den Maßen von `y` Zellen auf der Y-Achse und `x` Zellen auf der X-Achse initialisiert. Die Liste von `y` Listen mit `x` Elementen besteht dann zunächst nur aus den Zellwerten 1. Daher wird die Funktion `reresh_room(room)` angewendet, die eine aktualisierte Umgebungskarte zurückgibt, für eine frisch initialisierte Karte also alle inneren Zellen auf den Zellwert 2 aktualisiert.

Die Funktion `save_room(room, roomName)` speichert eine mittels `room` übergebene Umgebungskarte als txt-Datei mit dem Dateinamen `roomName`. Die Funktion `load_room(roomName)` lädt eine gespeicherte Umgebungskarte mit dem Dateinamen `roomName` und gibt diese zurück. So ist es möglich verschiedene Einsatzorte zu speichern und wieder zu laden.

Mit der Funktion `refresh_room(room)` wird eine Umgebungskarte komplett aktualisert. Dabei wird die Karte zunächst auf Hindernisse also Zellen mit dem Zellwert 0 untersucht. Ist eine Zelle mit dem Zellwert 0 gefunden, so werden mittels der Funktion `cell_zero(room, y, x)` alle direkt benachbarten Zellen aktualisiert. Darauf wird die Umgebungskarte auf Zellen mit dem Zellwert 1 untersucht. Für jede Zelle mit dem Zellwert 1 wird dann mittels der Funktion `cell_one(room, y, x)` überprüft, ob der Zellwert auf 2 aktualisiert werden kann. Die Funktion gibt am Ende den

komplett aktualisierten Raum zurück. Da dies erhebliche Rechenzeit beansprucht und ein paar Sekunden dauert, wird dringend empfohlen, diese Funktion nur zur Initialisierung eines Raumes zu verwenden.

Die Funktion `cell_zero(room, y, x)` wird verwendet, wenn die Zelle `(y, x)` der Umgebungskarte `room` den Wert 0 hat und die direkt benachbarten Zellen aktualsiert werden sollen. Dabei werden mittels der Funktion `cell_value_zero(room, y, x)` die Zellen, deren Zellwert 2 ist, auf 1 gesetzt und die Zellen, deren Zellwert 0 oder 1 ist, in ihrem Zellwert erhalten. Die Funktion gibt dann die aktualisierte Umgebungskarte für diese Zelle zurück. Die Funktion `cell_value_zero(room, y, x)` ist dabei lediglich eine Behilfsfunktion, die für eine Zelle, deren Zellwert 0 ist, 0 zurückgibt und für eine Zelle, deren Wert 1 oder 2 ist, 1 zurückgibt.

Durch die Funktion `cell_one(room, y, x)` wird überprüft, ob die Zelle `(y, x)` ausschließlich an Zellen angrenzt, deren Zellwert 1 ist. Wenn dem so ist, dann wird der Wert der Zelle auf 2 aktualisiert und die aktualisierte Umgebungskarte zurückgegeben.

Eine effizientere Funktion zur Aktualisierung einzelner Zellwerte ist die Funktion `refresh_cell(room, y, x, free)`. Diese erwartet eine Umgebungskarte `room`, eine Zellenangabe durch die Korrdinaten `y` und `x` und einen Boolean `free`. Der Boolean gibt dabei an, ob eine Zelle frei ist (`True`), also kein Hindernis beinhaltet oder nicht frei ist (`False`), also ein Hindernis beinhaltet. Dabei deckt diese Funktion drei Fälle ab:

1. Ist die zu aktualisierende Zelle frei, so wird, falls die Zelle zuvor nicht frei war, der Zellwert auf 1 gesetzt, dann für alle direkt benachbarten Zellen, deren Zellwert 1 beträgt, mittels der Funktion `cell_one(room, y, x)` überprüft, ob diese auf den Zellwert 2 aktualisiert werden können. Zuletzt wird dies ebenfalls für die aktualisierte Zelle geprüft.

2. Ist die zu aktualisierende Zelle nicht frei und ihr bisheriger Zellwert 2, so wird der Zellwert aller direkt benachbarten Zellen auf 1 gesetzt.

3. Ist die zu aktualisierende Zelle nicht frei und ihr bisheriger Zellwert 1, so werden die Zellwerte aller direkt benachbarten Zellen, deren Zellwert 2 beträgt auf den Zellwert 1 gesetzt.

Dann gibt die Funktion eine aktualisierte Umgebungskarte zurück.

4.4 Bewegung von Zelle zu Zelle

Zur Bewegung von Zelle zu Zelle muss zunächst beachtet werden, dass das Fahrzeug stets eine Orientierung beziehungsweise Richtung hat. Die Orientierung wird hier im Folgenden als Tupel `(y, x)` betrachtet, dabei gilt für $(x, y) : x, y \in \{-1, 0, 1\} \backslash (0, 0)$. Das Tupel $(0, 0)$ fällt heraus, da es keiner Richtung entspräche. Dabei entspräche ein Y-Wert von 1 einer Positiven Orientrierung in Y-Richtung, ein positiver Wert in Y- und X-Richtung einer diagonalen Orientierung in positiver Y- und X-Richtung und so weiter. Verdeutlicht ist dies noch einmal in Abblidung 4.2.

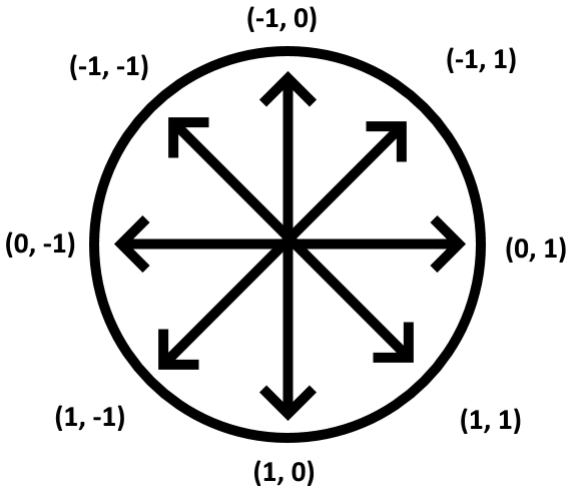


Abbildung 4.2: Visualisierung der Orientierung des Fahrzeuges anhand der dafür genutzten Tupel

Die Funktion `move_to_cell(start, dest, orient, speed)` der Datei ev3move.py erwartet einen Startpunkt `start` in Tupelform, einen Zielpunkt `dest` in Tupelform, eine Orientierung `orient` nach dem eben definiertem Muster und einen Geschwindigkeitswert `speed`. Zunächst wird überprüft, ob das Fahrzeug die nötige Orientierung hat, um die Zielzelle zu erreichen. Ist das Fahrzeug noch nicht in die richtige Richtung gedreht, so wird die Orientierung mittels der Funktion `adjust_orient(orient1, orient2)` angepasst. Ist die Orientierung korrekt, so wird das Fahrzeug mittels der Funktion `move_cell(orient, speed)` zum Zielpunkt bewegt.

Bei der Funktion `move_cell(orient, speed)`, die einen Orientierungswert `orient` und einen Geschwindigkeitswert `speed` erwartet, handelt es sich im Prinzip um simples Geradeausfahren. Dabei wird zunächst geprüft, ob sich das Fahrzeug diagonal

oder gerade innerhalb des Rasters der Umgebungskarte befindet. Je nach Orientierung wird das Fahrzeug dann geradeaus über den Abstand einer Zelle bewegt. Dabei wird die zurückzulegende Strecke bei einer geraden Ausrichtung des Fahrzeugs mit 10 cm und bei einer diagonalen Ausrichtung des Fahrzeugs mit $\sqrt{200}$ cm berechnet. Da zur Bewegung des Fahrzeugs die in Abschnitt 4.1 beschriebene Funktion `on_for_rotations()` verwendet wird, muss die zurückzulegende Strecke in Motorumdrehungen angegeben werden. Hierfür berechnen wir: Streckenwert / (π * Raddurchmesser), wobei der Raddurchmesser 4,3 cm beträgt. Da für den Antrieb der hinteren Räder zwei Zahnräder mit jeweils 36 und 12 Zähnen verwendet werden, müssen die notwendigen Motorumdrehungen noch um den Faktor 3 geteilt werden.

Die Funktion `adjust_orient(orient1, orient2)` erwartet zwei Orientierunswerte und wendet, sofern notwendig, ein Wendemanöver an, um Das Fahrzeug aus der Ausgangsorientierung `orient1` in die Zielorientierung `orient2` zu bringen. Hierfür wird die Differenz beider Orientierungen gebildet und diese in der kürzeren Version genutzt. Also wird für einen Winkel größer als 180° oder kleiner als -180° wiederum die Differenz zu 360° oder -360° gebildet, so dass stets die kürzere Version des Wendemanövers genutzt werden kann. So wird sich beispielsweise um 45° anstatt um 315° gedreht.
Für das eigentliche Wendemanöver wird der Abschnitt aus Abbildung 4.3 genutzt. Hier wird zunächst überprüft, ob es sich um eine negative oder positive Gradzahl für die Drehung handelt. Dann fährt das Fahrzeug 8,5 cm vor, damit es sich auf dem Mittelpunkt der aktuellen Position drehen kann. Dort nimmt es dann die Drehung vor und setzt zurück. Bei zurücksetzten werden 8,75 cm genutzt, da sich diese beim Testen als hilfreicher erwiesen und eher zu dem erwünschten Ergebnis führten. Die 15,6 cm in der dritten und siebten Zeile entstammen ursprünglich der mathematischen Berechnung des Wendekreises und wurden dann ebenfalls angepasst.

Bei der Implementierung der Wendemanöver ist es vermehrt zu Schwierigkeiten gekommen. Eine exakte Lenkung war zunächst überhaupt nicht möglich. Es erwies sich als sehr hilfreich die Gummibereifung des Stützrades zu entfernen und das Gewicht des Fahrzeuges nach Hinten zu verlagern. Hierfür wurde der EV3-Baustein oberhalb der hinteren Räder angebracht.

4.5 Umgebungserfassung mit dem Ultraschallsensor

In der Datei ultrasonic_measures.py existiert eine Funktion `update_cells(room, position, orient)`, diese erwartet eine Umgebungskarte `room` einen Standort `position` und eine Orientierung `orient`. Die Funktion nutzt dabei die bekannte Position des Fahrzeuges um die Umgebungskarte auf Grundlage der vorgenommenen Messungen zu aktualisieren. Dazu wird der Ultraschallsensor jeweils

```
if degree > 0:
  MoveTank(OUTPUT_B, OUTPUT_D).on_for_rotations(10, 10,
  (8.5/(math.pi * 4.3))/3)
  MoveTank(OUTPUT_B, OUTPUT_D).on_for_rotations(-10, 10,
  (15.6*math.pi*(degree/360)/(math.pi * 4.3))/3)
  MoveTank(OUTPUT_B, OUTPUT_D).on_for_rotations(-10, -10,
  (8.75/(math.pi * 4.3))/3)
else:
  MoveTank(OUTPUT_B, OUTPUT_D).on_for_rotations(10, 10,
  (8.5/(math.pi * 4.3))/3)
  MoveTank(OUTPUT_B, OUTPUT_D).on_for_rotations(10, -10,
  (15.6*math.pi*(abs(degree)/360)/(math.pi * 4.3))/3)
  MoveTank(OUTPUT_B, OUTPUT_D).on_for_rotations(-10, -10,
  (8.75/(math.pi * 4.3))/3)
```

Abbildung 4.3: Wendemanöver der Funktion adjust_orient()

in fünf verschiedene Positionen gebracht um die Umgebung zu erfassen. Zunächst dreht er sich um 90° nach links, vier mal um je 45° zurück, bis er sich in Bezug auf die Ausgangssituation 90° nach rechts gedreht hat. Zum Abschluss dreht er sich wieder in die Ausgangslage zurück. Dabei ist es möglich die Variable `depth` zu verändern. Standardmäßig hat diese den Wert 2. Sie gibt an, ie viele Zellen das Fahrzeug vom Ultraschallsensor aus gerechnet maximal untersuchen soll.

4.6 Routenfindung

Zur Routenfindung soll ein einfacher A*-Suchalgorithmus verwendet werden. Hier muss zum Glück nicht das Rad neu erfunden werden. Einen funktionierenden A*-Suchalgorithmus für Python stellt Nicholas Swift[2] zur Verfügung. Geringfügig angepasst, denn gültige Zellwerte zur Routenfindung sind hier 2 und nicht 0, kann die Funktion `astar(maze, start, end)` ebenfalls in der Datei pathfinder.py eine Umgebungskarte `maze`, einen Startort `start` und ein Zielort `end` nutzen um einen optimalen Pfad vom Start- zum Zielort zurückzugeben.

4.7 Speicherung- der Sensor und Statusdaten

Die Funktion `save_data(status, position, orient, usOrient)` ist in der Datei save2csv.py definiert, diese erwartet einen Status `status` des Fahrzeugs,

[2] https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2, zuletzt aufgerufen am 2020-04-24

die Position des Fahrzeuges `position` als Tupel (y, x), sowie die Orientierung `orient` des Fahrzeugs und des Ultraschallsensors `usOrient`, beide als Tupel (y, x). Sie schreibt eine neue Zeile in die bestehende CSV-Datei dataProtocol.csv oder erzeugt diese, falls noch nicht vorhanden. Dabei ist das erste Element der neuen Zeile stets ein Zeitstempel im ISO 8601 Format. Dieser wird durch `datetime.datetime.now().replace(microsecond=0).isoformat()` erzeugt und befindet sich dann im Format YYYY-MM-DDThh:mm:ss. Die nächsten drei Elemente sind die Sensordaten, so wird mit `UltrasonicSensor().distance_centimeters()` der zuletzt gemessene Abstand des Ultraschallsensors ausgelsene, mit `GyroSensor().angle()` wird die aktuelle Neigung des Gyroskopsensors ausgelesen und mittels `TouchSensor().is_pressed()` der aktuelle Status des Berührungssensors und damit des Bumpers ausgelesen. Die letzten vier Elemente der Zeile sind die vier an die Funktion übergebenen Argumente.

4.8 Hauptteil

Der Hauptteil ist in der Datei pathfinder_test.py implementiert. Hier wird zunächst angegeben, was Ausgangspunkt, Ausgangsorientierung, Zielpunkt und zu verwendenende Umgebungskarte für einen Einsatz sind. Dann wird rundenbasiert überprüft, ob der Gyroskopsensor oder der Berührungssensor ausschlagen und ggf. reagiert. Zudem wird die Nutzung der Routenfindung hier initialisiert und regelmäßig neue Zeilen mit Statusinformationen in die CSV-Datei geschrieben.

Kapitel 5

Evaluation

Dieses Kapitel beschäftigt sich mit der Evaluation. So werden in Abschnitt 5.1 die Ortungsmöglichkeiten auf Grundlage von Messungen von Signalstärken von Access Points ausgewertet. In Abschnitt 5.2 wird das autonome Fahren des entwickelten Fahrzeug auf Grundlage von verschiedenen Tests ausgewertet.

5.1 Evaluation der Ortungsmöglichkeiten auf Basis von WiFi

Zur Evaluierung der Genauigkeit, die mittels Ortung auf Basis von Profilen von Signalstärken verschiedener Access Points möglich ist, wurden die Signalstärken verschiedener Access Points, die aktuell am Standort verfügbar waren, ermittelt. Von den verfügbaren Access Points wurden alle fünf ausgewählt, deren Signalstärke 30 überstieg um möglichst signifikante Ergebnisse erzielen zu können. Als Messung dienten dann jeweils zehn Erfassungen der Signalstärken der fünf Access Point im Abstand von jeweils 60 Sekunden. Dieselbe Messung wurde dann für vier weitere Räume wiederholt. Hier wurde sich zunächst nur auf Räume beschränkt, um zu überprüfen, ob ein solch grober Ortungsansatz funktionieren könnte und die erwählten Access Points ausreichend stabile Ergebnisse liefern.

Auf Grundlage der Implementation in Abschnitt 4.2 wurde die Funktion `test_qualities()`, zu sehen in Abbildung 5.1, geschrieben. Diese bekommt eine Liste an BSSIDs übergeben. Sie führt dann mittels einer For-Schleife die Messung nach Access Points durch, durchsucht alle Ergebnisse nach den zuvor übergebenen Access Points anhand ihrer BSSIDs und gibt, sofern gefunden deren Qualität aus. Dann wartet die Funktion 60 Sekunden auf die nächste Durchführung der For-Schleife, so lange bis diese zehnmal ausgeführt wurde.

```
1  def test_qualities(bssids):
2    for count in range(10):
3      aps=get_scanner().get_access_points()
4      for x in range(len(bssids)):
5        for ap in aps:
6          if ap.bssid == bssids[x]:
7            print(bssids[x] + ' : ' + str(ap.quality))
8      time.sleep(60)
```

Abbildung 5.1: Funktion zur Messung der Signalstärken von Access Points anhand ihrer BSSID im Abstand von 60 Sekunden

In Abbildung B.3 sind die gemessenen Signalstärken der fünf Access Points zu allen zehn Messungen in Raum 1 in einem Diagramm aufgeführt. Dabei fällt vor allem der fünfte Access Point auf, da dieser bei vier der Messungen gar nicht verfügbar war, bei den anderen sechs Messungen jedoch stabile Signalstärken von 55 oder 57 verzeichnen konnte. Die Access Points 2 bis 4 hingegen liefern alle deutlich stabilere Werte, die Standardabweichung befindet sich hierbei jeweils bei 10,7. Der erste Access Point ist mit einer Standardabweichung von 15,4 instabiler in der Signalstärke und verzeichnet bei dieser einen Tiefstwert von 0 allerdings auch einen Maximalwert von 55.

Die gemessenen Signalstärken für Raum 2 sind in Abbildung B.4 zu erkennen. Hier sticht erneut der fünfte Access Point ganz besonders hervor, denn er konnte bei keiner der 10 Messungen erkannt werden. Aber auch der erste Access Point wurde lediglich bei den ersten beiden Messungen mit einer Signalstärke von nur 10 erfasst. Der Access Point 3 taucht in den Messungen mit größtenteils stabilen Werten auf, da jedoch zwei stark abweichende Werte gemessen wurden, ist die Standardabweichung hier bei 18,4. Damit sind die Signalstärken des Access Point 3 auf alle 10 Messungen betrachtet eher instabil.

In den Messungen in Raum 3, zu sehen in Abbildung B.5 und den Messungen in Raum 4, zu sehen in Abbildung B.6 fällt Access Point 3 mit besonders stabilen Messwerten auf. Die Signalstärken sind hier bei allen Messungen so ähnlich, dass die Standardabweichung in Raum 4 nur 0,8 und in Raum 3 sogar nur 0,6 beträgt. Auch Access Point 5 fällt in den Messungen in beiden Räumen wieder auf. Hier beträgt die Signalstärke bei jeder erfolgreichen Messung etwas über 50, in etwa der Hälfte der Messungen kann jedoch keine Signalstärke gemessen werden.

Bei den Messungen in Raum 5, zu sehen in Abbildung B.7, konnten Access Point

1 und Access Point 5 jeweils nur bei zwei der Messungen erkannt werden. Da die dort gemessenen Signalstärken allerdings weniger stark waren, geben die Standardabweichungen mit 3,4 für Access Point 1 und 11,8 für Access Point 5 eine höhere Stabilität an. Mit einer Standardabweichung von 19,4 ist hier besonders der Access Point 3 instabil in seiner Signalstärke.

Die Standardabweichungen sind noch einmal gemeinsam in Abbildung B.1 visualisiert. Hier wird erneut besonders deutlich, dass Access Point 5 bezüglich der gemessenen Signalstärken wenig stabil und damit kaum aussagekräftig ist. Für die Durchschnittsignalstärken, hier in Abbildung B.2 zu erkennen, wurden daher nur die Access Points 1 bis 4 betrachtet. Hier ist gut zu erkennen, dass die Werte in den Räumen 2, 3 und 4 sehr ähnlich sind, jedoch in Raum 1 deutlich abweichen und in Raum 5 aufgrund des Access Point 4 unterschiedlich wirken. Zusammenfassend ergeben die Messdaten jedoch das ernüchternde Ergebnis, dass selbst wenn man sich nur auf die stabileren Access Points bezieht, die Fluktuationen in der Signalstärke so erheblich sind, dass auf Grundlage der Signalstärken nicht einmal die Einordnung in Räume möglich ist. Von einer Ortung auf Zellebene, so wie sie für dieses Projekt nötig wäre, ist daher abzusehen.

5.2 Evaluation des autonomen Fahrens

Das autonome Fahren als solches stellte sich als größere Herausforderung dar, als Eingangs vermutet. Insbesondere zur Erfassung, Speicherung bzw. Verarbeitung der Umgebung sind teilweise komplexe Konzepte notwendig. Zur Evaluation der in dieser Arbeit beschriebenen Lösungen sollte das Fahrzeug eine kleine Route berechnen und diese abfahren. Zunächst ohne Hindernis, dann mit Hindernis und dann ein weiteres mal mit dem selben Hindernis, dass nun in der Umgebungskarte gespeichert sein sollte. Die Ergbenisse aller drei Tests sind per Video aufgenommen und befinden sich in den angehangenen Dateien. Auf den Aufnahmen ist ein Quadrat zu sehen, in dem sich das Fahrzeug eingangs befindet. Dies bildet die neun Zellen ab, die das Fahrzeug einnimmt. Ebenfalls ist ein Pfad zum Zielort gekennzeichnet, der grob dem entspricht, den der A*-Algorithmus berechnen sollte. In allen drei Tests sollte das Fahrzeug am Ziel ankommen.
Der erste Test verlief erfolgreich und bildete ebenfalls eine CSV-Datei und eine hindernisfrei Umgebungskarte ab. Die Tests zwei und drei hingegen waren wenig erfolgreich. Zwar erreichte das Fahrzeug in beiden Fällen die grobe Richtung des Ziels, jedoch war dies nicht wirklich nah am erwarteten Ergebnis. Zudem verhielt sich das Fahrzeug "seltsam" so wurden Wendemanöver vorgenommen, die so auf Grundlage der erfassten Umgebung wenig plausibel erschienen.
Hier lässt sich abschließend sagen, dass trotz erheblicher Bemühungen das autonome

Fahren eine nicht zu unterschätzdende Herausforderung darstellt und insbesondere die Abbildung im kleinen Rahmen zusätzliche Herausforderungen bietet, die teils nur sehr schwer zu lösen sind.

Kapitel 6

Zusammenfassung und Ausblick

Zusammenfassend lässt sich sagen, dass festgestellt werden konnte, dass es nicht ohne erheblichen Aufwand möglich ist Fähigkeiten autonom fahrender Fahrzeuge im kleinen Rahmen zu reproduzieren. Allerdings sind grundlegende Fahraufgaben theoretisch realisierbar auch ohne die Nutzung von erheblich sensiblen Daten, wie Kamerabildern oder Standortdaten. Jedoch lässt sich vermuten, dass diese die Qualität des autonomen Fahrens im großen Rahmen nicht unwesentlich verbessern.

6.1 Ausblick mit anderer Technik

Kleinere Schwierigkeiten des autonomen Fahrens mit einem EV3 als Bordcomputer ließen sich mit Sicherheit durch die Nutzung eines leistungsfähigeren Rechners lösen. So könnte man beispielsweise die Zellen der hier genutzten Karten erheblich kleiner skalieren und somit eine deutlich genauere Routenfindung ermöglichen. Ebenfalls könnte man verschiedene Funktionen parallelisieren und so beispielsweise während des Fahrens konstant die Umgebung mit dem Ultraschallsensor erfassen, um eine noch genauere Karte und somit auch effizientere Routen erstellen zu können. Der Einsatz von mehreren Ultraschallsensoren auch näher am Boden, ließe auch die Erkennung tieferer Hindernisse zu. Ein weiterer Gyroskopsensor könnte genauere Kurse und somit effizientere Routen und Ausweichmanöver ermöglichen. Bei der Verwendung weiterer Sensoren ist allerdings zu beachten, dass der EV3 lediglich vier Anschlüsse für Sensoren bereit stellt. Würde man also am EV3 weiterarbeiten, so wären Splitter für die Sensoranschlüsse oder eine andere Technik als Kernelement, beispielsweise ein Raspberry Pi, zu empfehlen.

6.2 Fazit

Es lässt sich kaum bestreiten, dass viele Ziele dieser Arbeit nur teilweise erreicht werden konnten. Insbesondere in der Erkenntnis, woran es gescheiter sein könnte und der Vielzahl an untersuchten Technologien und entwickelten Konzepten, kann der Mehrwert dieser Arbeit gesehen werden. Auf Grundlage des geschaffenen, lassen sich leicht feinere und schärfer auf einzelne Fragestellungen zugeschnittene Aufgaben bewältigen. Hinsichtlich der Herausforderung des autonomen Fahrens ohne eine Serververbindung und ohne eine Vielzahl an sensiblen Sensordaten zu erheben, lässt sich allerdings beobachten, dass diese Hindernisse nicht ausschlaggebend für die bei dieser Arbeit aufgetretenen Probleme waren und sollten sie genutzt werden, vermutlich auch nicht Abhilfe verschaffen. Dies liegt vor allem an der Natur der aufgetretenen Problemen.

Literaturverzeichnis

[AUD20] AUDI AG: *Der neue Audi A8 hochautomatisiertes Fahren auf Level 3.* https://www.audi-mediacenter.com/de/per-autopilot-richtung-zukunft-die-audi-vision-vom-autonomen-fahren-9305/der-neue-audi-a8-hochautomatisiertes-fahren-auf-level-3-9307, zuletzt aufgerufen am 2020-01-31, 2020.

[ben19] BENQ.EU: *USB Wireless Dongle | WDRT8192.* https://www.benq.eu/de-de/projector/accessory/wdrt8192/specifications.html, zuletzt aufgerufen am 2020-01-30, 2019.

[gena] GENERATIONROBOTS.COM: *Gyroscopic Sensor for Lego Mindstorms NXT.* https://www.generationrobots.com/en/401183-gyroscopic-sensor-for-lego-mindstorms-nxt.html, zuletzt aufgerufen am 2019-12-16.

[genb] GENERATIONROBOTS.COM: *Hochpräziser Lego Mindstorms NXT IR-Sensor mittlerer Reichweite von MindSensors.* https://www.generationrobots.com/de/401337-infrarot-sensor-fuer-programmierbare-lego-mindstorms-nxt-roboter.html, zuletzt aufgerufen am 2019-12-16.

[genc] GENERATIONROBOTS.COM: *Lego Mindstorms NXT Ultraschallsensor.* https://www.generationrobots.com/de/401180-ultraschallsensor-lego-mindstorms-nxt.html, zuletzt aufgerufen am 2019-12-16.

[Jia17] JIAO, JICHAO AND YUAN, LIBIN AND TANG, WEIHUA AND DENG, ZHONGLIANG AND WU, QI: *A Post-Rectification Approach of Depth Images of Kinect v2 for 3D Reconstruction of Indoor Scenes.* ISPRS International Journal of Geo-Information, 6:349, 11 2017.

[Kep13] KEPSKI, MICHAL AND KWOLEK, BOGDAN: *Human Fall Detection Using Kinect Sensor*, Band 226, Seiten 743–752. 05 2013.

[LEG13] LEGO Group: *LEGO MINDSTORMS EV3 Firmware Developer Kit*. https://www.lego.com/cdn/cs/set/assets/blt77bd61c3ac436ea3/LEGO_MINDSTORMS_EV3_Firmware_Developer_Kit.pdf, zuletzt aufgerufen am 2020-01-31, 2013.

[LEG15] LEGO Group: *LEGO Mindstorms EV3 Bedienungsanleitung*. https://www.lego.com/cdn/cs/set/assets/bltd65492add9dd16ba/User_Guide_LEGO_MINDSTORMS_EV3_11_All_DE.pdf, zuletzt aufgerufen am 2020-01-12, 2015.

[LEG19a] LEGO Group: *EV3 Berührungssensor*. https://www.lego.com/de-de/product/ev3-touch-sensor-45507, zuletzt aufgerufen am 2019-12-06, 2019.

[LEG19b] LEGO Group: *EV3 Gyrosensor*. https://www.lego.com/de-de/product/ev3-gyro-sensor-45505, zuletzt aufgerufen am 2019-12-06, 2019.

[LEG19c] LEGO Group: *EV3-Infrarot-Detektor*. https://www.lego.com/de-de/product/ev3-infrared-sensor-45509, zuletzt aufgerufen am 2019-12-06, 2019.

[LEG19d] LEGO Group: *EV3 Ultraschallsensor*. https://www.lego.com/de-de/product/ev3-ultrasonic-sensor-45504, zuletzt aufgerufen am 2019-12-06, 2019.

[LEG19e] LEGO Group: *LEGO® MINDSTORMS® - Kompatibilität mit NXT*. https://education.lego.com/de-de/support/mindstorms-ev3/nxt-compatibility, zuletzt aufgerufen am 2019-12-06, 2019.

[LEG19f] LEGO Group: *LEGO MINDSTORMS EV3 MIT DEM WLAN-NETZWERK VERBINDEN*. https://www.lego.com/de-de/service/help/produkte/themen-bausatze/mindstorms/lego-mindstorms-ev3-mit-dem-wlan-netzwerk-verbinden-408100000007883, zuletzt aufgerufen am 2020-01-31, 2019.

[net08] netgear.com: *Wireless-N 150 USB Adapter mit Cradle WNA1100 - Datenblatt*. http://www.downloads.netgear.com/files/GDC/datasheet/de/WNA1100.pdf, zuletzt aufgerufen am 2020-01-31, 2008.

[SAE18] SAE Institute: *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, 06 2018.

[San17] Sanjay Seshan and Arvind Seshan: *ADVANCED EV3 PROGRAMMING LESSON - Gyro Sensor Revisited*. http://ev3lessons.com/

en/ProgrammingLessons/advanced/GyroRevisited.pdf, zuletzt aufgerufen am 2019-12-06, 2017.

[Tes19] TESLA: *Fahren in der Zukunft.* https://www.tesla.com/de_DE/autopilot?redirect=no, zuletzt aufgerufen am 2019-12-16, 2019.

[www16] WWW.ROBOT-ADVANCE.COM: *NXT TOUCH SENSOR.* https://www.robot-advance.com/EN/art-nxt-touch-sensor-582.html, zuletzt aufgerufen am 2019-12-16, 2016.

Anhang A

Bilder des Fahrzeuges

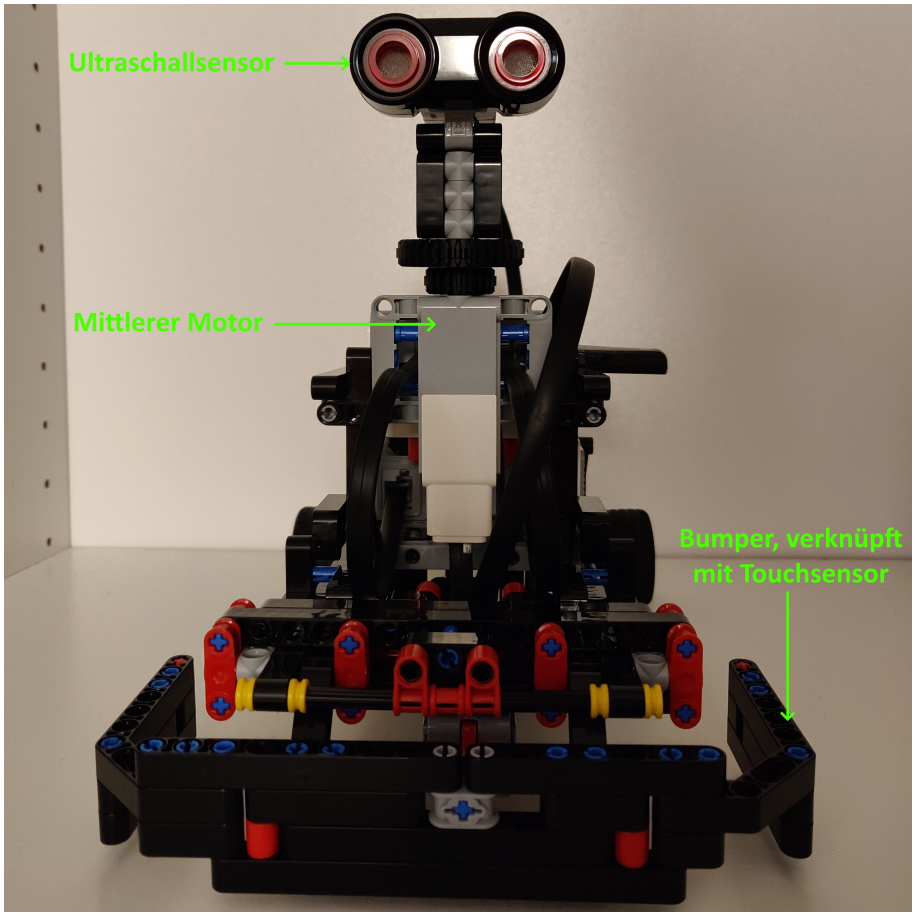


Abbildung A.1: Frontseite des Fahrzeuges

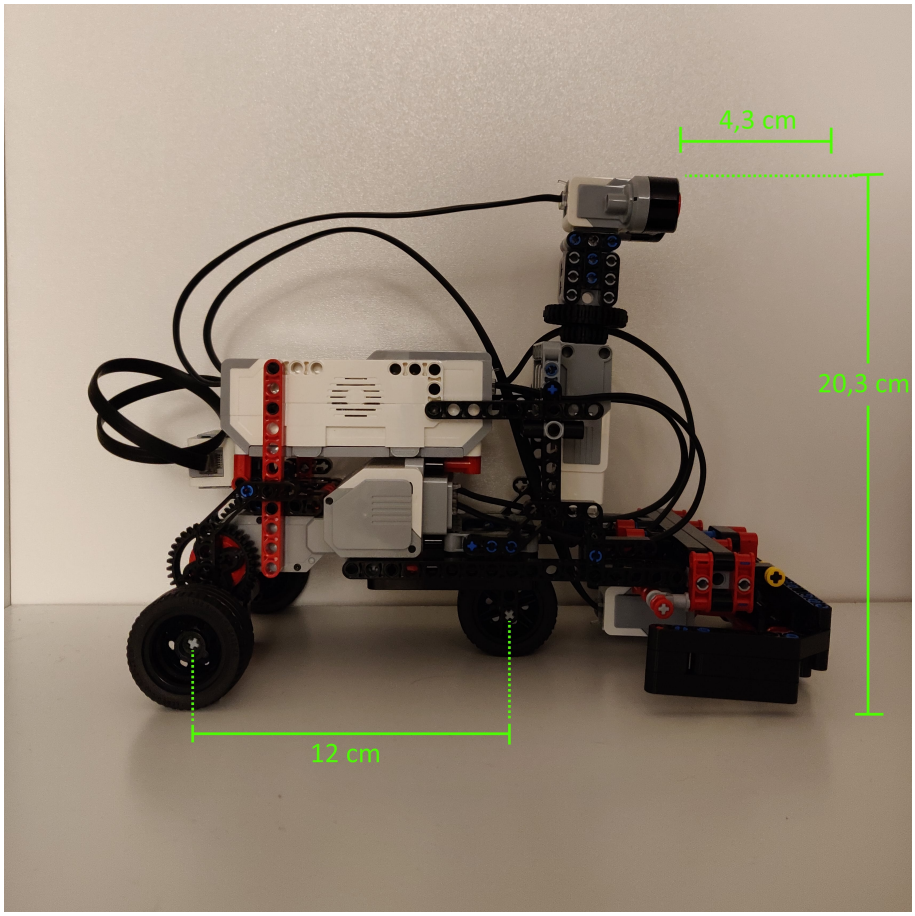


Abbildung A.2: Rechte Seite des Fahrzeuges

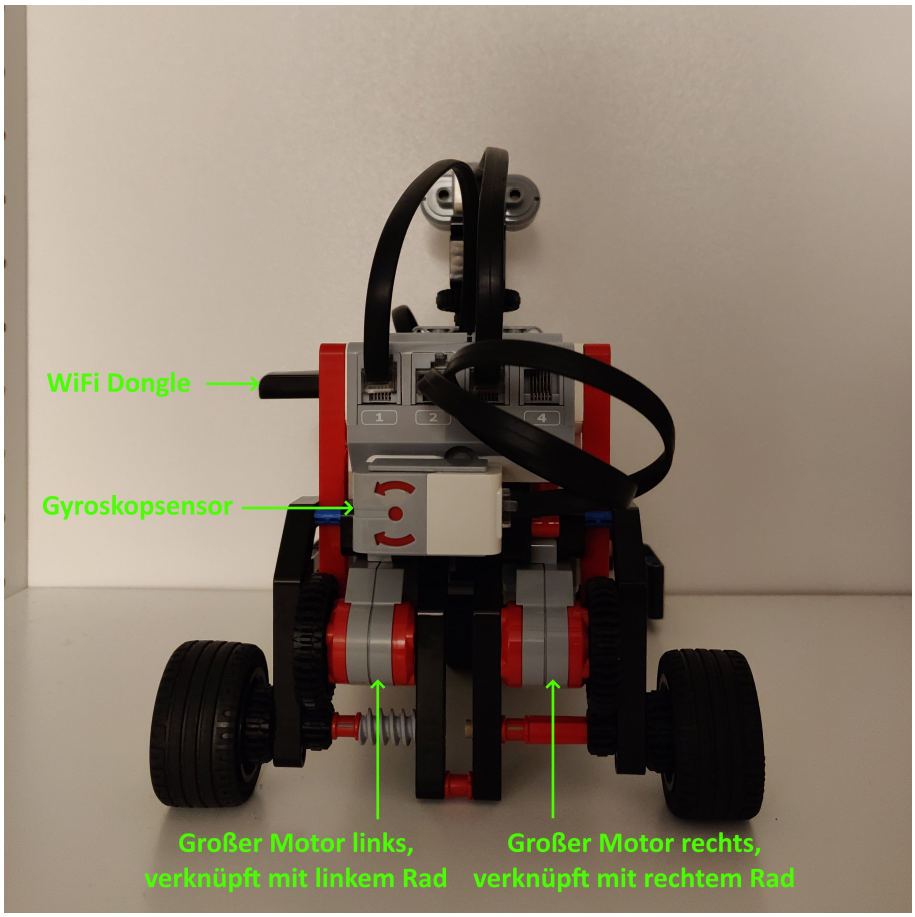


Abbildung A.3: Rückseite des Fahrzeuges

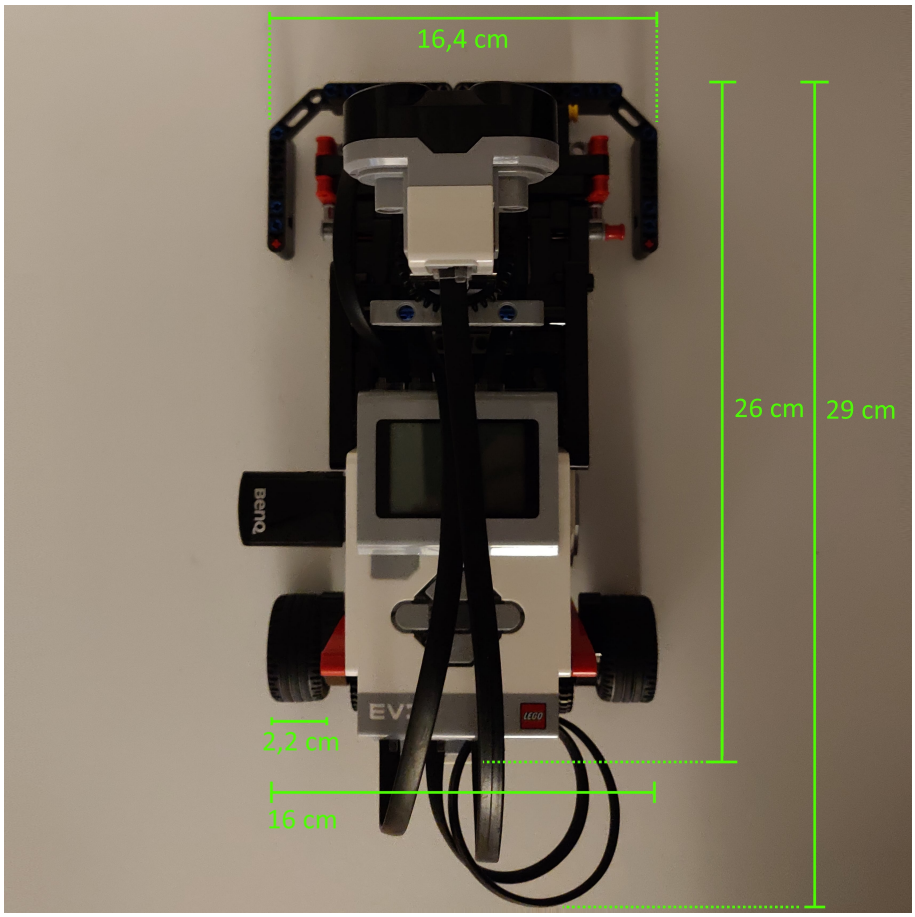


Abbildung A.4: Oberseite des Fahrzeuges

Anhang B

Messdaten des WLAN-Dongles

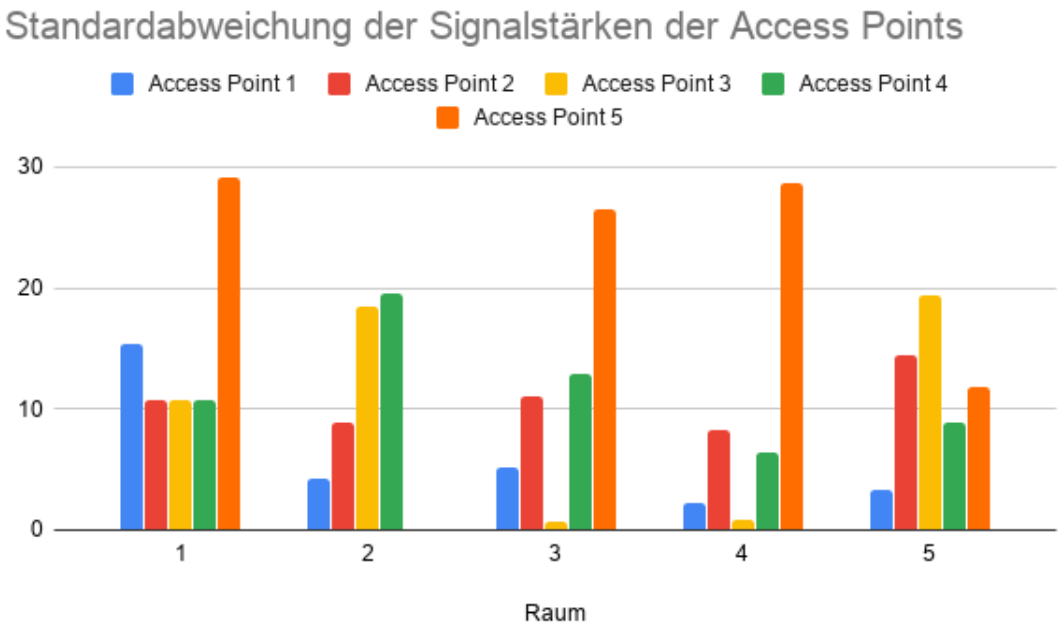


Abbildung B.1: Standardabweichung der Signalstärken der Access Points über zehn Messungen je Raum

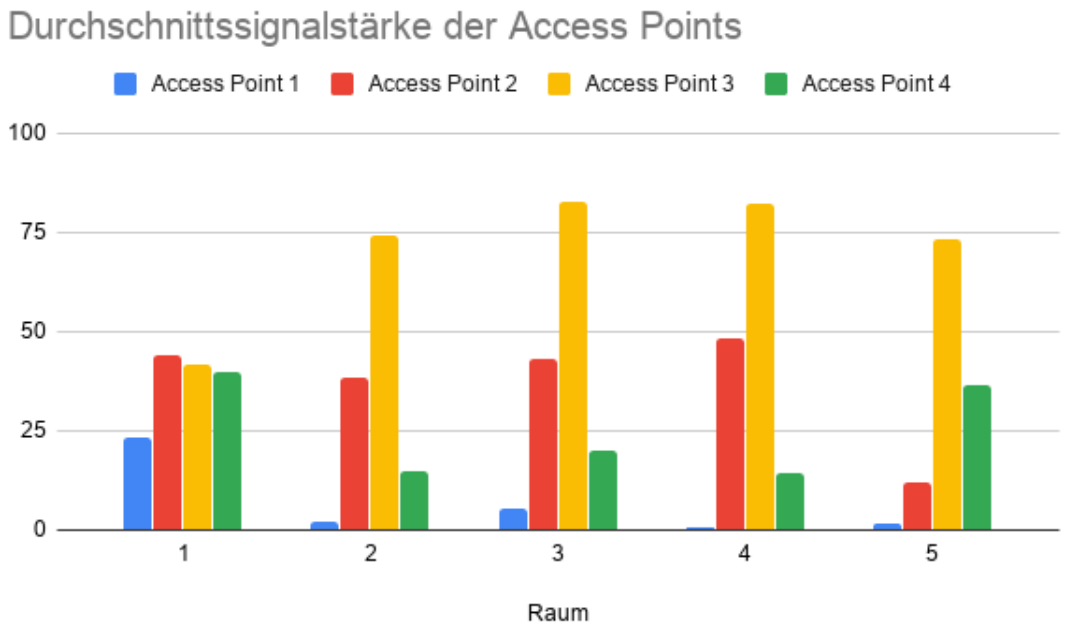


Abbildung B.2: Durchschnitt der Signalstärken der Access Points über zehn Messungen je Raum

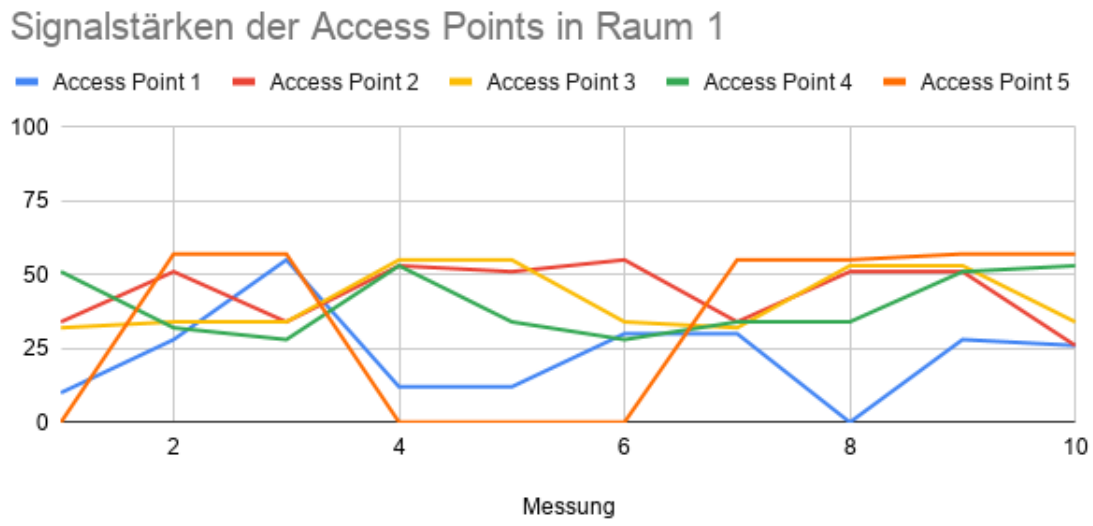


Abbildung B.3: Signalstärken der Access Points in Raum 1 über zehn Messungen

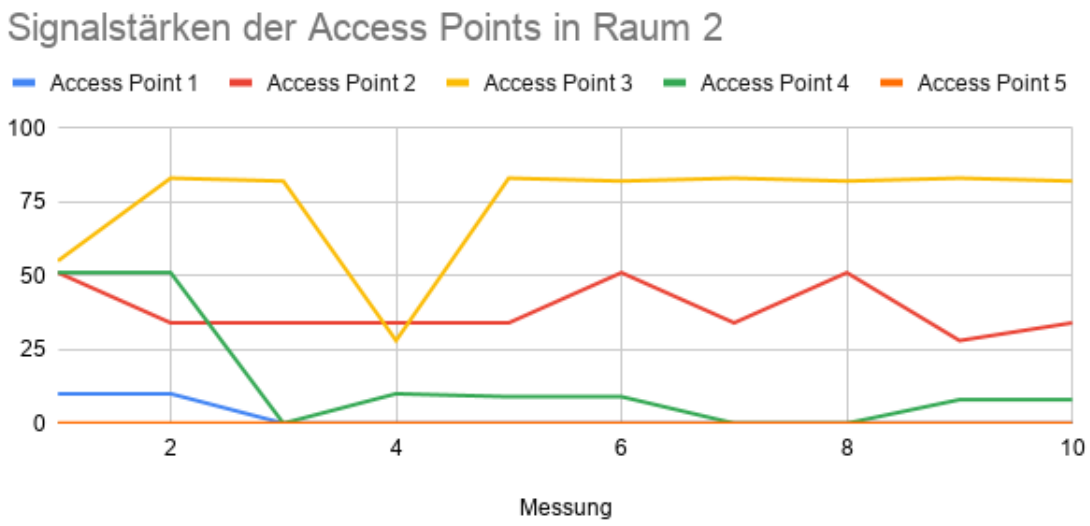


Abbildung B.4: Signalstärken der Access Points in Raum 2 über zehn Messungen

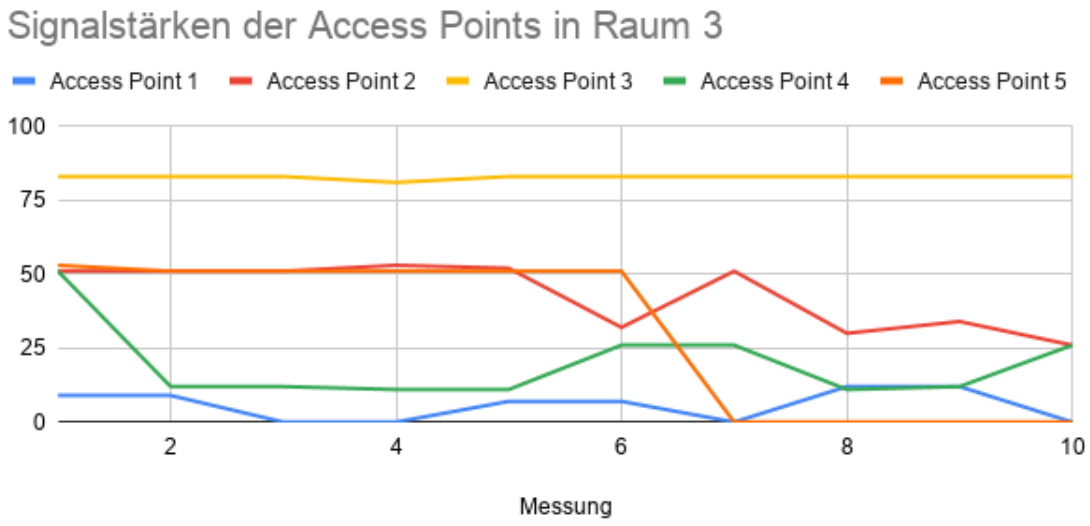


Abbildung B.5: Signalstärken der Access Points in Raum 3 über zehn Messungen

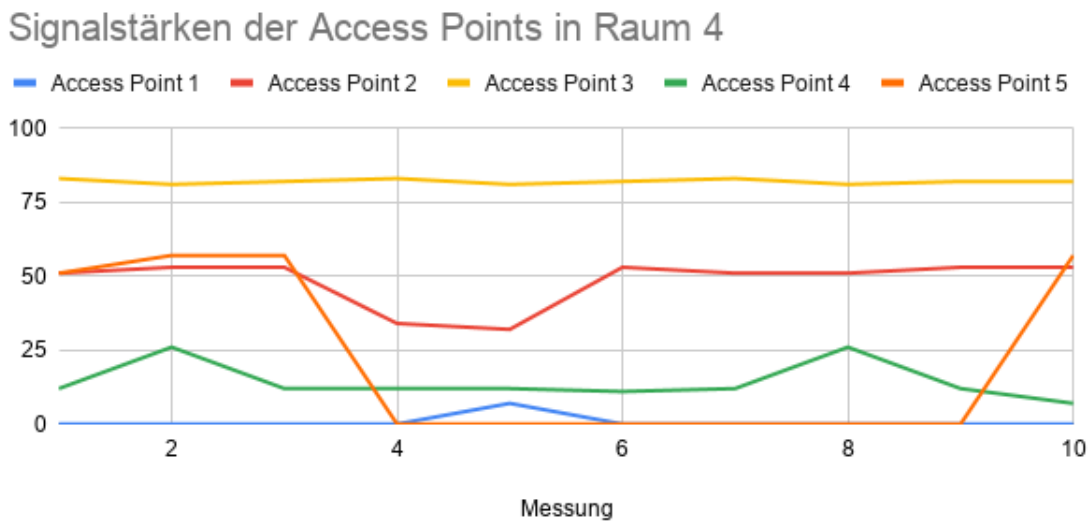


Abbildung B.6: Signalstärken der Access Points in Raum 4 über zehn Messungen

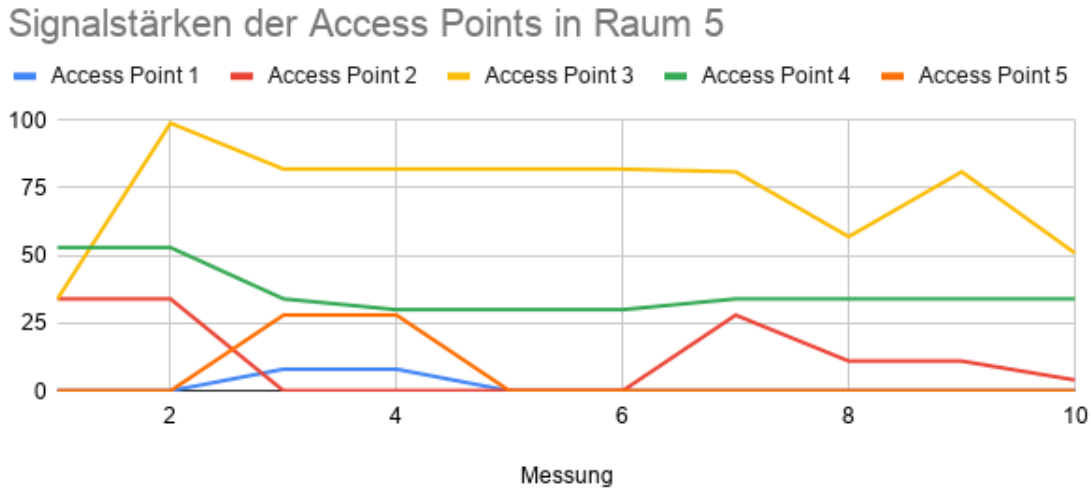


Abbildung B.7: Signalstärken der Access Points in Raum 5 über zehn Messungen

Anhang C

Aufbau der zugehörigen ZIP-Datei

C.1 Literatur

Im Ordner Literatur befinden sich alle im Literaturverzeichnis angegebenen Quellen entsprechend ihres Kürzels im Literaturverzeichnis. Dort befindet sich zusätzlich der Ordner Fußnoten, der alle Fußnoten enthält.

C.2 Abbildungen

Alle in dieser Bachelorarbeit verwendeten Abbildungen befinden sich im Ordner Abbildungen.

C.3 Quellcode

Jeglicher Quellcode, der für diese Bachelorarbeit genutzt wurde und hier als Dateinamen angegeben wurde, befindet sich

C.4 Videos

Die Videos sind aufgrund ihrer Größe zunächst nur per Link verfügbar. Der Upload dauerte schlicht zu lange, da der Autor dieser Arbeit vergaß, dass es sehr aufwendig sein kann Videos in 4k-Qualität hochzuladen. Der Link ist dieser https://photos.app.goo.gl/hQA5Rq4iH64jiy4r8. Komprimierte Dateien werden später im Stud.IP bereit gestellt.

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, den 2020-04-24