Bernburg
Dessau
Köthen

Hochschule Anhalt
Anhalt University of Applied Sciences

emw
Fachbereich
Elektrotechnik, Maschinenbau
und Wirtschaftsingenieurwesen

Masterarbeit

zur Erlangung des akademischen Grades Master of Engineering (M. Eng.)

Felix Stange

Vorname Nachname

Master Elektro- und Informationstechnik, 2016, 4056379

Studiengang, Matrikel, Matrikelnummer

Thema:
Untersuchungen und Implementierung eines automatisierten Fahrens mittels Pololu Zumo

Prof. Dr. Ingo Chmielewski

Vorsitzender der Masterprüfungskommission und 1.Prüfer

Prof. Dr. Michael Brutscheck

2. Prüfer

30.05.2018

Abgabe am

Abstract

Die vorliegende Masterarbeit dokumentiert Untersuchungen zum autonomen Fahren und beschreibt die Implementierung für ein Arduino-gestütztes Roboterfahrzeug. Dabei werden zunächst allgemeine Fragen und Aspekte zum autonomen Fahren bearbeitet, bevor auf das Roboterfahrzeug „Pololu Zumo 32U4“ eingegangen wird. Die Arduino-Plattform und das Fahrzeug werden detailliert beschrieben und die Hard- und Software-seitigen Merkmale herausgearbeitet. Für die Erfüllung der Aufgaben (Spurhalten, Abbiegen, Überholen) wurde eine passende Fahrstrecke angefertigt, die in der Arbeit vorgestellt wird. Danach wird die resultierende Programmierung mit allen Unterfunktionen ausführlich beschrieben. Entsprechend dem Fokus auf die Programmierung bildet diese auch den Großteil der Arbeit. Abschließend werden die Probleme, die während der Bearbeitung auftraten, erklärt und eine Einschätzung des Pololu Zumo abgegeben. Somit ist die Masterarbeit lohnenswert für alle Interessierten des Themas „Autonomes Fahren“ sowie der Arduino-Programmierung.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die Arbeit selbstständig verfasst, in gleicher oder ähnlicher Fassung noch nicht in einem anderen Studiengang als Prüfungsleistung vorgelegt wurde und keine anderen als die angegebenen Hilfsmittel und Quellen, einschließlich der angegebenen oder beschriebenen Software, verwendet wurden.

_____ _____

Ort, Datum Unterschrift des Verfassers

Sperrvermerk

Sperrvermerk: ja ☐ nein ☐

Wenn ja: Der Inhalt dieser Arbeit darf Dritten ohne Genehmigung des Unternehmens _____ nicht zugänglich gemacht werden. Dieser Sperrvermerk gilt für die Dauer von ___ Jahren.

_____ _____

Ort, Datum Unterschrift des Betreuers

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ABS	Antiblockiersystem
CSR	Cambridge Silicon Radio
ESP	Elektronisches Stabilitätsprogramm
I²C	Inter-Integrated Circuit
IDE	Integrierte Entwicklungsumgebung
IR	Infrarot
ISM	Industrial, Scientific, Medical Band
ISP	In-System-Programmierung
GND	Masse
GPS	Globales Positionsbestimmungssystem
HF	Hochfrequenz
LCD	Flüssigkristallanzeige
LED	Leuchtdiode
PAN	Privates Netzwerk
PWM	Pulsweitenmodulation
RAA	Richtlinie für die Anlage von Autobahnen
SAE	Verband der Automobilingenieure
USB	Universelles serielles Bussystem

1 Einleitung

Der Begriff Automobil besagt bereits, dass sich das Fahrzeug ohne äußerlich wirkende Kraft von Menschen oder Tieren fortbewegt. Durch die zunehmende Elektronikdichte in Fahrzeugen erfährt der Begriff Autonomie eine Wandlung. Die Vielzahl an verbauten Sensoren, Aktoren und Prozessoren machen nicht den Fahrer autonom, sondern das Fahrzeug selbst. Nachdem sich das Fahrzeug von dem Pferdegespann befreit hat, wird es nun unabhängig vom Fahrer. Viele große und kleine Unternehmen beschäftigen sich intensiv mit diesem neuen Markt. Bis zum vollautomatisierten Automobil auf öffentlichen Straßen ist es in vielerlei Hinsicht noch ein langer Weg.

Die Untersuchung des Autonomiebegriffs erfolgt überwiegend unabhängig von konkreten Fahrzeugen. Für die Implementierung eines autonomen Systems wird in dieser Arbeit das Roboterfahrzeug „Zumo 32U4“ der Firma Pololu [1] genutzt. Der Zumo bietet mit seinem Arduino-Mikrocontroller sowohl Hardware- als auch Software-seitig eine geeignete Umgebung für die Programmierung. Es wird ein Programm für den Zumo erstellt, das den Zumo befähigt mit Hilfe seiner vorinstallierten Sensoren autonom auf einem vorgegebenen Kurs zu fahren. Der Kurs sieht eine zweispurige Straße mit gegenläufigem Verkehr vor und bietet Möglichkeiten zum Abbiegen und Überholen. Entsprechend wird der Zumo befähigt Fahrbahnmarkierungen zu erkennen, um die Spur zu halten und Kreuzungen und Überholstrecken zu nutzen. Außerdem kann das Testfahrzeug Abstände messen, um seine Umgebung zu kontrollieren und Kollisionen mit Hindernisse zu vermeiden. Zusätzlich wird eine Kommunikation mit einem anderen bau- und programmgleichen Fahrzeug aufgebaut, das sich auf derselben Strecke bewegt.

Das genutzte Roboterfahrzeug stellt eine gute Möglichkeit dar ein Thema, das normalerweise große Fahrzeuge betrifft, auf kleinem Raum und unter Laborbedingungen zu bearbeiten. Andererseits kann die erstellte Lösung nicht ohne Umstände auf diese Fahrzeuge übertragen werden.

Zunächst werden die gesellschaftlichen, rechtlichen und technischen Aspekte zu diesem Thema abgehandelt. Dabei werden auch einige Projekte von bekannten Unternehmen beschrieben, die bereits direkt den öffentlichen Straßenverkehr betreffen. Anschließend wird die Hardware vorgestellt, Arduino im Allgemeinen und der Zumo mit seiner Vielzahl an Sensoren und weiteren Bauteilen im Konkreten. Außerdem werden die Unterschiede zwischen dem Laboraufbau und den öffentlichen Straßen erläutert. Abschließend wird die Programmierung detailliert mit allen Funktionen erklärt.

2 Stand von Wissenschaft und Technik

In diesem Kapitel wird zunächst auf das Thema „Autonomes Fahren“ eingegangen. Anschließend wird die Arduino-Plattform mit ihrem Zusammenspiel von Hardware und Software beschrieben.

2.1 Das autonome Fahren

Zum Begriff des Autonomen Fahrens findet man je nach Quelle verschiedene Definitionen. Im Kern aller Erklärungsversuche steht jedoch immer das Fahrzeug, welches sich selbstständig und zielgerichtet ohne Eingreifen eines Fahrers bewegt.
In diesem Zusammenhang findet im Folgenden eine Auseinandersetzung mit den rechtlichen und gesellschaftlichen Aspekten statt und es werden die Klassifizierungsstufen und bekannte Projekte vorgestellt.

2.1.1 Rechtliche und gesellschaftliche Aspekte

Das Thema des Autonomen Fahrens wird teilweise sehr engagiert in der Öffentlichkeit beworben aber auch diskutiert. Insbesondere Automobilhersteller aber auch Unternehmen aus anderen Geschäftsbereichen werben mit Erfolgsmeldungen ihrer Projekte. Wohingegen die öffentliche Meinung alle Formen zwischen völliger Ablehnung und Begeisterung annehmen kann.
Der Mensch ist es gewohnt sein Fahrzeug selbst zu steuern und fühlt sich teilweise sogar in seiner Ehre verletzt, wenn er das Lenkrad aus welchem Grund auch immer abgeben muss. In ein selbstfahrendes Auto, das sich ohne sein Zutun den Weg durch den Verkehr bahnt, steigt man nur mit Unbehagen oder sogar Angst ein. Die neuen Techniken sind fremd und man vertraut ihnen nicht die richtigen Entscheidungen zu treffen. Hilfssysteme wie das Antiblockiersystem (ABS) und das Elektronische Stabilitätsprogramm (ESP) wurden noch schnell akzeptiert und sind in normaler Fahrsituation auch nicht bemerkbar, da sie nur in Notfällen eingreifen. Der drohende Verlust der Freiheit gefällt vielen Menschen jedoch gar nicht. Immer mehr Systeme werden gesetzlich vorgeschrieben. Seit März 2018 ist das Notrufsystem inklusive Ortung mit dem Globalen Positionsbestimmungssystem (GPS) in Neufahrzeugen Pflicht. Dies betrifft nicht das autonome Fahren, geht aber einher mit dem Verlust der besagten Freiheit. Sofort stehen dann auch ethische Grundfragen im Raum, um

den Schutz persönlicher Daten und die Sicherheit auf den Straßen zu gewährleisten. Dazu hat im Sommer eine Ethikkommission um den Verfassungsrechtler Udo Di Fabio zwanzig Regeln aufgestellt [2]. Darunter findet sich die wesentliche Vorgabe, dass Autos in gefährlichen Situationen nicht nach Alter, Geschlecht oder Herkunft entscheiden sollten. Diese Herausforderungen müssen und werden die Entwickler lösen, sonst setzt sich die Technik nicht durch. Welche Strecke sie bereits auf dem Weg dorthin bewältigt haben, lässt sich am besten an dem sechsstufigen Modell der internationalen Ingenieurs- und Automobilindustrievereinigung (SAE) zeigen, das im folgenden Kapitel erklärt wird.
Andere Menschen können den steigenden Grad der Autonomie ihrer Fahrzeuge kaum abwarten. Technikfans sind begeistert, welche neuen Möglichkeiten sich ergeben. Die Vielzahl der einzelnen Hilfssysteme ist groß, egal ob sie der Sicherheit oder dem Komfort dienen. Insbesondere Vielfahrer haben sich sicherlich über den ersten Tempomat gefreut. Heute kommen für die Fahrt auf der Autobahn Spurhalteassistent und Abstandsregeltempomat hinzu. Aber auch für die Fahrt im städtischen Verkehr kommen immer mehr Systeme, wie die Notbremsfunktion oder der Einparkautomat, hinzu. Ob die Systeme genutzt werden, ist stark von der Persönlichkeit der Fahrer abhängig.
Entgegen der gesetzlichen Vorschrift Systeme ins Fahrzeug einzubauen, weigert sich der Gesetzgeber im gleichen Zug das Fahrzeug gänzlich sich selbst zu überlassen. Während die Automobilhersteller immer mehr Systeme als Extra in ihren Fahrzeugen anbieten wollen, werden sie auch in der Zulassung der Systeme im öffentlichen Straßenverkehr zunächst gebremst. Ein System muss sich stets erst in seiner Sicherheit und Zuverlässigkeit beweisen. Gleiches gilt auch für die Prototypen der forschenden Unternehmen. Angeblich sind Lenkrad und Pedale dort bereits überflüssig und dadurch entsprechend auch der Fahrer, für den Gesetzgeber sind diese Komponenten aber noch essentiell für den sicheren Verkehr auf den Straßen [3].

2.1.2 Klassifizierung – vom Assistenzsystem zur Vollautomatisierung

Die Autonomie von Fahrzeugen im Straßenverkehr wird in sechs Stufen unterteilt. Die Klassifizierung der Autonomiestufen ist in der Norm SAE J3016 beschrieben. Die Norm wurde von der Organisation für Mobilitätstechnologie herausgegeben und von der Bundesanstalt für Straßenwesen größtenteils übernommen. Im folgenden Text werden die Stufen detailliert erklärt. In Abbildung 1 sind die Stufen der Automatisierung übersichtlich dargestellt.

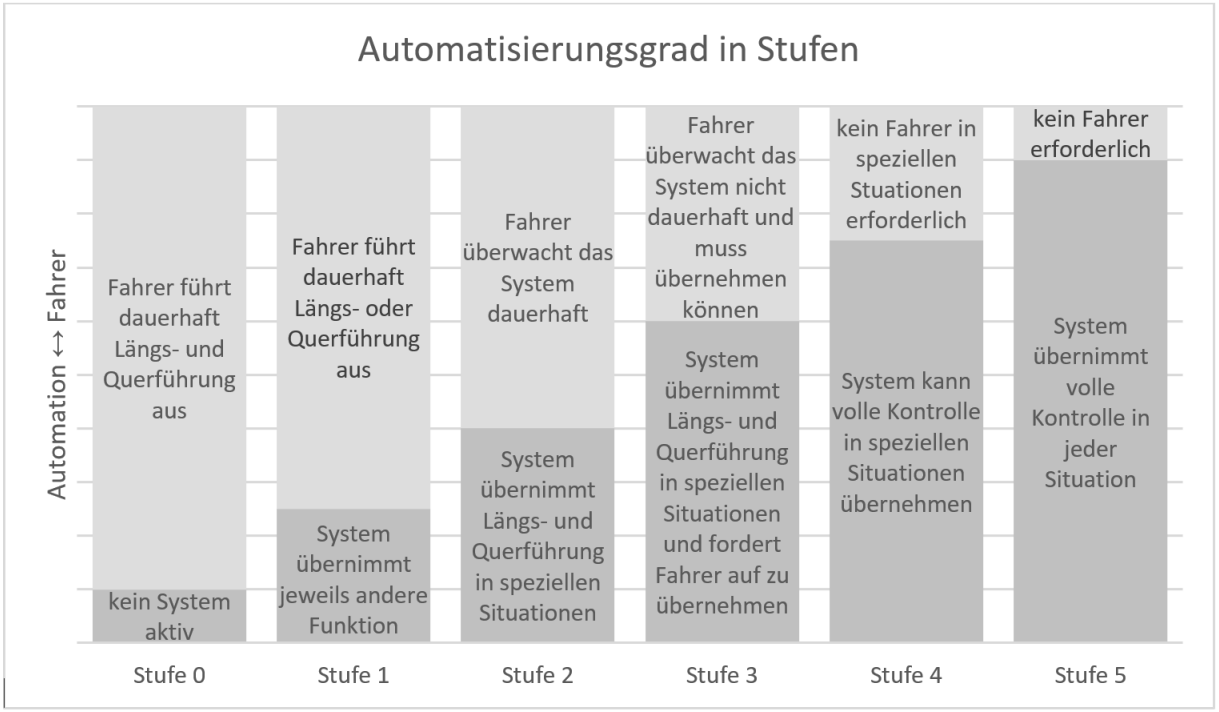


Abbildung 1: Automatisierungsgrad in Stufen nach SAE J3016 [4]

Stufe 0 sieht keine Automatisierung vor. Der Fahrer fährt eigenständig (lenken, beschleunigen, bremsen usw.), wobei er von Systemen wie ABS und ESP unterstützt werden kann.

Stufe 1 beinhaltet die Nutzung von Assistenzsystemen. Die Fahrassistenzsysteme helfen bei der Fahrzeugbedienung bei Längs- oder Querführung z.B. mit Abstandsregeltempomat.

Stufe 2 bedeutet eine „Teilautomatisierung". Hier helfen Fahrassistenzsysteme situationsbedingt bei der Fahrzeugbedienung sowohl bei Längs- als auch bei der Querführung. Beispiele sind die Einparkautomatik, der Spurhalteassistent oder der Stauassistent. Auf die Gegebenheiten der Umgebung muss der Fahrer noch selbst reagieren. Fahrzeuge bis Stufe 2 sind bereits allgegenwärtig und bewegen sich in einer Vielzahl auf den Straßen.

In **Stufe 3**, der „Bedingten Automatisierung", wird das Fahrzeug längere Zeit vollständig vom System geführt und der Fahrer muss das System nicht ständig überwachen. Nur auf Anforderung zum Eingreifen muss der Fahrer innerhalb einer Vorwarnzeit reagieren.

Die vorletzte **Stufe (4)** beschreibt die „Hochautomatisierung". Das Fahrzeug wird dauerhaft autonom vom System geführt. Sollte das System die Fahraufgabe nicht mehr bewältigen können, kann der Fahrer aufgefordert werden einzugreifen.

Die höchste **Stufe (5)** der Automatisierung ist die „Vollautomatisierung“. Das Fahrzeug wird vollständig autonom geführt, auf jeder Fahrbahn und bei jeder Umgebungsbedingung. Es ist kein Fahrer und somit auch kein Eingreifen mehr notwendig. Der Mensch legt nur noch das Ziel fest und startet das System. Er ist nur noch Mitfahrer [4].

Der Übergang von Stufe 2 auf 3 bildet momentan die Grenze zwischen dem was rechtlich und gesellschaftlich akzeptiert wird. Diese Stufe ist auf Autobahnen bereits technisch machbar (Blinken, Spurwechsel, Spurhalten usw.) und der Gesetzgeber ist bestrebt Fahrzeuge der Stufe 3 zuzulassen. Die Algorithmen sind heute allerdings noch schlicht. Sie decken die Entwicklungsstufe 3 ab und müssen noch viel lernen. Daran arbeiten Autoindustrie und Technologiekonzerne. Für den serienmäßigen Sprung auf die Stufe 4 gilt es, weitere Hürden aus dem Weg zu räumen. So ist es nötig, einen einheitlichen Standard für die Kommunikation zwischen den Fahrzeugen zu schaffen. Fahrzeuge tauschen hierbei im Vorbeifahren relevante Informationen über Verkehr, Straßenzustand, Witterungsverhältnisse und freie Parkplätze aus. Oder sie informieren andere Pkw, die ebenso auf eine Kreuzung zufahren, ohne dass eine direkte Sichtverbindung besteht. Im nächsten Schritt muss zudem die Kommunikation mit der Umwelt realisiert und verstärkt an hochauflösenden Karten gearbeitet werden. In den höchsten Stufen kommen Bereiche wie neuronale Netze, „Deep Learning“ und künstliche Intelligenz hinzu.

2.1.3 Bekannte Projekte

Mercedes-Benz Future Truck 2025

Abbildung 2: Mercedes-Benz Future Truck 2025 auf der A14 [5]

Auf einem Teilabschnitt der A14 bei Magdeburg bestand der Future Truck von Mercedes-Benz, zu sehen in Abbildung 2, am 3.Juli 2014 seine Feuertaufe. Ein Actros mit dem Assistenzsystempaket „Highway Pilot“ fuhr bei realistischen Fahrsituationen vollkommen autonom. Damit will Mercedes-Benz sowohl die Effizienz erhöhen, als auch für mehr Sicherheit auf deutschen Autobahnen sorgen. Sobald der Fahrer den LKW auf eine Autobahn gesteuert und die rechte Fahrspur eingenommen hat, kann der Highway Pilot aktiviert werden. Bei Baustellen, schlechtem Wetter und anderen Behinderungen warnt das System den Fahrer, der wieder das Steuer übernimmt. Um selbstständig fahren zu können, kombiniert das Assistenzsystempaket Abstandsregeltempomat, Bremsassistent, Stabilitätsassistent, Spurhalteassistent und den vorausschauenden Tempomat, der Informationen über Topologie und Streckenverlauf nutzt. Bis zum Jahr 2025 will man den Future Truck bzw. den Highway Pilot bis zur Serienreife entwickelt haben, wenn die Rahmenbedingungen stimmen [6].

Alphabet/Google Waymo

Abbildung 3: Waymos autonom fahrender Chrysler Pacifica [7]

2014 stellte Google seine autonomen Fahrzeuge vor und ließ sie drei Jahre über die Straßen von Kalifornien und Texas fahren. Die Prototypen mit dem Namen „Firefly“ boten für den Fahrer nur wenig Komfort und fuhren lediglich mit einer Geschwindigkeit von 40 Kilometern pro Stunde. Nach den Plänen des Herstellers sollten selbst das Lenkrad und die Pedale in späteren Versionen entfallen, jedoch wurde dieses Vorhaben von den Behörden untersagt. Trotzdem war es den „Fireflies“ möglich sich vollständig autonom auf den Straßen zu bewegen. Angetrieben wurden die Fahrzeuge ausschließlich elektrisch.

Mittlerweile ist die Testphase beendet und die kleinen Zweisitzer sind von den Straßen verschwunden. Nach der Neustrukturierung Googles bietet Alphabets Tochterfirma Waymo die gewonnen Erkenntnisse den Autobauern an. Demnächst führt Fiat-Chrysler eigene Testwagen vor, die Googles Technologie verwenden. Familien in Arizona können die Chrysler Pacifica, zu sehen in Abbildung 3, testweise im Alltag nutzen. Waymo selbst führt seine Tests derweil mit Lastkraftwagen fort, die bereits durch die öffentlichen Straßen der US-Ostküstenstadt Atlanta fahren und dort Frachten transportieren [8].

Tesla Model S/X/3

Abbildung 4: Tesla Model S [9]

Der bekannte Elektroautomobil-Hersteller Tesla baut seine Fahrzeuge, in Abbildung 4 ein Model S, bereits mit der benötigten Hardware für autonomes Fahren und bewirbt es auch groß auf der eigenen Internetseite. Dazu nutzt Tesla mehrere Sensorsysteme mit komplexer Steuerung und hoher Rechenleistung. Acht Kameras bieten eine Rundumüberwachung. Hinzu kommen zwölf Ultraschallsensoren und ein vorwärts gerichtetes Radar, um selbst bei schlechten Wetterverhältnissen sicher zu navigieren.
Das Werbevideo vermittelt den Eindruck, dass man ein vollautomatisiertes Fahrzeug erwarten kann, in Worten findet man jedoch eine deutlich weniger umfangreiche Funktionsbeschreibung. Die „verbesserte Autopilot-Funktionalität" bietet umfassende Funktionen auf der Autobahn (Geschwindigkeitsregeltempomat, Spurhalteautomatik, Spurwechselautomatik, Autobahnabfahrtswarner) und beim Einparken (Parkplatzsuche und Einparkautomatik). Grund dafür sind rechtliche Bestimmungen. Zwar ist das autonome Fahren mit einem Tesla bereits möglich, jedoch darf der Hersteller die Funktionen nur so weit implementieren wie es vom Gesetzgeber zugelassen wird. Man verspricht durch Updates künftig weitere Funktionen nachzureichen, sobald die rechtlichen Genehmigungen eingeholt wurden [10].

2.2 Kurzvorstellung der Arduino-Plattform

Unter Arduino versteht man üblicherweise vorgefertigte Mikrocontroller-Boards, die mithilfe einer bereitgestellten Programmierumgebung genutzt werden können. Da die Hardwaredesign-Informationen verfügbar sind, können die Boards auch in Eigenproduktion hergestellt und personalisiert werden. Die Arduino-Boards sind simple Input-Output-Mikrocontroller, die eine Processing- bzw. Wiring-Programmiersprache nutzen. Die Kombination aus Hardware und Software stellt ein unabhängiges Mikrocomputersystem dar. Arduino ist ein Open Source und Open Hardware Projekt. Sowohl die Hardwaredesigns als auch der Quellcode sind verfügbar. Kommerzielle Produkte von Fremdherstellern nutzen häufig die Endung „duino", jedoch nicht den vollen Namen Arduino.

2.2.1 Hardware

Der Kern eines Arduino-Boards ist ein Atmel AVR Mikrocontroller (z.B. ATmega32U4), der von zusätzlicher Hardware unterstützt wird. Dazu zählen mindestens ein 5V-Linearspannungsregulator und ein 16 MHz-Oszillator. Durch einen vorprogrammierten Bootloader ist kein externer Programmer notwendig. Aktuelle Boards werden über einen Anschluss am Universellen-seriellen-Bussystem (USB) programmiert, der mithilfe eines USB-auf-Seriell-Adapterchips realisiert wird.
Die meisten vorhandenen Input-/Output-Pins des Controllers werden anderen Schaltkreisen verfügbar gemacht. Ein Arduino Leonardo, das beispielhaft in Abbildung 5 zu sehen ist, hat 20 digitale Pins. Davon können sieben als Pulsweitenmodulation (PWM)-fähige Ausgänge und 12 als analoge Eingänge genutzt werden. Diese Pins befinden sich auf der Oberseite des Boards und sind als Buchsen im 0,1 Zoll Format gebaut. Um vorgefertigte Boards zu nutzen und dennoch auf die Anwendung optimal zuzuschneiden, werden sogenannte Shields gefertigt. Diese Plug-In-Boards werden passend zu den Buchsen auf dem Arduino-Board mit Header-Pins auf der Unterseite gefertigt und können lötfrei aufgesteckt werden. Als Alternative zu den Shields existieren die sogenannten Breakout-Boards, welche nicht an die Dimensionen der Arduino-Platine gebunden sind. Diese Erweiterungsplatinen bieten oft Zugang zu einem einzelnen Schaltkreis (IC) oder einzelnen Komponenten (Kamera, Sensor usw.) [11].

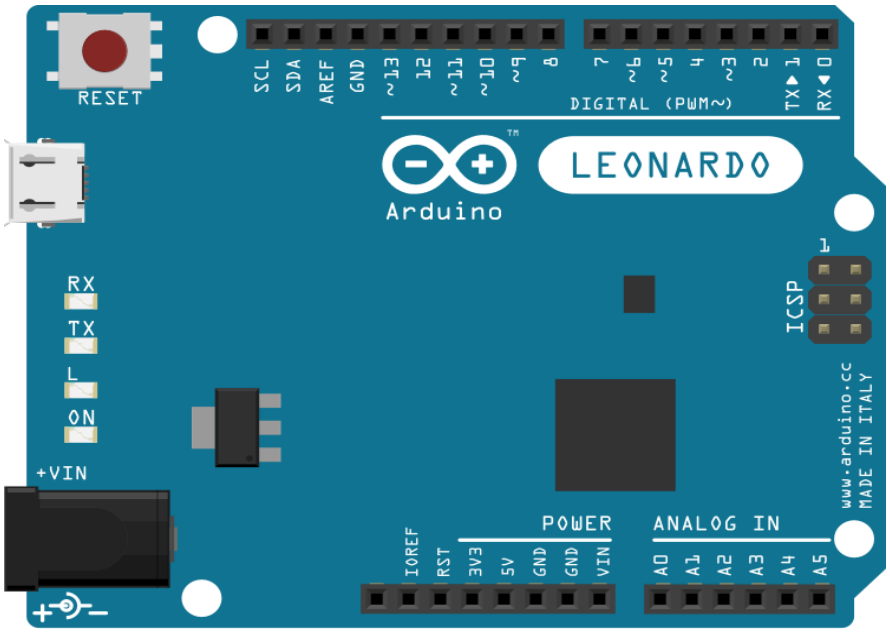


Abbildung 5: Arduino Leonardo (erstellt mit Fritzing)

2.2.2 Software

Zur Programmierung der Boards können verschiedene Programme verwendet werden, z.B. die Integrierte Entwicklungsumgebung (IDE) von Arduino. Dabei handelt es sich um eine Java-Cross-Plattform, die einen Programmcode-Editor und einen Compiler bereitstellt. Außerdem ist es möglich Firmware seriell zum Board zu übertragen. Die Programmierung selbst erfolgt in einer C-ähnlichen, vereinfachten Sprache.
Der Programmcode besteht aus zwei Teilen, die die Anweisungen umschließen. Die Funktion setup() wird einmalig beim Programmstart aufgerufen und muss vor dem anderen Teil ausgeführt werden. Sie beinhaltet Variablendefinitionen, Pinzuweisungen und die Initiierung der seriellen Kommunikation. Anschließend folgt die Funktion loop(), die in einer endlosen Schleife ausgeführt wird. Sie enthält den eigentlichen Programmablauf [12].

2.3 Filterung der Messwerte mittels Medianfilter

Um Fehlmessungen von Sensoren auszusortieren eignen sich Rangordnungsfilter. Diese gehören zu den nichtlinearen Filtern und können nicht durch eine Faltung beschrieben werden. Zu den Rangordnungsfiltern gehören neben dem Medianfilter, je nach Reihenfolge in der Filterliste, auch der Minimum- und der Maximumfilter.

Bei jedem neuen Durchlauf der Sensorfunktionen werden dem Filterfenster neue Messwerte hinzugefügt. Die ältesten Messwerte werden jeweils in gleicher Anzahl entfernt. Der Filter ordnet die neue Wertereihe der Größe nach und gibt den mittleren Wert des Fensters zurück.

$$Median\{128, 130, 134, 141, \mathbf{678}\} = 134 \quad (1)$$

So werden Ausreißer, also besonders große oder kleine Messwerte, die nur vereinzelt auftreten, ignoriert. Kommt es wirklich zu einer Wertänderung der Sensoren, werden diese erst nach wiederholtem Durchlauf erkannt, wenn die Werte das Fenster allmählich ausfüllen [13]. Beispielsweise sind bei einem Filter mit einer Fenstergröße von fünf Werten mindestens drei Durchläufe nötig, um auf die neuen Werte zu reagieren.

$$Median\{130, 141, \mathbf{656}, \mathbf{678}, \mathbf{689}\} = \mathbf{656} \quad (2)$$

Bei Mittelwertberechnungen werden Messfehler auch bei einmaligem Auftreten berücksichtigt und verfälschen die Messung.

$$Mittelwert\{128, 130, 134, 141, \mathbf{678}\} = \mathbf{242,2} \quad (3)$$

3 Vorstellung der Hardware – Pololu Zumo 32U4

Der Zumo 32U4 von der Firma Pololu ist ein kleiner vorgefertigter Roboter, siehe Abbildung 6. Er misst in Länge sowie Breite jeweils weniger als 10 cm. An der Front besitzt er ein Schild, das die Sensoren sowohl mechanisch als auch technisch abschirmt. Gesteuert wird er von einem Arduino-kompatiblen ATmega32U4-Mikrocontroller. Zu dem Mikrocontroller besitzt er Motoren und verschiedene Sensoren. Hinzu kommen Leuchtdioden (LED), eine Flüssigkristallanzeige (LCD) und ein Summer für die Datenausgabe und Taster für Benutzereingaben. Zur Programmierung steht ein USB-Anschluss bereit, der über die bekannte Arduino IDE genutzt werden kann. Da der Zumo kein Produkt von Arduino ist, sondern nur auf dessen Plattform basiert, ist die Installation zusätzlicher Treiber und Bibliotheken von Pololu nötig [1]. In den folgenden Kapiteln wird genauer auf die einzelnen Bestandteile, insbesondere die Sensoren, und das Bluetooth-Modul eingegangen.

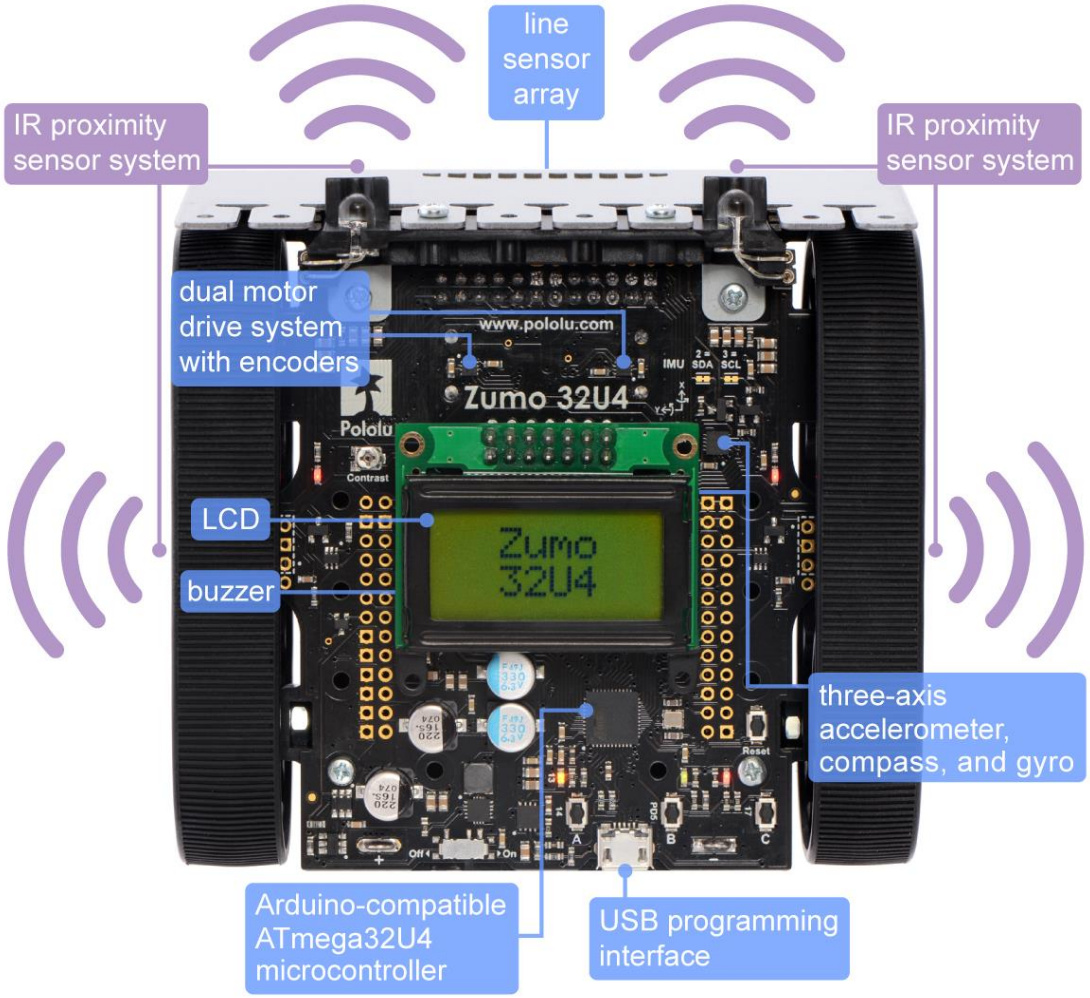


Abbildung 6: Ausstattung des Pololu Zumo 32U4 [1, p. 3]

3.1 Grundlegende Bauteile

Neben den Sensoren besitzt der Zumo weitere essentielle Bauteile. Motoren ermöglichen die Bewegung. Ein Mikrocontroller verarbeitet Daten und steuert das Fahrzeug. Außerdem stehen Möglichkeiten zur Interaktion bereit.

3.2.1 Mikrocontroller

Der ATmega32U4 (Abbildung 7) befindet sich auf der Zumo-Hauptplatine. Dieser AVR 8 bit-Mikrocontroller wird von einem Kristalloszillator mit einer Präzision von 16 MHz getaktet. Der Mikrocontroller und die Taktfrequenz werden auch für den Arduino Leonardo und Micro genutzt, sodass Programmcode universell für diese drei Boards geschrieben werden kann. Dem 32U4 stehen 32 KB In-System-Programm (ISP)-Flashspeicher zur Seite. Über den USB-Micro-B-Anschluss kann eine Verbindung zu einem Computer hergestellt werden. Über diese Verbindung können Daten übertragen, sowie der Zumo mit Energie versorgt und programmiert werden. Durch den vorinstallierten Bootloader ist es möglich den Zumo, wie andere Boards, mithilfe der Arduino IDE zu programmieren [11].

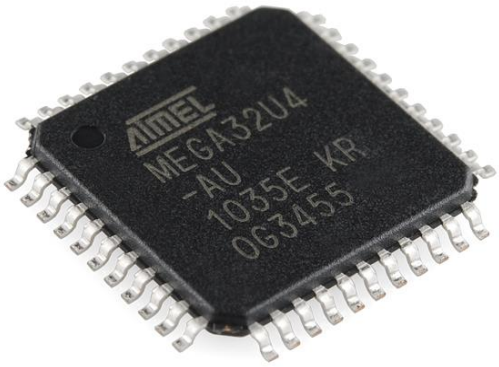


Abbildung 7: Mikrocontroller ATmega32U4 [11]

3.2.2 LEDs

Der Zumo besitzt insgesamt acht LEDs auf der Hauptplatine, davon sind drei (rot, gelb und grün) frei programmierbar. Diese drei LEDs blinken beim Hochladen eines Sketches bzw. beim Datenaustausch über USB, sind zum Programmstart aber wieder verfügbar. Die Anderen sind für spezielle Vorgänge vorgesehen. Zwei Rote zeigen die Aktivität der Infrarotemitter (IR) an.

Zwei Blaue teilen dem Nutzer mit, dass über die Batterien Spannung anliegt und der Zumo eingeschaltet ist. Eine weitere grüne LED leuchtet, wenn der Zumo über den USB-Anschluss mit Spannung versorgt wird [1, p. 12].

3.2.3 Taster

Von den insgesamt vier Tastern auf der Hauptplatine sind drei frei belegbar. Diese Benutzertaster sind unterhalb der LEDs angeordnet und mit den Buchstaben A, B und C gekennzeichnet. Ein weiterer vierter Taster ist für das Zurücksetzen (Reset) des Zumo vorgesehen (Abbildung 6).

3.2.4 LCD

Zum Lieferumfang des Zumo gehört auch ein zweifarbiges LCD mit zwei Zeilen zu je acht Zeichen. Es ist mittels eines 2x7-Headers an der Oberseite der Hauptplatine verbunden und kann bei Nichtgebrauch entfernt werden. Zur Justierung des Kontrasts befindet sich ein Potentiometer auf der Platine.

3.2.5 Summer

Mit dem eingebauten Summer können einfache Töne abgespielt werden. Man kann Töne mit bestimmter Frequenz und Dauer generieren, um so sogar Musik, ähnlich den polyphonen Klingentönen alter Mobiltelefone, spielen zu können.

3.2.6 Motoren

Zwei Motorantriebe DRV8838 von Texas Instruments sind mit den Getriebemotoren des Zumo verbunden, zu sehen in Abbildung 8. Das Getriebeverhältnis beträgt 75:1. Über vier Pins können Richtung und Geschwindigkeit für die linke und die rechte Seite getrennt vorgegeben werden. Dadurch ist es möglich den Zumo auf der Stelle drehen zu lassen.

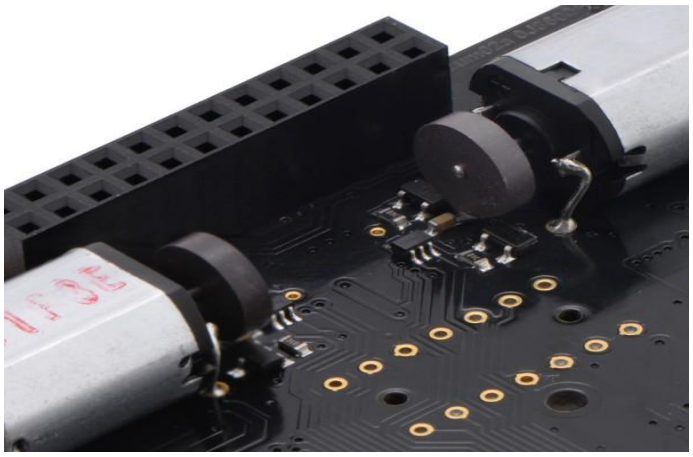


Abbildung 8: Antriebe mit Quadraturencodern [1, p. 15]

3.2 Sensoren

Der Zumo besitzt verschiedene Arten von Sensoren, dazu zählen:

- Zwei Quadraturencoder
- Fünf Liniensensoren
- Drei Abstandssensoren mit vier Sendedioden
- Ein Beschleunigungssensor in drei Dimensionen und
- Ein Drehbewegungssensor in drei Dimensionen.

In den folgenden Punkten wird näher auf die einzelnen Sensortypen eingegangen.

3.3.1 Quadraturencoder

Jeder Motorantrieb des Zumo besitzt einen eigenen Quadraturencoder. Diese bestehen aus Magnetscheiben, die sich auf der herausstehenden Motorwelle befinden, und einem Paar Halleffekt-Sensoren auf der Unterseite der Hauptplatine (Abbildung 8). Die Quadraturencoder können die Rotationsgeschwindigkeit und die Drehrichtung der Motorachse aufnehmen. Jede Umdrehung N der Motorachse entspricht zwölf Punkten der Encoder. Multipliziert man dies mit dem Getriebeverhältnis (75:1) erhält man 909,7 Punkte je Umdrehung (CPR).

$$N = 12 * 75{,}81 = 909{,}7\ CPR \tag{4}$$

3.3.2 Frontsensorplatine

Zusätzlich zur Hauptplatine besitzt der Zumo eine kleinere Platine an der unteren Front. Wie in Abbildung 9 zu sehen, befinden sich hier die fünf Liniensensoren und die drei Abstandssensoren. Von diesen acht Sensoren können nur sechs zur gleichen Zeit betrieben werden. Die Auswahl erfolgt durch Steckbrücken an der Frontsensorplatine. Es kann jeweils zwischen einem seitlichen Liniensensor und einem seitlichen Abstandssensor gewählt werden. D.h. wenn alle fünf Liniensensoren genutzt werden sollen, steht nur noch der vordere Abstandssensor zur Verfügung. Benötigt man alle drei Abstandssensoren, muss auf die Liniensensoren „DN2“ und „DN4“ verzichtet werden.

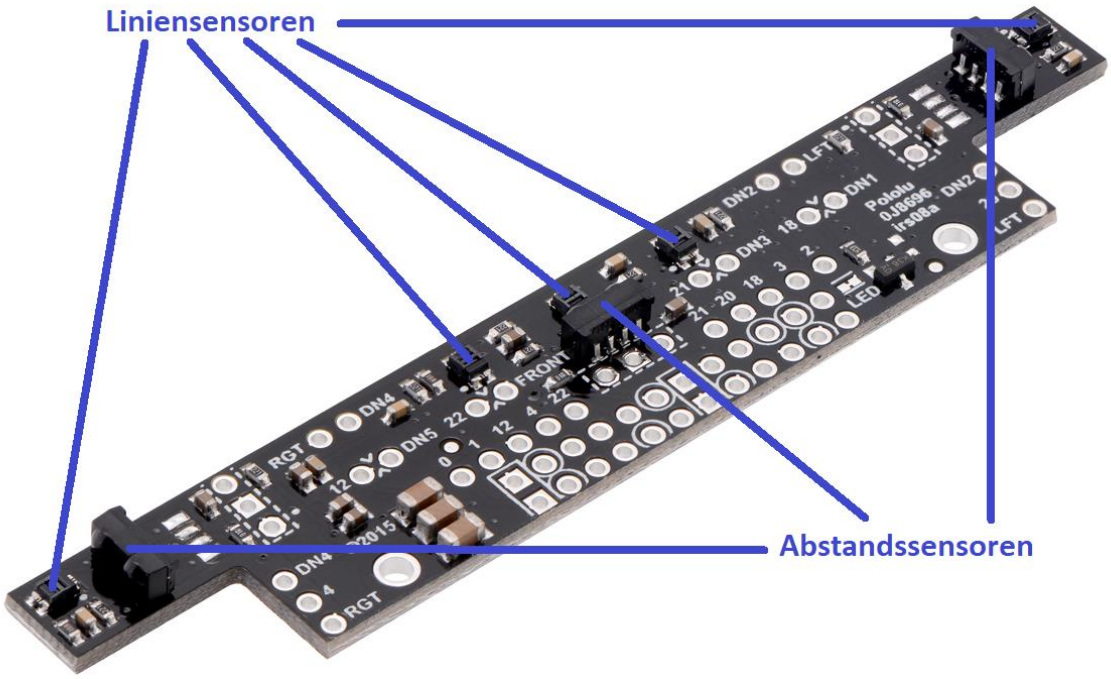


Abbildung 9: Frontsensorplatine mit Linien- und Abstandssensoren (vgl. [14])

3.3.3 Liniensensoren

Die fünf Reflexlichtsensoren sind nach unten gerichtet und unterscheiden zwischen hellen und dunklen Oberflächen. Jeder Sensor besteht aus einer Infrarot-LED als Sender und einem Fototransistor als Empfänger, die in dieselbe Richtung gerichtet sind. Die Infrarotstrahlen werden vom Boden reflektiert und zum Sensor zurückgeworfen, siehe Abbildung 10.

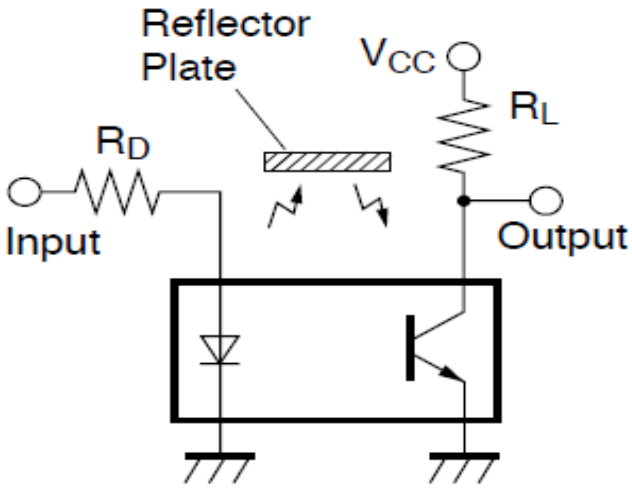


Abbildung 10: Funktionsweise eines Reflexlichtsensors [15, p. 6]

Dazu strahlen die Sender das infrarote Licht mit einer maximalen Wellenlänge von 950 nm bei einer Frequenz von 300 kHz aus. Die optimale Sensitivität des Empfängers liegt bei 930 nm, wobei der Empfangsbereich von 700 bis 1200 nm reicht. Je nach Beschaffenheit des Bodens werden die Strahlen vor der Reflektion unterschiedlich stark absorbiert. Anhand der Intensität des zurückgestrahlten Lichts ändert sich der elektrische Strom in der Empfängerdiode. Die kompakte Bauweise soll die ungestörte Sensorfunktion unterstützen, indem fälschliche Detektion von sichtbarem Licht vermindert wird. Die Liniensensoren vom Typ GP2S60 [15] stammen von der Firma Sharp. Laut Hersteller liegt der optimale Arbeitsabstand bei 0,7 mm.

3.3.4 Abstandssensoren

Die drei Abstandssensoren des Zumo überwachen verschiedene Richtungen und detektieren dabei nahe Objekte. Es ist auch möglich Befehle von einer Infrarotfernbedienung zu empfangen. Die Sensoren empfangen wie die Liniensensoren infrarotes Licht. Das Lichtsignal wird mit einer Pulsfrequenz von 38 kHz ausgesendet bzw. empfangen, um Störungen durch andere Geräte zu vermeiden. Der optimale Empfangsbereich der Sensoren liegt bei 950 nm Wellenlänge, siehe Abbildung 11.

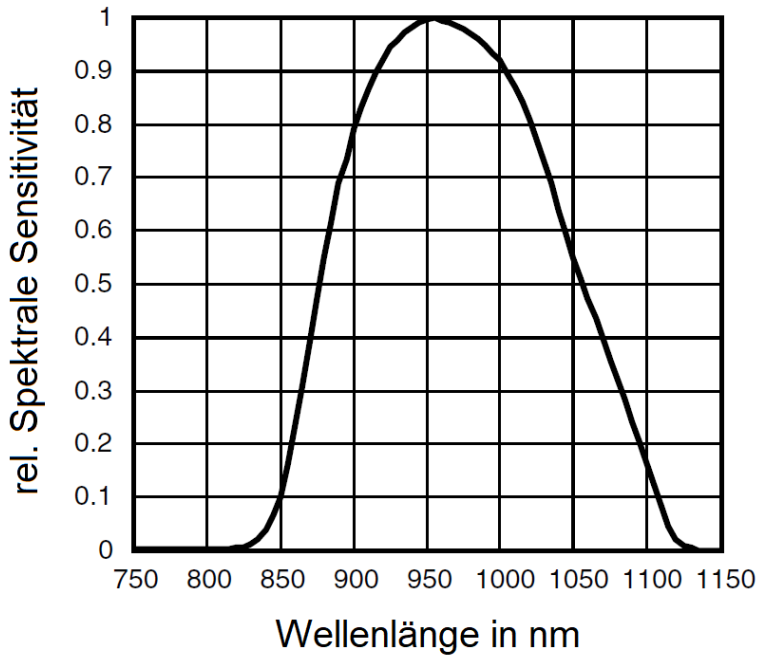


Abbildung 11: Übertragungsbereich der Abstandssensoren, vgl. [16, p. 4]

Um Beeinflussungen mit den Liniensensoren auszuschließen, werden diese von den bereitgestellten Bibliotheken im Moment der Nutzung deaktiviert. Die Empfänger werden von der Firma Vishay hergestellt, Typ TSSP77038 [16]. Aufgrund der Form des Erkennungsbereiches ist es nicht ausgeschlossen, dass Hindernisse in bestimmten seitlichen Positionen erst spät erkannt werden. Wie in Abbildung 12 dargestellt, werden bei einem Winkel von ±50° Objekte nur noch bis zur Hälfte der Übertragungsstrecke erkannt.

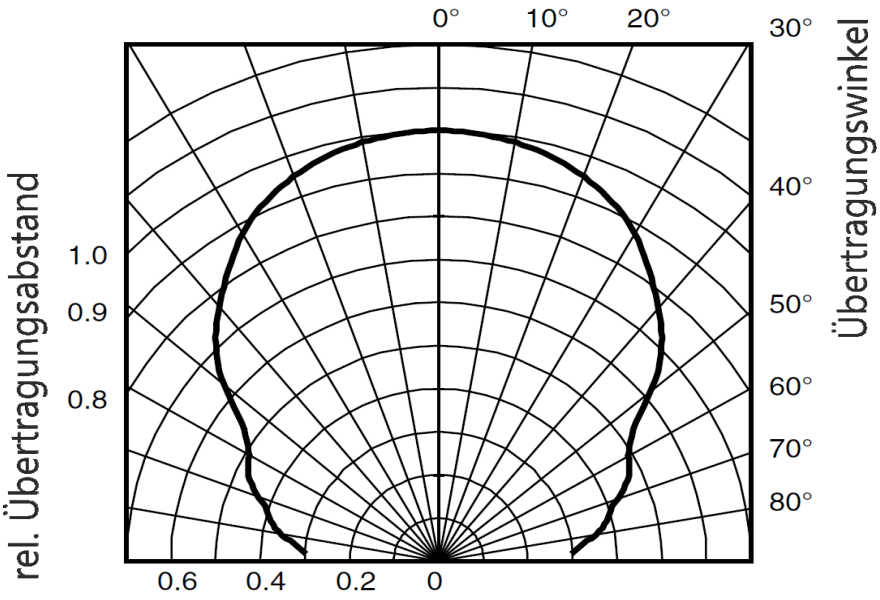


Abbildung 12: Horizontaler Erkennungsbereich der Abstandssensoren, vgl. [16, p. 4]

Die Sensoren besitzen keine direkt verbauten Infrarotsender. Der Zumo hält für die Abstandssensoren vier extra LEDs bereit, die Infrarotes Licht (940 nm) mit der benötigten Frequenz ausstrahlen. Zwei LEDs befinden sich an der vorderen Seite der Hauptplatine (Abbildung 13) und jeweils eine weitere LED an der rechten bzw. linken Seite der Hauptplatine zwischen den Rädern (Abbildung 14).

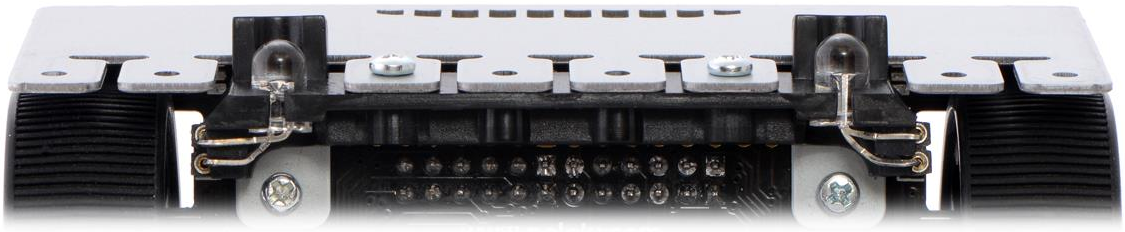


Abbildung 13: Infrarot-LEDs an der Vorderseite der Hauptplatine [1, p. 20]

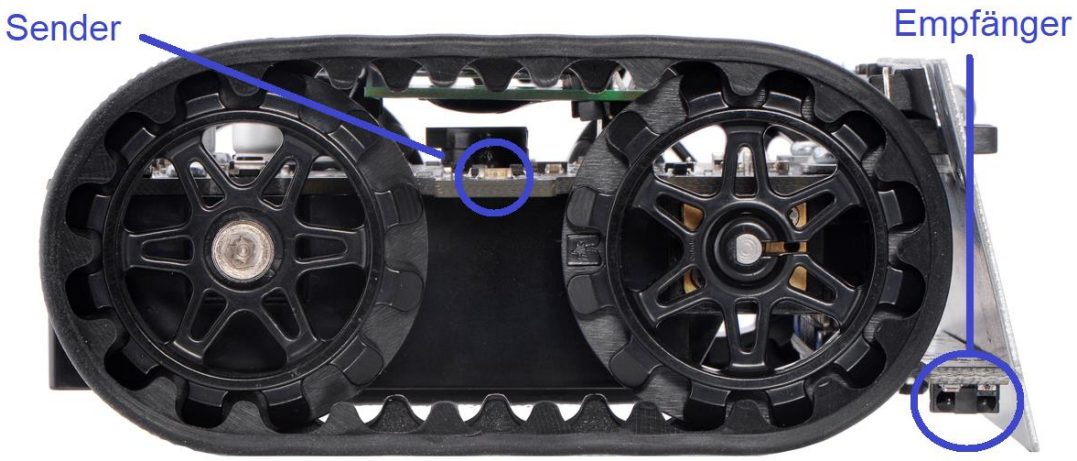


Abbildung 14: Seitenansicht des Zumo mit IR-Sender und –Empfänger, vgl. [1, p. 8]

Eine Möglichkeit der Abstandsmessung mit IR-Sensoren ist die Laufzeitmessung, anschaulich dargestellt in Abbildung 15. Dabei wird die Zeit t zwischen dem Zeitpunkt des Ausstrahlens und dem Moment des Empfangs des reflektierten Lichts gemessen und anschließend mit der Lichtgeschwindigkeit c multipliziert. Da das Licht den doppelten Weg zurückgelegt hat, muss dieser Wert noch halbiert werden.

$$d = \frac{c * t}{2} \tag{5}$$

Vorausgesetzt die Hindernisse haben die gleiche Farbe und Oberfläche kann die Distanz zum Objekt bestimmt werden. In realistischen Umgebungen haben Objekte jedoch unterschiedliche Farben und Oberflächen. Farbige Oberflächen reflektieren zudem unterschiedlich viel Licht. Während Weiß sehr gut erkannt wird, ist Schwarz nahezu unsichtbar. Wird ein IR-Signal vom Sensor empfangen, kann man davon ausgehen, dass ein Hindernis in Reichweite ist. Fehlt ein IR-Signal heißt das jedoch nicht, dass kein Hindernis vorhanden ist. Diese Problematik schränkt die Nutzung von IR-Sensoren für die Abstandsmessung stark ein.

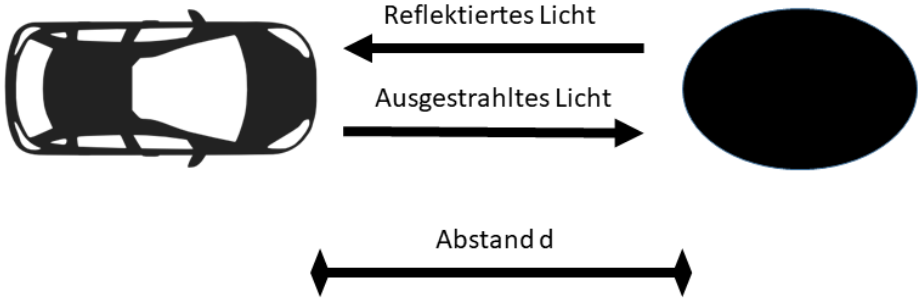


Abbildung 15: Funktionsweise der Abstandsmessung (erstellt mit Microsoft PowerPoint 2013)

3.3.5 Beschleunigungs- und Drehbewegungssensor

Der Zumo besitzt ein Paket von Inertialsensoren. Der erste Chip, ein LSM303D von ST, ist ein 3D-Kompassmodul und kombiniert einen Beschleunigungsmesser mit einem Magnetometer. Somit ist es möglich Kollisionen zu erkennen und die Messwerte zur Lagebestimmung zu referenzieren. Der zweite Chip, ein L3GD20H von ST, ist ein 3D-Gyrometer. Damit kann der Drehwinkel in Relation zur Ausgangsposition bzw. in Bezug auf die Zeit die Drehgeschwindigkeit bestimmt werden. Beide Chips teilen sich einen Inter-Integrated-Circuit-Bus (I^2C), über den sie an den Mikrocontroller angebunden sind.

3.4 Bluetooth-Modul

Die Kommunikation mit dem Zumo ist lediglich seriell über USB-Kabel bzw. direkt am Gerät mit LCD, LEDs und Tastern möglich. Da der Zumo während der Ausführung seiner Programmierung aber stets in Bewegung sein wird, ist eine kabellose Kommunikation nötig. Eine gute Möglichkeit bieten die Bluetooth-Module der Firma Allnet. Diese Module setzen auf den kabellosen Verbindungsstandard Bluetooth v2.0 und dienen der Kommunikation über

kurze Distanzen von bis zu 10 m (ohne Hindernisse). Es werden ultrahochfrequente Radiowellen im „Industrial, Scientific and Medical Band“ (ISM) zwischen 2,4 und 2,485 GHz belegt, um feste und mobile Stationen zu verbinden und private Netzwerke (PAN) aufzubauen. Es werden Übertragungsraten von standardmäßig 9600 bps genutzt.

Es stehen zwei Modulvarianten zur Verfügung. Das Modul HC-05 (siehe Abbildung 16) kann die Bluetooth-Verbindung sowohl als Master als auch als Slave aufbauen und bietet so die Grundlage mehrere der Fahrzeuge ohne externe Zentrale (Host) miteinander zu vernetzen. Dem entgegen steht das Modul HC-06, welches nur eine Funktion als Slave bietet. Das Modul basiert auf einen Cambridge Silicon Radio Chip (CSR) BC417, dem ein 8 Mbit Flashspeicher, ein 26 MHz Quarzkristall, eine Antenne und die nötigen Hochfrequenz (HF)-Leitungen beiseite stehen. Des Weiteren befinden sich auf der Erweiterungskarte (Breakout-Board) ein Spannungsregler, Pegelwandler und eine Status-LED [17].

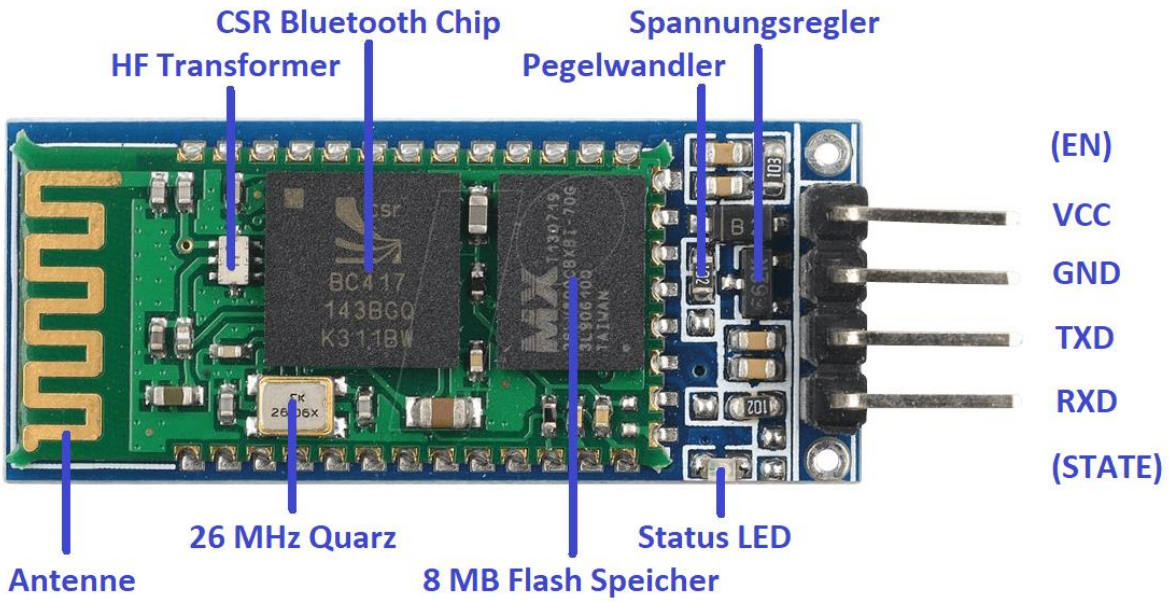


Abbildung 16: Bluetooth-Modul HC-05, vgl. [18]

An der rechten Seite sind die Pins für den externen Anschluss angebracht. An den Pin „VCC“ wird die Versorgungsspannung von 3,6 V bis 6 V angelegt (in diesem Fall 5 V). Darunter ist der nötige Masseanschluss (GND) vorbereitet. Über den Pin „TXD“ werden die Daten vom Bluetooth-Modul an den Arduino gesendet. Die Signalspannung beträgt 3,3 V. Am Arduino ist der damit verbundene Pin als „RXD“ (Empfänger) gekennzeichnet. Das Modul empfängt Daten vom Arduino wiederum über den eigenen „RXD“-Pin zum „TXD“-Pin (Sender) des Arduino. Die beiden Anschlüsse „EN“ und „STATE“ haben bei diesem Modul keinen Pin erhalten, da sie im serienmäßigen Betrieb nicht benötigt werden. Gibt man Spannung auf

„EN“ startet das Modul im „AT Command“-Modus statt im standardmäßigen Daten-Modus. Hier kann unter anderem konfiguriert werden, ob das Modul als Master oder Slave arbeitet und wie hoch die Übertragungsgeschwindigkeit sein soll. „STATE“ gibt an, ob eine Bluetooth-Verbindung hergestellt wurde. Die Leitungen des Zumo sind bereits für die vorinstallierten Module reserviert und sehen keine zusätzliche Hardware vor. Daher kann das Display nicht zeitgleich mit dem Bluetooth-Modul genutzt werden. Eine beispielhafte Verschaltung eines Arduino Leonardo mit einem HC-05 Modul ist in Abbildung 17 dargestellt.

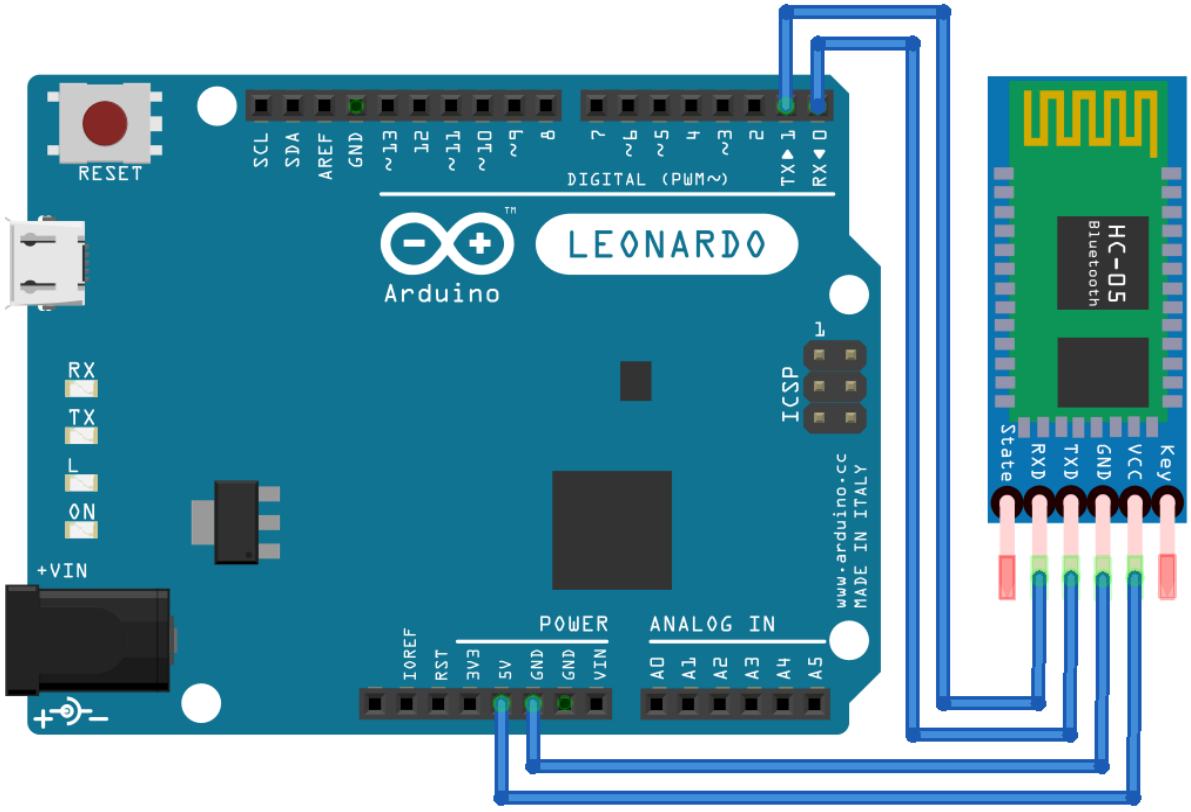


Abbildung 17: Beispielhafte Anbindung des HC-05 an ein Arduino Leonardo (erstellt mit Fritzing)

4 Unterschiede zwischen Testlabor und realem Straßenverkehr

Im folgenden Kapitel werden sowohl die Gegebenheiten im Labor und als auch im realen Straßenverkehr beschrieben. Aus diesen beiden Umgebungen resultieren Unterschiede, die für die weitere Bearbeitung des Projektes von Bedeutung sind. Es wird insbesondere auf die Kommunikationsmöglichkeiten und die baulichen Unterschiede der Straßen eingegangen.

4.1 Kommunikationsmöglichkeiten

Die Kommunikation zwischen den Verkehrsteilnehmern im realen Straßenverkehr unterscheidet sich erheblich von der Kommunikation zwischen den Roboterfahrzeugen. In beiden Fällen sind die Möglichkeiten stark eingeschränkt, entweder in der Anzahl der Übertragungswege oder in der Datenmenge.
Im realen Straßenverkehr spielt insbesondere die Entfernung eine Rolle. Auf weite Entfernungen übernimmt das Radio eine Art der gemeinschaftlichen Informationsverbreitung. Es wird vor Staus, Blitzern und Gefahren gewarnt, um eine vorausschauende Fahrweise zu ermöglichen, d.h. Staus und Gefahren zu umfahren oder die eigene Fahrweise an die Gegebenheiten anzupassen. Umso näher sich die Fahrzeuge kommen, umso mehr geht die Kommunikation einen direkteren Weg. Die Möglichkeit akustische und optische Signale zu geben entsteht. Zunächst kann mit Licht und der Hupe gearbeitet werden, später auch mit Handzeichen. Dabei bleiben die Informationen, die man austauscht, jedoch sehr simpel und erreichen nie eine hohe Komplexität. Man beschränkt sich darauf an Engstellen anderen Verkehrsteilnehmern den Vorrang zu gewähren oder sich über deren Fehlverhalten zu beschweren.
Im beschriebenen Projekt nimmt zwar die Anzahl der Übertragungswege ab, trotzdem steigt die Menge und Komplexität der Informationen enorm. Wie zuvor beschrieben werden zur Datenübertragung Bluetooth-Module genutzt. Auf diese Weise tauschen die Fahrzeuge Fahrdaten aus und erhalten die Mittel, um sich in speziellen Situationen abzusprechen. So kann beispielsweise verhindert werden, dass ein schnelleres Fahrzeug zum Überholen ansetzt, während das langsamere Fahrzeug in eine Kreuzung einbiegt.

4.2 Bauliche Unterschiede der Straßen

Im Durchschnitt ist ein aktuelles Mittelklassefahrzeug etwa 1,80 m breit und 4,80 m lang. Eine Fahrspur auf der Fernautobahn ist laut Richtlinien für die Anlage von Autobahnen (RAA) mindestens 3,50 m breit, also fast das Doppelte der Fahrzeugbreite [19]. Der hier genutzte Zumo ist etwa 10 cm breit. Im normalen Straßenverkehr werden Fahrzeuge hauptsächlich optisch auf die korrekte Richtung kontrolliert, sodass im Regelfall die Fahrbahnmarkierungen nicht berührt werden sollten. Der Zumo hingegen kann erst auf die Berührung der Markierungen reagieren. Um dem Zumo eine möglichst gerade Spur fahren zu lassen, wurde eine Spurbreite von 11 cm gewählt. Damit ist die Fahrbahn nicht viel breiter als der Zumo selbst und verhindert ein zu starkes Pendeln zwischen den Markierungen.
Die Deckschichten öffentlicher Straßen bestehen je nach Verkehrsbelastung aus Asphalt, Beton oder Pflastersteinen. Diese Materialien bieten gegenüber den Fahrzeugen das benötigte Maß an Festigkeit, Haltbarkeit und Haftung. Für den Zumo sind teilweise andere Aspekte wichtig. Insbesondere darf die Rauheit nicht zu groß sein, da das Schild des Zumo schnell an Unebenheiten hängen bleiben kann und der Antrieb durch die hohe Haftung an der Gummikette nicht optimal arbeitet.
Die verschiedenen Materialien der Straßendecken sind auch der Grund für die unterschiedlichen Farben und Helligkeiten der Straßen. Üblicherweise ist der Belag dunkel und es sind weiße Fahrbahnmarkierungen aufgebracht. Diese Kombination mit möglichst hohem Kontrast ist auch ideal für die Liniensensoren des Zumo, da dieser lediglich Helligkeitsunterschiede bzw. Reflexionsgrade erkennt. Die korrekte Einstufung der Helligkeit ist außerdem stark von den herrschenden Lichtverhältnissen abhängig. Die Fahrzeuge im Straßenverkehr sind für alle Licht- und Wetterverhältnisse ausgerüstet und die Fahrer passen ihren Fahrstil entsprechend an, der Zumo jedoch benötigt für eine fehlerfreie Erkennung der Fahrbahn konstante Lichtverhältnisse, die die Sensoren nicht beeinflussen.

4.3 Beschreibung der Teststrecke im Labor

Entsprechend der Aufgabenstellung und den vorher diskutierten Aspekten wurde die Strecke für den Zumo erstellt. Die in Abbildung 18 dargestellte Straßenführung stellt die einfachste Lösung der Aufgabenstellung dar. Die Strecke besteht aus einem äußeren Ring sowie einer Verbindungstraße in der Mitte. An den beiden Kreuzungen kann in zwei Richtungen abgebogen werden. Da man aus drei Richtungen an die Kreuzung gelangt, ergeben sich sechs

Kombinationen. Auf den drei Geraden zwischen den Kreuzungen besteht die Möglichkeit zu überholen. Die vier Kurven auf dem äußeren Ring schließen die Fahrbahn ab.

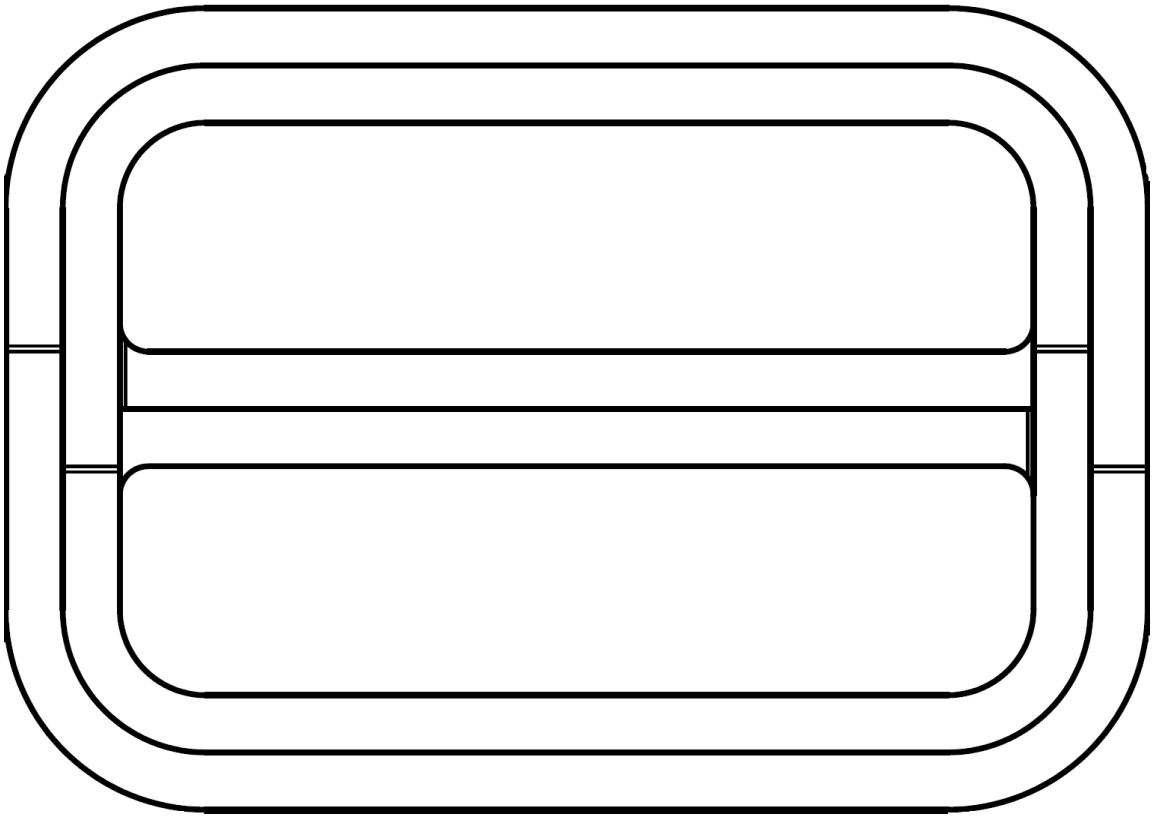


Abbildung 18: Teststrecke im Labor (erstellt mit Autodesk AutoCAD 2016)

Die Erstellung der Teststrecke stellte sich als schwieriger heraus als erwartet. Zunächst war angedacht, dass eine Holzplatte mit dunklem Furnier das Grundgerüst bilden soll. Das Furnier hätte auch als Straßenbelag fungiert. Darauf wurden weiße Streifen als Fahrbahnmarkierungen und einige rote Streifen als Signalgeber geklebt. Die Signale kündigen Kreuzungen an und enthalten einen Hinweis auf die möglichen Richtungen, in die abgebogen werden kann. Die Oberflächen wurden von den Liniensensoren gut erkannt. Leider wurde die Teststrecke unverhältnismäßig groß, sodass Transport und dauerhafter Aufbau im Labor nicht oder nur mit größerem Aufwand möglich wäre.

Letztendlich wurde die Strecke auf Papier gedruckt und die einzelnen Seiten zusammengeklebt. Der Aufbau gestaltete sich auf diese Weise deutlich einfacher und das gesamte Gebilde kann bei Bedarf zusammengerollt werden. Einen Nachteil hat das Material der Strecke jedoch; die Messwerte der Liniensensoren liegen bei allen Farben auf Papier deutlich näher beieinander. Wie in Tabelle 1 zu sehen ist, liegt die Differenz zwischen Furnier und Isolierband bei etwa 900 Punkten, zwischen schwarz bedrucktem und weißem Papier

jedoch nur bei 500 Punkten. Außerdem schwanken die Werte und die einzelnen Sensoren weichen für die gleiche Oberfläche stark voneinander ab. Daher wurde im finalen Aufbau ein furnierähnlicher Klebestreifen statt dem Isolierband als Signalgeber aufgeklebt. So sind die Messwerte für die verschiedenen Oberflächen gut verteilt. Der maximal darstellbare Bereich liegt zwischen 0 und 2000 Punkten.

Farbe/Material	**Linker Sensor**	**Mittlerer Sensor**	**Rechter Sensor**
Weißes Papier	130	77,5	120
Rotes Papier	230	120	260
Schwarzes Papier	635	327,5	607,5
Weißes Isolierband	300	170	330
Rotes Isolierband	330	210	360
Dunkles Furnier	1227,5	640	1137,5

Tabelle 1: Messwerte der Liniensensoren unterschiedlicher Oberflächen

In Abbildung 19 wurden noch einmal die Ergebnisse der Oberflächenuntersuchung zusammengefasst.

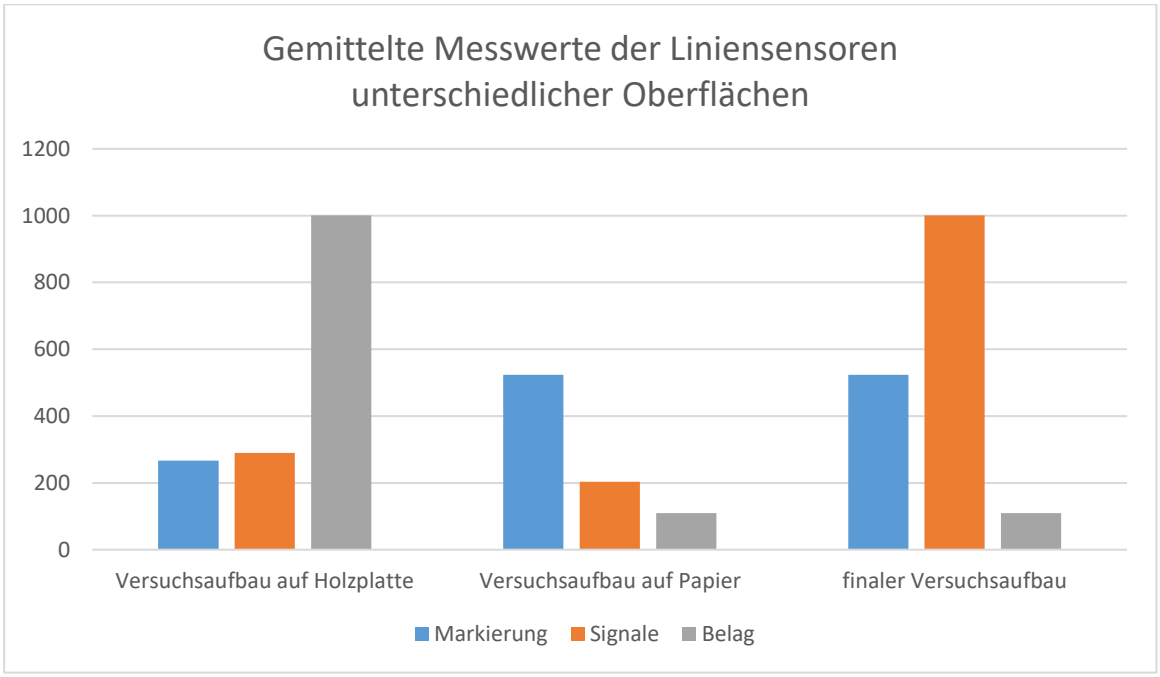


Abbildung 19: Gemittelte Messwerte der Liniensensoren unterschiedlicher Oberflächen

5 Programmablauf und Funktionen

Grundlegend bestehen Programmierungen für Arduino-basierte Systeme aus zwei Funktionen. Die Funktion Setup läuft einmalig nach dem Start des Systems und wird normalerweise nicht wiederholt. Anschließend wird die Funktion Loop durchlaufen. Nach dem Ende der Funktion springt die Programmierung zurück an den Start der Loop und durchläuft diese erneut. Es handelt sich um eine Schleife, die immer wieder ausgeführt wird, bis das System ausgeschaltet wird. Programmteile, die mehrfach aufgerufen werden, können in Unterfunktionen ausgelagert werden. Der Aufruf der Unterfunktionen kann sowohl aus den Grundfunktionen als auch aus anderen Unterfunktionen heraus erfolgen. Diese Unterteilung fördert auch die Übersichtlichkeit von langen Programmierungen. In den Anlagen befinden sich Programmablaufpläne und der vollständige Programmcode.
Lokale Variablen können in den Funktionen deklariert werden. Diese sind nur verfügbar während die Funktion durchlaufen wird und nach dem Ende der Funktion wird der belegte Speicherbereich wieder freigegeben. Vorteilhaft ist die optimale Speichernutzung, jedoch kann die Variable nicht von anderen Funktionen genutzt werden. Um Variablen außerhalb der erzeugenden Funktion zu nutzen, müssen diese entweder übergeben werden oder globale Variablen festgelegt werden. Globale Variablen und Konstanten werden am Anfang der Programmierung noch vor den Funktionen deklariert und definiert. Sie sind für alle Funktionen verfügbar. Die Einbindung von Bibliotheken (.h-Dateien) und die Objekterstellung erfolgen ebenfalls vor den Funktionen.

5.1 Einbindung der Bibliotheken, Objekte, Globale Variablen

Diese Programmierung nutzt drei externe Bibliotheken. „Wire.h“ ermöglicht dem Arduino das von Philips entwickelte I²C-Protokoll zu nutzen, über das die Inertialsensoren angebunden sind. „Zumo32U4.h“ stellt alle weiteren Vorbereitungen zur Verfügung, um die Sensoren und Möglichkeiten des Zumo komfortabel zu nutzen. Anschließend werden den Klassen in der Bibliothek „Zumo32U4.h“ die gewünschten Objektbezeichnungen in der Programmierung zugewiesen. Diese Bezeichnungen werden im folgenden Programmcode für die Aufrufe der verwendeten Bauteile genutzt. Eine weitere Bibliothek „MedianFilter.h“ stellt Funktionen für den Medianfilter bereit, der für die Filterung der Sensorwerte verwendet wird. Für jeden Abstands- und Liniensensor wird ein separates Objekt entsprechend der Medianfilterklasse

erstellt. Das Filterfenster umfasst drei Werte, sodass möglichst schnell auf veränderte Werte reagiert werden kann.
Außerdem müssen die globalen Variablen für die Sensorwerte deklariert werden. Da diese Werte von mehreren Funktionen benötigt werden, können keine lokalen Variablen verwendet werden. Neben den Sensordaten werden auch Variablen für die Zeiten im System, die maximale Geschwindigkeit, die serielle Ausgabe (in Form eines Datenfeldes) und für Daten, die aus der Bluetooth-Verbindung gelesen werden, bereitgestellt.

5.2 Setup

In der Setup-Funktion wird als erstes die Bluetooth-Verbindung in einer Unterfunktion initialisiert. Danach werden nacheinander die weiteren Unterfunktionen des Setups durchlaufen, um die Sensoren zu initialisieren und teilweise auch zu kalibrieren. Nachdem dies abgeschlossen ist, wird das Programm pausiert, bis die weitere Programmierung in der Loop-Funktion durch einen Tastendruck gestartet wird. Vor und nach der Sensorkalibrierung wird je eine Pause von 500 ms eingelegt, um dem Nutzer Zeit zu geben die Hände vom Gerät zu nehmen ohne die Funktion zu stören.

5.3 Setup-Unterfunktionen

Die fünf Setup-Unterfunktionen werden nur einmalig aufgerufen. Um die einzelnen Codezeilen zu ordnen und die Programmierung übersichtlicher zu gestalten, wurden bereits hier Unterfunktionen erstellt. In den Funktionen werden die Linien-, Abstands-, Drehbewegungs- und Beschleunigungssensoren initialisiert und, abgesehen von den Abstandssensoren, auch kalibriert. Außerdem wird die Bluetoothverbindung aufgebaut.

5.3.1 Bluetooth-Verbindung

Die Verbindung wird mit einer Geschwindigkeit von 9600 Baud initialisiert und als „Serial1“ bezeichnet. Normalerweise kann die „read“-Funktion bis zu 1000 ms dauern, daher wird mit einem Timeout die maximale Dauer auf 10 ms festgelegt. Nachdem die Verbindung hergestellt wurde, wird eine Meldung ausgegeben. Abschließend werden eventuell vorhandene Daten aus dem Puffer gelesen und verworfen, um einen unbelasteten Programmstart zu ermöglichen.

5.3.2 Liniensensoren

Nach der Initialisierung von drei Liniensensoren mit „lineSensors.initThreeSensors()“ werden diese kalibriert, d. h. während eine Schleife läuft, überfährt der Zumo das Gebiet, auf dem er steht, in einem Halbkreis und wieder zurück auf seine Ausgangsposition. Dabei wird für jeden Sensor eine Messreihe mit insgesamt 120 Messwerte aufgenommen und summiert. Teilt man diese Summen durch die Anzahl, erhält man den Mittelwert für jeden der drei Sensoren, der als Offset von den späteren Messwerten abgezogen wird.

$$l_{kal} = \frac{\sum_{i=0}^{119} l_i}{120} = \frac{l_0 + l_1 + l_2 + \cdots + l_{119}}{120} \tag{6}$$

5.3.3 Abstandssensoren

Die Abstandssensoren werden lediglich mit dem Befehl „proxSensors.initThreeSensors()“ initialisiert. Eine Kalibrierung ist nicht nötig.

5.3.4 Drehbewegungssensor

Zunächst wird der Drehbewegungssensor mit „gyro.init()“ initialisiert und die Verbindung überprüft. Sollte keine Verbindung hergestellt werden können, wird eine Fehlermeldung ausgegeben. Darauf folgend werden Registereinträge durch den Befehl „gyro.writeReg()“ geändert. Im ersten Register „CTRL1“ werden die Ausgaberate (800 Hz) und der Tiefpassfilter (100 Hz) festgelegt. Außerdem wird der normale Modus aktiviert und alle drei Achsen (x, y, z) verfügbar gemacht. Im Register „CTRL4“ wird für die höchste Genauigkeit die höchste verfügbare Auflösung gewählt. Der Hochpassfilter wird im Register „CTRL5“ deaktiviert. Anschließend erfolgt auch hier eine Kalibrierung, in dem 1024 Messwerte genommen werden, deren Mittelwert als Offset genutzt wird.

$$g_{kal} = \frac{\sum_{i=0}^{1023} g_i}{1024} = \frac{g_0 + g_1 + g_2 + \cdots + g_{1023}}{1024} \tag{7}$$

Für die spätere Programmierung wird nur die Bewegung um die z-Achse des Sensors benötigt, daher wird auch nur ein Offset berechnet. Die Abbildung 20 soll verdeutlichen um welche Dimension des Drehbewegungssensors es sich handelt.

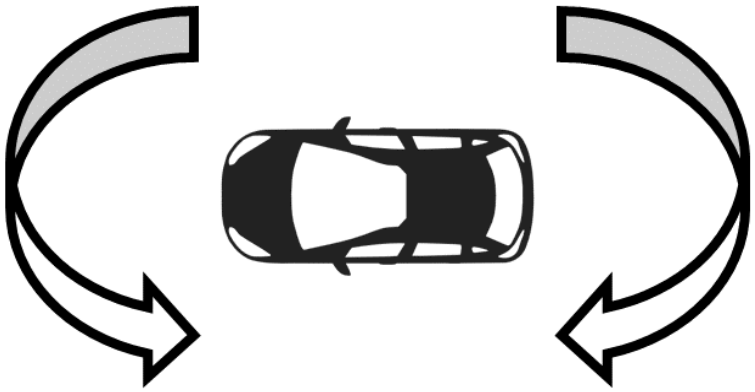


Abbildung 20: Drehbewegungssensor z-Achse (erstellt mit Microsoft PowerPoint 2013)

5.3.5 Beschleunigungssensor

Auch der Beschleunigungssensor wird mit „compass.init()“ initialisiert und wieder die Verbindung geprüft. Anschließend erfolgt die Kalibrierung. Diesmal werden die Grundeinstellungen verwendet (compass.init). Ähnlich der Offsetberechnung des Drehbewegungssensors wird auch hier der Mittelwert aus 1024 Messwerten als Offset genutzt.

$$c_{kal} = \frac{\sum_{i=0}^{1023} c_i}{1024} = \frac{c_0 + c_1 + c_2 + \cdots + c_{1023}}{1024} \quad (8)$$

Da für die spätere Programmierung nur die x-Achse gebraucht wird, wird auch nur für diese ein Offset berechnet. Abbildung 21 verdeutlicht die genutzte Dimension.

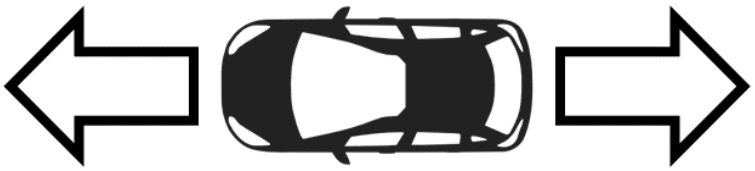


Abbildung 21: Beschleunigungssensor x-Achse (erstellt mit Microsoft PowerPoint 2013)

5.4 Loop

Die Loop-Funktion ist der Teil der Programmierung, der immer wieder ausgeführt wird. Bevor eine Entscheidung getroffen werden kann, was der Zumo im nächsten Moment zu tun hat, müssen aktuelle Messwerte erzeugt werden. Dazu werden die Update-Funktionen durchlaufen. Zu diesem Zweck hat jeder Sensor wie im Setup eine eigene Unterfunktion. Dieses Mal berechnet eine zusätzliche Funktion die Zykluszeit und eine weitere Funktion misst die Anzahl der Motordrehungen.
Bevor eine Funktion anhand der Sensordaten ausgewählt wird, muss überprüft werden, ob neue Daten vom Bluetooth-Modul empfangen wurden. Führt eines der Fahrzeuge bereits ein Manöver aus, wurde dies per Bluetooth mitgeteilt. Dazu liest der Zumo nun die Daten aus dem Puffer des Bluetooth-Moduls und schreibt diese in die Variable „btData“. Um die Abfragezeit so kurz wie möglich zu halten, wird auf Stichworte im Klartext verzichtet. Es wird lediglich eine „1“ für ein laufendes Manöver gesendet bzw. eine „0“ für die Beendigung des Manövers. Anschließend wird eine weitere Variable „btBuffer“ als Puffer eingesetzt, damit der Status bis zum Erhalt einer neuen Nachricht erhalten bleibt.
Im Falle eines laufenden Manövers wird die Geschwindigkeit verringert und eine verhaltene Funktionsauswahl durchgeführt. Der Zumo stoppt, wenn eine Kreuzung erreicht, eine Markierung überfahren oder ein Hindernis bzw. überholendes Fahrzeug erkannt wird. Ist dies nicht der Fall, wird die Funktion „Spurhalten“ mit der verminderten Geschwindigkeit ausgeführt.
Wurden von anderen Fahrzeugen keine Manöver angekündigt, führt der Zumo die standardmäßige Funktionsauswahl durch. Anhand der Messwerte der Beschleunigungs-, Abstands- und Liniensensoren wird die Funktion mit der höchsten Priorität ausgeführt. Die Funktion „Kontaktvermeiden“ startet, wenn der Beschleunigungssensor einen hohen Wert in Richtung der x-Achse misst. Zeigen die vorderen Abstandssensoren den höchsten Wert (6) an, wird die Funktion „Hindernisumfahren“ durchlaufen. Abschließend werden die Messwerte für die Liniensensoren kontrolliert. Liegt einer dieser Werte über 300/500 Punkten, wurde eine Kreuzung erreicht und die Funktion „Abbiegen“ wird ausgeführt. Erkennt der mittlere Liniensensor die Fahrbahnmarkierung, startet die Funktion „Spurfinden“. Wenn keine der genannten Bedingungen erfüllt wurde, wird die Funktion „Spurhalten“ ausgeführt.

5.5 Update-Funktionen

In den folgenden Punkten werden die vorher genannten Update-Funktionen der „Loop"-Funktion näher beschrieben.

5.5.1 Zykluszeit

Um die Dauer eines vollständigen Durchlaufs der Loop-Funktion nachvollziehen zu können, wird jeweils am Anfang und am Ende der Loop die Funktion „LoopTiming" aufgerufen. Beim ersten Aufruf wird die aktuelle Zeit mit dem Befehl „millis()" erfasst und in eine Variable abgelegt. Beim zweiten Aufruf wird ein neuer Zeitwert abgerufen. Zieht man die im vorherigen Aufruf ermittelte Zeit von der zuletzt erfassten Zeit ab, erhält man die Durchlauf- bzw. Zykluszeit.

$$\Delta t_Z = t_{neu} - t_{alt} \tag{9}$$

Um die serielle Ausgabe nicht mit den Messwerten zu überfluten, wird nur nach jeweils 100 Aufrufen der Funktion eine Auswertung der Messwerte ausgegeben. Dazu zählen ein Mittelwert aus allen Zykluszeiten der aktuellen Messreihe sowie der Minimal- und der Maximalwert.
Die serielle Ausgabe anderer Messwerte erfolgt auch in diesem Rhythmus seriell über die Bluetooth-Verbindung. Dazu wird ein Container erstellt, in dem die Variablen eingetragen werden. Zu den übergegebenen Variablen zählen die aktuellen Messwerte der Linien- und Abstandssensoren, sowie der berechnete Drehwinkel und die Umdrehungen laut Encodern.

5.5.2 Abstandssensoren

Das Update der Abstandssensoren ist umfangreicher als die Setup-Funktion. Zwar bedürfen sie keiner Berechnung, jedoch ist eine Abfrage nötig, ob bereits ein anderes Fahrzeug für die Abstandsmessung IR-Signale aussendet. Wenn mehrere Fahrzeuge gleichzeitig Abstandsmessungen durchführen, wird das ausgesendete Licht eines Zumos auch von anderen Zumos empfangen und als eigenes Messsignal interpretiert. Daher wird vor der Durchführung der Messung mit der Arduino-Funktion „pulseIn" auf den Pins 20, 22 und 4

geprüft, ob bereits Signale unterwegs sind. Erst wenn keine Fremdmessungen mehr erkannt werden, startet die eigene Abstandsmessung.
Die Sensoren werden mit dem Befehl „proxSensors.read()“ abgefragt und die Werte mit „proxSensors.countsXWithYLeds()“ in Variablen abgelegt. Für X ist die Position des Empfängers und für Y ist die Position des Senders einzugeben. Die Abstandsmessung sowie der vorherige Abfragevorgang dauern nur wenige Millisekunden, sodass keine Probleme auftreten, wenn die Messung zunächst in die „Warteschlange“ verschoben wird.
Die erhaltenen Messwerte werden zunächst in die Filterobjekte „ProxFilter“ verschoben, um Ausreißer auszusortieren. Wertänderungen werden so erst nach dem zweiten Durchlauf weitergegeben und führen zu einer Funktionsauswahl.

5.5.3 Liniensensoren

Um aktuelle Werte für die Liniensensoren zu erhalten, genügt ein einzelner Befehl. „lineSensors.read(lineRaw)“ fragt die Sensoren ab und speichert diese in der Variable „lineRaw“ ab. Um kalibrierte Werte zu erhalten, werden anschließend die Kalibrierungswerte l_{kal} aus der Setup-Funktion abgezogen.

$$l = l_{neu} - l_{kal} \tag{10}$$

Auch die Messwerte der Liniensensoren werden mittels Medianfilter optimiert. Dazu werden sie in die Filterobjekte „LineFilter“ verschoben.

5.5.4 Drehbewegungssensor

Aktuelle Werte des Drehbewegungssensors werden mit dem Befehl „gyro.read()“ ausgelesen und für alle drei Achsen gespeichert. Der für die Berechnung des Drehwinkels benötigte Wert ist in „gyro.g.z“ zu finden. Zunächst wird wieder der, im Setup berechnete, Offset abgezogen.

$$g = g_{neu} - g_{kal} \tag{11}$$

Anschließend wird dem Messwert ein Zeitwert zugewiesen und die Differenz zum vorherigen Zeitwert berechnet.

$$\Delta t_g = t_{neu} - t_{alt} \tag{12}$$

Aus dem Messwert des Gyrometers und der Zeitdifferenz kann nun der Drehwinkel in Grad (°) umgerechnet werden.

$$\alpha = g * \Delta t_g * 360 \tag{13}$$

5.5.5 Beschleunigungssensor

Um den Beschleunigungssensor zu nutzen, werden aktuelle Werte des Sensors mit „compass.read()“ abgefragt und für die x-Achse in „compass.a.x“ abgelegt. Vom Rohwert werden der, anfangs berechnete, Offset und der Messwert aus dem vorhergehenden Zyklus abgezogen.

$$c = c_{neu} - c_{kal} - c_{alt} \tag{14}$$

Der aktuelle, kalibriete Wert wird für die Berechnung im folgenden Zyklus in der Variable „compassLastUpdate“ gespeichert. Ein Rohwert von 16384 Punkten entspricht etwa der Fallbeschleunigung mit 9,81m/s².

5.5.6 Quadraturencoder

Die Werte der Quadraturencoder können mit den Befehlen „encoders.getCountsLeft“ und „encoders.getCountsRight“ abgefragt und in Variablen abgelegt werden. Um die Anzahl der Umdrehungen N der Kettenräder zu berechnen, müssen die „Counts“ n durch 909,7 dividiert werden.

$$N = \frac{n}{909{,}7} \tag{15}$$

Die Berechnung der Übersetzung ist in Formel (4) zu finden.

5.6 Loop-Unterfunktionen

In den folgenden Punkten werden die, in Kapitel 5.4 genannten, Unterfunktionen der „Loop"-Funktion näher beschrieben.

5.6.1 Kontaktvermeiden

Wurde ein Kontakt mithilfe des Beschleunigungssensors festgestellt, startet die Funktion „Kontaktvermeiden", jedoch nur unmittelbar nach der Funktion „Spurhalten", da bei Geschwindigkeitsänderungen der Sensor definitiv ansprechen würde. Der Zumo stoppt zunächst und fährt dann rückwärts, um sich in einen größeren Abstand zum Hindernis zu bringen. Dabei wird verhindert, dass die Fahrbahn verlassen wird. Start und Ende des Manövers werden jeweils durch eine Meldung über die Bluetooth-Verbindung angekündigt. Anschließend kann bei ausreichendem Abstand die Funktion „Hindernisumfahren" ausgeführt werden. In Abbildung 22 ist der Ablauf der Funktion bildlich dargestellt.

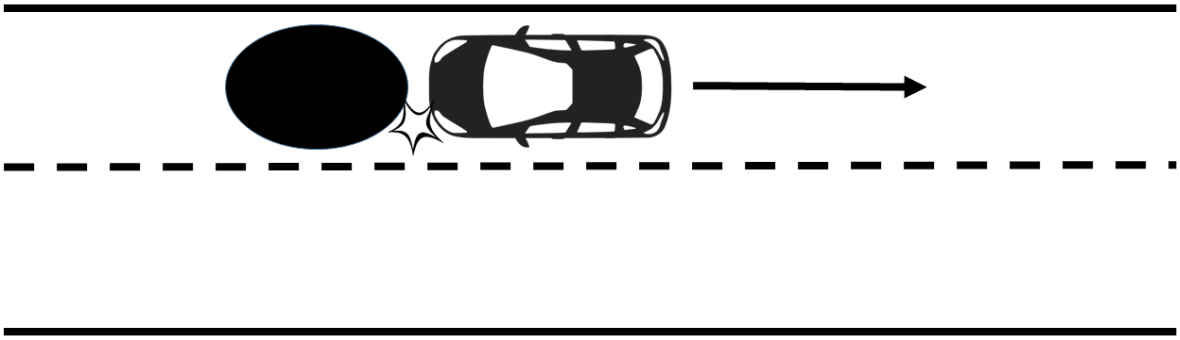


Abbildung 22: Grafik zur Funktion "Kontaktvermeiden" (erstellt mit Microsoft PowerPoint 2013)

5.6.2 Hindernisumfahren

Wird ein Hindernis in einem ausreichenden Abstand detektiert, startet die Funktion „Hindernisumfahren". Bei dem Hindernis kann es sich um ein weiteres in Fahrt befindliches Fahrzeug handeln oder um ein stillstehendes Objekt. Auch dieses Manöver wird über die Bluetooth-Verbindung gemeldet. Die Funktion besteht aus drei Teilen. Im ersten Teil erfolgt der Spurwechsel auf die Gegenfahrbahn. Dazu wird zunächst der Winkel auf null gesetzt. Anschließend dreht das Fahrzeug nach links bis es in einem Winkel von 45° steht. Nun kann

der Zumo die Mittellinie überfahren. Sobald auch der rechte Liniensensor die Mittellinie passiert hat, folgt die Drehung nach rechts bis der Winkel wieder 0° beträgt.
Da sich das Fahrzeug nun auf der Gegenfahrbahn befindet, kann, wenn die Fahrspur frei ist, der zweite Teil des Überholvorgangs beginnen. Der Zumo erhöht die Geschwindigkeit und fährt solange gerade aus, bis das Hindernis passiert wurde. Dazu muss zunächst das Hindernis eingeholt werden. Anschließend wird an dem Hindernis vorbeigefahren. Ob die rechte Fahrspur wieder frei ist, überprüft der rechte seitliche Abstandssensor. Bevor der letzte Teil beginnt, fährt das Fahrzeug noch ein Stück weiter, um einen Sicherheitsabstand zu erlangen, und verringert anschließend die Geschwindigkeit wieder auf das Standardniveau.
Im dritten Teil folgt der Spurwechsel zurück auf die rechte Fahrspur. Dieser läuft sehr ähnlich zum ersten Teil ab. Der Winkel wird erneut auf null gesetzt. Nun wird eine Drehung nach rechts solange ausgeführt, bis der Winkel -45° beträgt. Anschließend wird die Mittellinie überfahren. Hat auch der linke Liniensensor die Mittellinie passiert, wird eine Bewegung nach links durchgeführt, bis der Winkel wieder 0° beträgt. Der Zumo befindet sich nun wieder vollständig auf der rechten Fahrspur und hat das Hindernis passiert. Der Überholvorgang ist abgeschlossen und es erfolgt eine entsprechende Meldung über die Bluetooth-Verbindung. In Abbildung 23 ist der Ablauf der Funktion bildlich dargestellt.

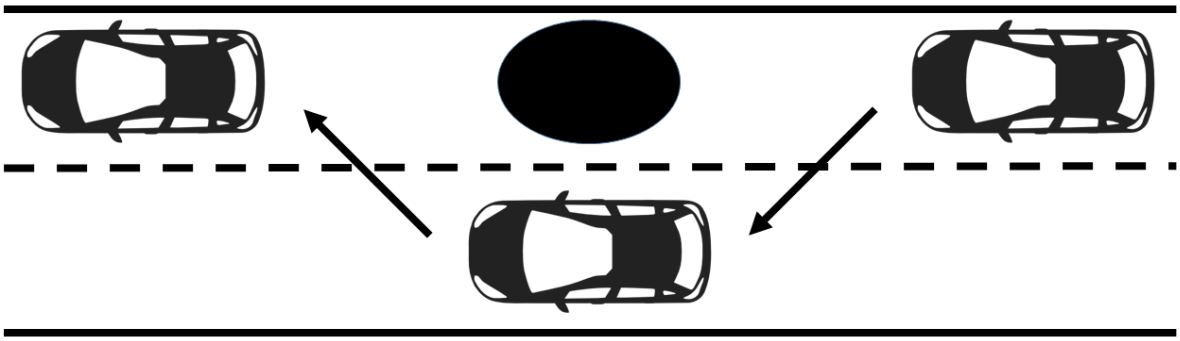


Abbildung 23: Grafik zur Funktion "Hindernisumfahren" (erstellt mit Microsoft PowerPoint 2013)

5.6.3 Abbiegen

Die Kreuzungen auf der Strecke sind speziell markiert. Für die Liniensensoren besitzen die Markierungen einen besonders hohen Wert, um sich von den restlichen Fahrbahnmarkierungen abzuheben. Die Funktion „Abbiegen“ kündigt sich über eine entsprechende Meldung auf der Bluetooth-Verbindung an. Bevor das Abbiegemanöver beginnt, muss der Zumo eine Richtung auswählen. Um dem Zumo mitzuteilen welche Form die Kreuzung hat, wurde die Markierung bearbeitet. Erfassen nur der linke und mittlere

Liniensensor die Markierung, ist es möglich nach links abzubiegen oder geradeaus zu fahren. Analog dazu ist es möglich nach rechts abzubiegen oder geradeaus zu fahren, wenn der nur der rechte und mittlere Liniensensor die Markierung erkennen. Wird die Markierung von beiden äußeren Sensoren detektiert, kann in beide Richtungen abgebogen werden, jedoch ist es nicht möglich geradeaus zu fahren. Es existieren demzufolge stets zwei Möglichkeiten. Um diese Entscheidung zu treffen, wird eine Zufallszahl erzeugt und mit den Möglichkeiten verglichen. Durch die Wahl der Fahrbahnmarkierungen muss unterschieden werden zwischen dem Ablauf beim Linksabbiegen von der Verbindungsstrecke und dem Linksabbiegen vom Rundkurs. Insgesamt existieren damit vier Manöver, die der Entscheidungsfindung folgen können:

- nach rechts abbiegen,
- geradeaus fahren,
- nach links abbiegen von der Verbindungsstrecke oder
- nach links abbiegen vom Rundkurs.

Die Abbiegemöglichkeiten in Abhängigkeit vom Startpunkt sind in Abbildung 24 bildlich dargestellt.

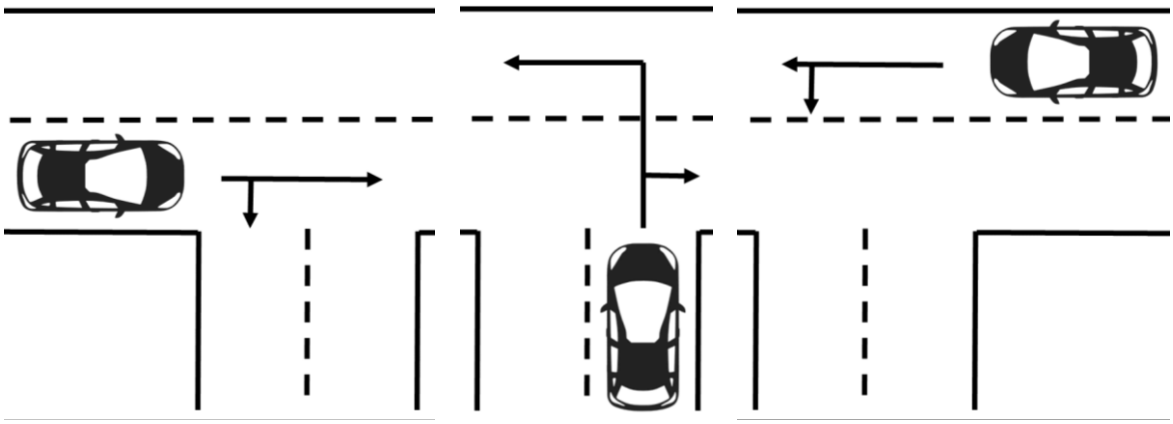


Abbildung 24: Grafik zur Funktion "Abbiegen" (erstellt mit Microsoft PowerPoint 2013)

Um von der Verbindungsstrecke nach links abzubiegen, müssen zwei Teile durchgeführt werden. Zunächst wird bis zur Kreuzungsmitte gefahren. Dazu fährt der Zumo zunächst geradeaus über die äußere Fahrbahnmarkierung und anschließend weiter bis die Liniensensoren die Mittellinie registrieren. Dieser Weg entspricht ungefähr der Länge des Zumo. Anschließend wird der Winkel null gesetzt und das Fahrzeug dreht sich solange nach links bis ein Winkel von 90° erreicht wurde. Es wird darauf geachtet nicht die Fahrbahnmarkierungen zu überfahren, um auf der richtigen Fahrspur anzukommen.

Beim Abbiegen vom Rundkurs nach links, ist es nicht nötig vor dem Abbiegen geradeaus zu fahren, da die Platzierung der Markierung entsprechend weiter hinten erfolgte. Sobald die Markierung erreicht und die Entscheidung für das Abbiegen getroffen wurde, wird der Winkel auf null gesetzt. Anschließend erfolgt die Linksdrehung bis 90°.
Auch beim Rechtsabbiegen kann auf die Fahrt zur Kreuzungsmitte verzichtet werden. Wie beim Abbiegen nach links wird der Winkel auf null gesetzt, jedoch erfolgt die Drehung nach rechts, bis der Winkel in diesem Fall -90° beträgt. Auch hier wird zusätzlich nachgebessert mit Hilfe der Liniensensoren, um die Fahrspuren nicht zu verlassen. Anschließend kann die Fahrt auf der neuen Fahrspur fortgesetzt werden.
Die letzte Möglichkeit ist die einfachste, da nicht abgebogen wird: die Kreuzung wird nur überfahren. Der beendete Abbiegevorgang wird in allen vier Fällen abschließend mit einer Meldung über die Bluetooth-Verbindung kommuniziert.

5.6.4 Spurhalten

Obwohl die Funktion „Spurhalten“ den anderen Funktionen nachsteht und nur ausgeführt wird, wenn die Bedingungen aller anderen Funktionen nicht erfüllt sind, sollte sie doch die am häufigsten ausgeführte Funktion sein. Sie hält das Fahrzeug auf seiner Spur und verhindert, dass diese verlassen wird. Abhängig von den Liniensensoren gibt es drei Möglichkeiten wie die Funktion ausgeführt wird.
Registriert der linke Liniensensor, dass die Fahrbahnmarkierung überfahren wurde, wird kurzzeitig nach rechts gelenkt, indem der Antrieb auf der linken Seite schneller dreht als rechts. Analog dazu wird kurzzeitig nach links gesteuert, wenn der rechte Liniensensor das Überfahren der Fahrbahnmarkierung erkennt (Abbildung 25).
Die beiden Funktionen werden in einer Schleife für 100 ms ausgeführt, um eine gleichmäßigere Bewegung zu realisieren. In beiden Fällen wird die Funktion abgebrochen, wenn der Liniensensor auf der anderen Seite bereits die Fahrbahnmarkierung erkennen sollte. Registriert keiner der Liniensensoren die Fahrbahnmarkierungen fährt der Zumo unverändert weiter geradeaus.

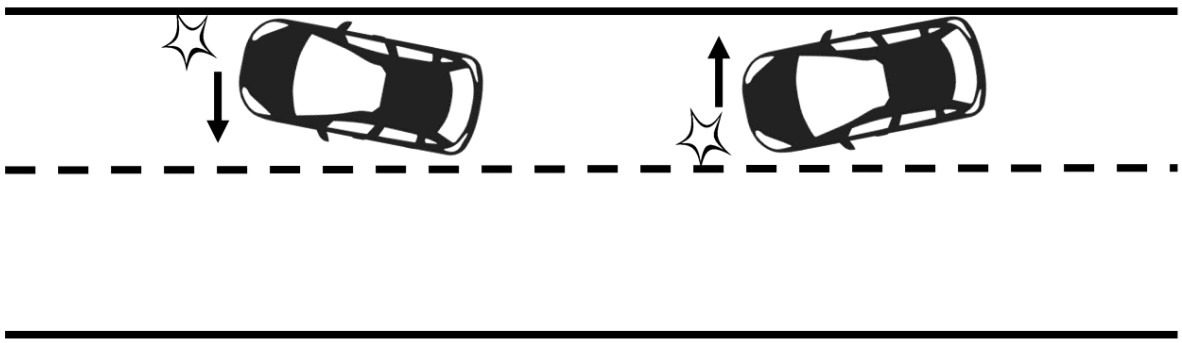


Abbildung 25: Grafik zur Funktion "Spurhalten" (erstellt mit Microsoft PowerPoint 2013)

5.6.5 Spurfinden

Sollte der mittlere Liniensensor die Fahrbahnmarkierung wahrnehmen, ist davon auszugehen, dass die Linie bereits überfahren wurde (Abbildung 26). Um den Zumo wieder vollständig in die Fahrspur zu steuern, fährt dieser solange rückwärts bis einer der äußeren Liniensensoren die Fahrbahnmarkierung erkennt. Da dieses Manöver die Wahrscheinlichkeit von Kollisionen mit anderen Fahrzeugen erhöht, wird auch hier eine Meldung über die Bluetooth-Verbindung ausgegeben.

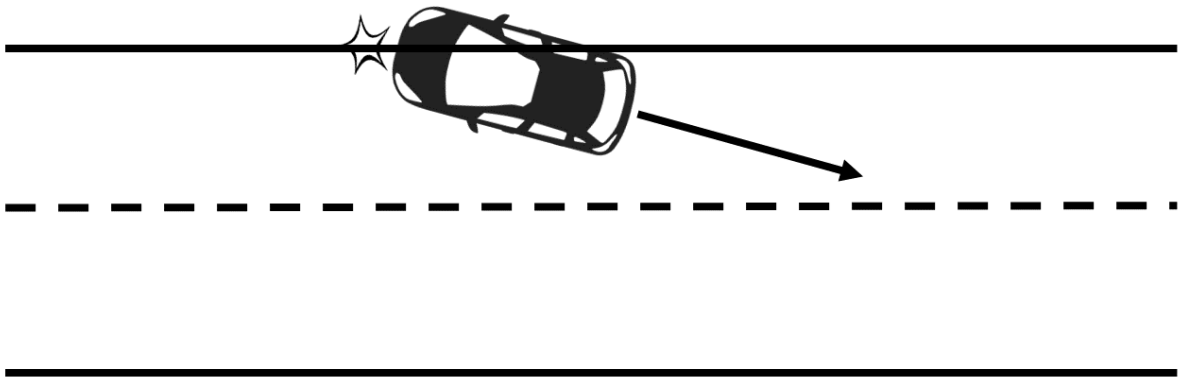


Abbildung 26: Grafik zur Funktion "Spurfinden" (erstellt mit Microsoft PowerPoint 2013)

6 Zusammenfassung und Ausblick

Sowohl auf der Straße als auch im Labor zeigt sich, dass das Thema „Autonomes Fahren“ noch nicht abgeschlossen ist und weiterhin viel Arbeit bedarf. Zwar wurden die Aufgaben der Masterarbeit erfüllt, jedoch ist die Technik nicht vollkommen zuverlässig. Zur Realisierung der Programmierung wurde eine Fahrstrecke mit zweispuriger Straße angefertigt, die Möglichkeiten zum Abbiegen und Überholen bietet und in beiden Richtungen befahren werden kann. Der Zumo erkennt Kreuzungen und biegt an diesen ab. Werden Hindernisse erkannt, wird, bei bereits laufendem Manöver, langsam herangefahren und gestoppt oder, wenn keine Manöver angekündigt wurden, überholt. Je nach Situation wird die Geschwindigkeit angepasst. Über Bluetooth kommunizieren die Fahrzeuge miteinander und kündigen die Manöver an, um Kollisionen zu vermeiden.

Jedoch kommt es hin und wieder zu Fehlmessungen der Sensoren und somit zur falschen Funktionsauswahl. Teilweise erfolgt die Funktionsauswahl aufgrund der Messung eines einzelnen Sensors, ohne von weiteren Sensoren bestätigt zu werden. Von großem Einfluss ist insbesondere die starke Abhängigkeit bezüglich der Oberflächenbeschaffenheit (Rauheit und Farbe) der Hindernisse für die Abstands- und Liniensensoren. So kommt es, dass manche (dunkle) Objekte von den Abstandssensoren sehr spät erkannt werden. Andere (helle) Objekte, die sich weit abseits der Fahrstrecke befinden, werden jedoch als Hindernis eingestuft. Teilweise werden auch Unebenheiten der Fahrstrecke als Hindernis wahrgenommen. Ein weiterer Punkt, der zu Fehlmessungen der Abstandssensoren führt, ist die Interferenz der Fahrzeuge untereinander. Die von einem Fahrzeug ausgesendeten IR-Strahlen werden fälschlicherweise von dem anderen Fahrzeug als die Eigenen registriert. In allen drei Fällen versucht der Zumo die angeblich vorhandenen Hindernisse zu umfahren. Die Liniensensoren funktionierten erst beim dritten Teststreckenaufbau zufriedenstellend. Der erste Aufbau auf einer Holzplatte mit Klebeband war in der benötigten Größe nicht transportabel. Die Teststrecke ausschließlich auf bedrucktem Papier zu realisieren lieferte keine ausreichend differenzierten Messwerte für die drei Kategorien Belag, Markierung und Kreuzungssignal. Der finale Aufbau löste diese Probleme, jedoch entstehen nun die genannten Unebenheiten (Bodenwellen), die teilweise zur fehlerhaften Detektion von Hindernissen führen. Um nicht an den aufgeklebten Kreuzungssignalen hängen zu bleiben, wurde bereits das Frontseitig-angebrachte Schild demontiert. Insbesondere bei der Funktion „Abbiegen“ zeigte sich, dass die Geschwindigkeit der Antriebe von der restlichen Batteriespannung abhängig ist, d.h. ein Fahrzeug mit neuen Energiezellen ist schneller als ein

Fahrzeug mit bereits entladenen Batterien. Beim Abbiegen werden unterschiedliche Geschwindigkeiten für den linken und den rechten Antrieb vorgegeben. Da sich die Änderungen der Geschwindigkeiten und der Batterieladungen nicht in konstanter Abhängigkeit zu einander befinden, erfolgt das Abbiegen in verschiedenen Radien, was es erschwert stets die richtige Spur zu erreichen. Auch der Beschleunigungssensor ist problematisch in der Handhabung. Die Messwerte sind sehr „sprunghaft". Obwohl keine Ursache dafür erkenntlich ist, gibt es immer wieder Ausreißer. Jedoch ist eine Filterung in diesem Fall unpassend, da die realen Werte nur einmalig auftreten, sodass bereits auf die erste Wertänderung reagiert werden muss.

Die Art und Anzahl der Sensoren ist beim Zumo fest vorgegeben. Das Nachrüsten weiterer Sensoren ist nur schwer möglich, da der Hersteller die Platine speziell für die vorhandene Konfiguration entwarf. Beispielweise wurde durch die Verwendung des Bluetooth-Moduls bereits das LCD außer Funktion gesetzt. Für die Angabe der aktuell laufenden Funktion wäre es durchaus nützlich gewesen. Ebenso ist die „werksseitige" Doppelbelegung der Leitungen für Linien- und Abstandssensoren unglücklich, da die zwei nicht verwendeten Liniensensoren zur optimalen Fahrspurerkennung beigetragen hätten. Strebt man die Fortführung von Projekten mit Roboterfahrzeugen an, ist es ratsam trotz des zusätzlichen Aufwands das Fahrzeug selbst nach den jeweiligen Anforderungen des Projekts zu entwerfen oder sich bei Weiternutzung des Zumo nur auf einzelne Funktionen zu konzentrieren.

Unabhängig vom Zumo scheint das Thema „Autonomes Fahren" den Zeitgeist voll und ganz zu treffen. In fast jeder Fachzeitschrift oder allgemeinen Presse ist ein Artikel dazu enthalten: eine weitere Stadt gibt Straßen zu Testzwecken frei [20]; ein Softwarefehler ist schuld am Unfall des Uber-Fahrzeugs [21]; Elektro-Busse fahren autonom durch Hamburg [22]. Da sich die Technik und die Fahrzeuge noch in der Entwicklung befinden, kann es sich als lohnend herausstellen dieses Forschungsziel in näherer Zukunft weiterzuverfolgen. Oft wird das autonome Fahren mit regenerativen Energiesystemen und diversen Vernetzungstechnologien in Verbindung gebracht, die ebenfalls viel Potenzial bieten.

Literatur- und Quellenverzeichnis

[1] Pololu, *Pololu Zumo 32U4 Robot User's Guide,* Las Vegas: Pololu Corporation, 2015.

[2] Ethik-Kommission, Automatisiertes und vernetztes Fahren, Bundesminister für Verkehr und digitale Infrastruktur, 2017.

[3] M. Maurer, C. Gerdes, B. Lenz und H. Winner, *Autonomes Fahren - Technische, rechtliche und gesellschaftliche Aspekte,* Berlin: Springer-Verlag GmbH, 2015.

[4] Society of Automotive Engineers, *J3016 - Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems,* Warrendale: SAE International, 2014.

[5] M. KR, „Wikipedia - Autonomes Fahren," 03 07 2014. [Online]. Available: https://de.wikipedia.org/wiki/Autonomes_Fahren#/media/File: Daimler_2014_Mercedes_Autonomes_Fahren_Magdeburg_5430.jpg. [Zugriff am 20 12 2017].

[6] Daimler, „Autonomer Pionier," Daimler AG, 2017. [Online]. Available: https://www.daimler.com/innovation/autonomes-fahren/mercedes-benz-future-truck.html. [Zugriff am 20 12 2017].

[7] Alphabet, „Waymo Chrysler Pacifica Minivan," Alphabet, 2017. [Online]. Available: https://storage.googleapis.com/sdc-prod/v1/press/waymo_minivan_1.jpg. [Zugriff am 20 12 2017].

[8] Alphabet, „Waymo," Alphabet, 2017. [Online]. Available: https://waymo.com/. [Zugriff am 20 12 2017].

[9] Tesla, „Model S," 2017. [Online]. Available: https://www.tesla.com/tesla_theme/assets/img/compare/model_s--side_profile@2x.png?20170524. [Zugriff am 20 12 2017].

[10] Tesla, „Autopilot," Tesla Deutschland, 2017. [Online]. Available: https://www.tesla.com/de_DE/autopilot. [Zugriff am 20 12 2017].

[11] Atmel, *Atmel-7766H-USB-ATmega16U4-32U4-Datasheet_09/2014,* San Jose: Atmel Corporation, 2014.

[12] D. Finn Hillebrandt, *Arduino Programmier-Handbuch,* Freeduino, 2014.

[13] M. J. B. Wilhelm Burger, Digitale Bildverarbeitung: Eine algorithmische Einführung mit Java, Berlin: Springer Vieweg, 2015.

[14] Pololu Corporation, „Zumo 32U4 Robot," 2017. [Online]. Available: https://a.pololu-files.com/picture/0J6247.1200.jpg?e000b8686ad22c0df670b5e65fd4559d. [Zugriff am 20 12 2017].

[15] Sharp, *D3-A02101EN - GP2S60,* Sharp Corporation, 2005.

[16] Vishay, *TSSP77038 - IR Receiver for Remote Control Systems,* Malvern: Vishay Intertechnology Inc., 2012.

[17] E. Yu, „HC Serial Bluetooth Products - User Instructional Manual," 04 2011. [Online]. Available: https://cdn.makezine.com/uploads/2014/03/hc_hc-05-user-instructions-bluetooth.pdf. [Zugriff am 23 05 2018].

[18] reichelt elektronik, „Arduino HC-05-4," 2017. [Online]. Available: https://cdn-reichelt.de/bilder/web/xxl_ws/A300/HC-05_1.png. [Zugriff am 20 12 2017].

[19] Bundesministerium für Verkehr, Bau und Stadtentwicklung, RAA - Richtlinie für die Anlage von Autobahnen, Bonn, 2008.

[20] WDR, „Selbstfahrende Autos in Aachen schon in drei Jahren," WDR, 15 05 18. [Online]. Available: https://www1.wdr.de/nachrichten/rheinland/autonomes-fahren-aachen-100.html. [Zugriff am 18 05 18].

[21] Autobild, „Tödlicher Softwarefehler," Autobild, 08 05 18. [Online]. Available: http://www.autobild.de/artikel/autonomes-fahren-toedlicher-unfall-von-uber-13410181.html. [Zugriff am 18 05 18].

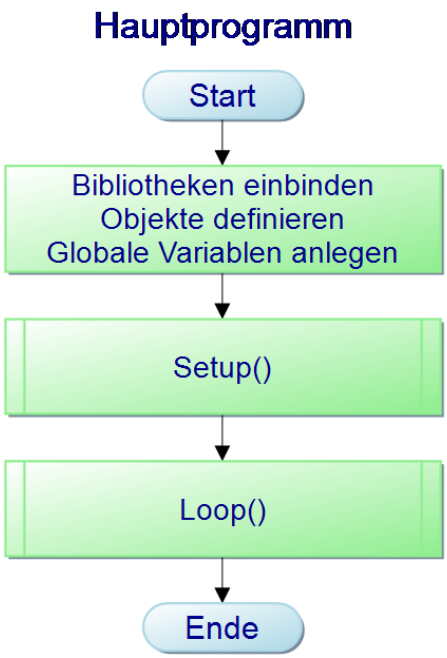
[22] T. Petersen, „Autonome Busse rollen durch die Hafencity," Hannoversche Allgemeine Zeitung, 12 01 18. [Online]. Available: http://www.haz.de/Nachrichten/Der-Norden/Uebersicht/Autonome-Busse-in-Hamburgs-Hafencity. [Zugriff am 18 05 18].

Anlagen

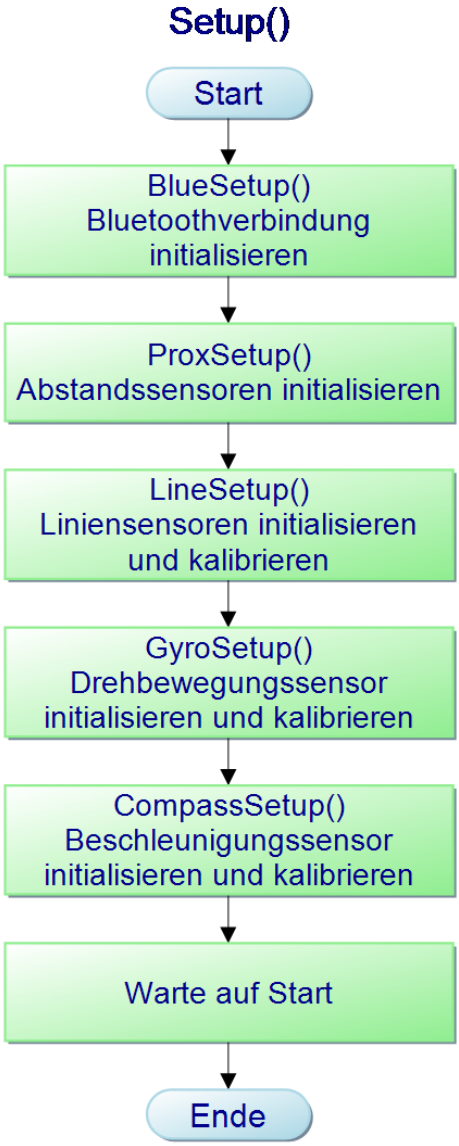
Anlage 1: Programmablaufplan

Anlage 2: Hauptprogramm

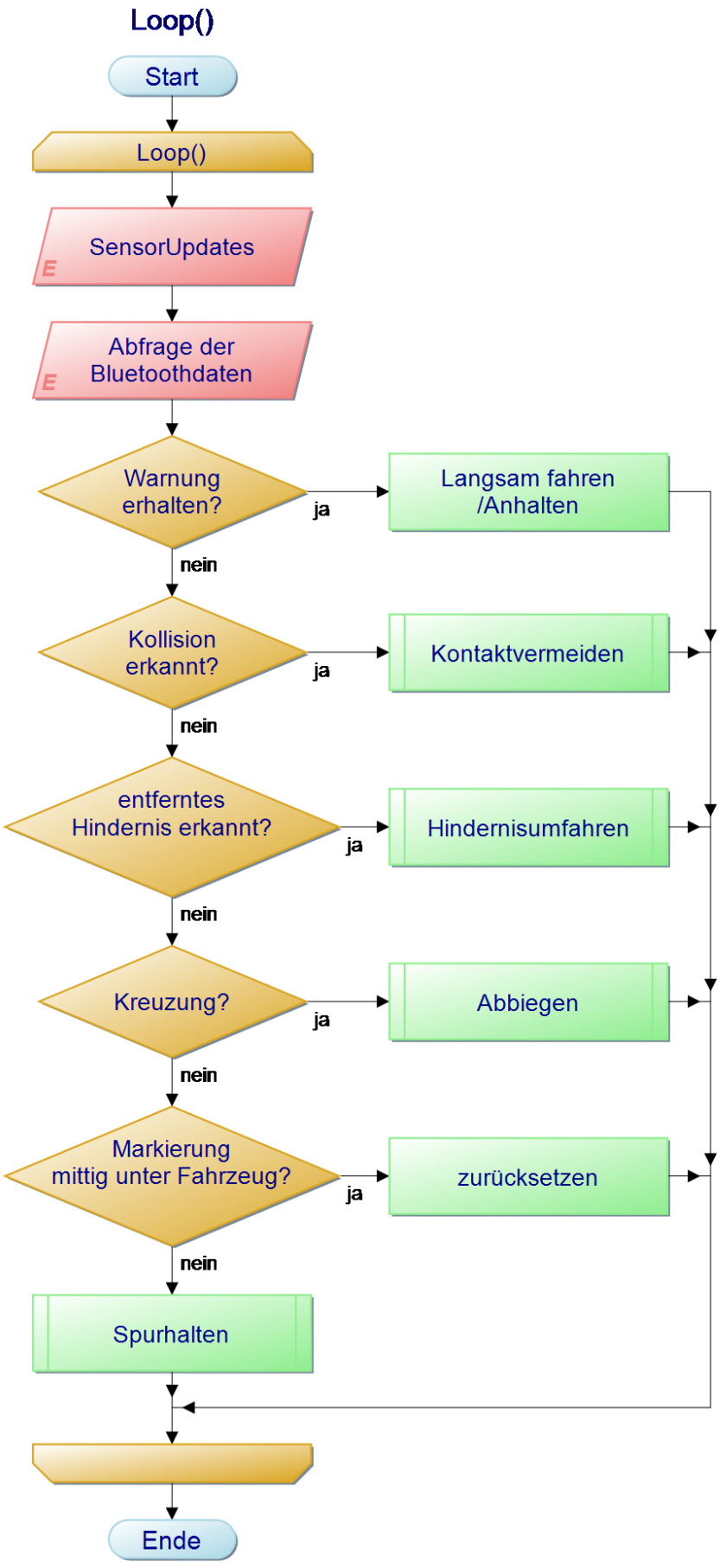
Anlage 1: Programmablaufplan



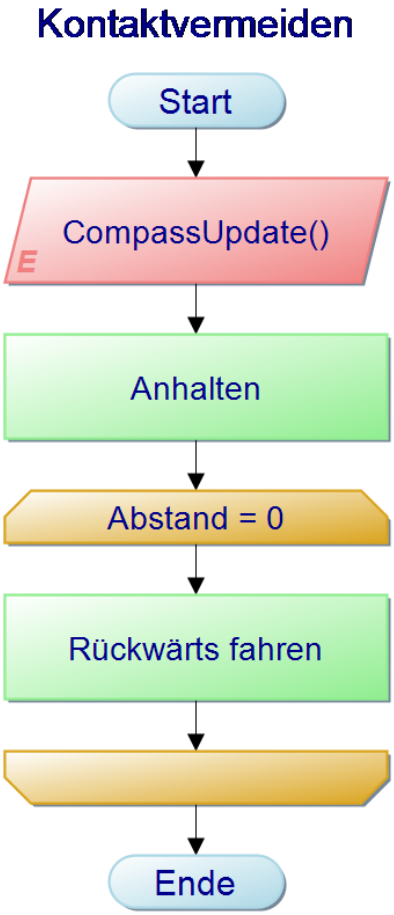
Anlage 1: Programmablaufplan



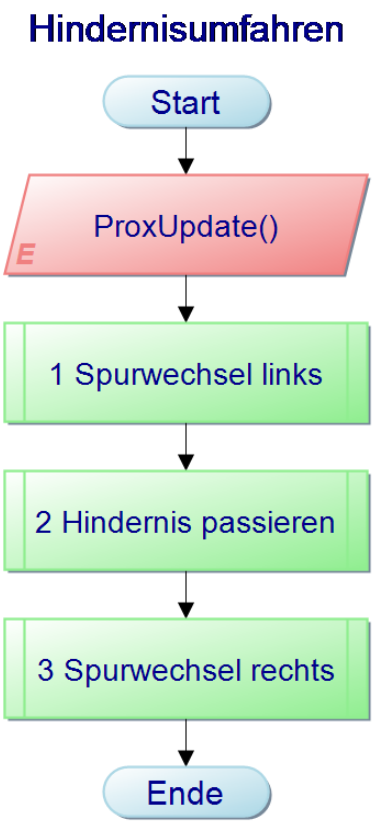
Anlage 1: Programmablaufplan

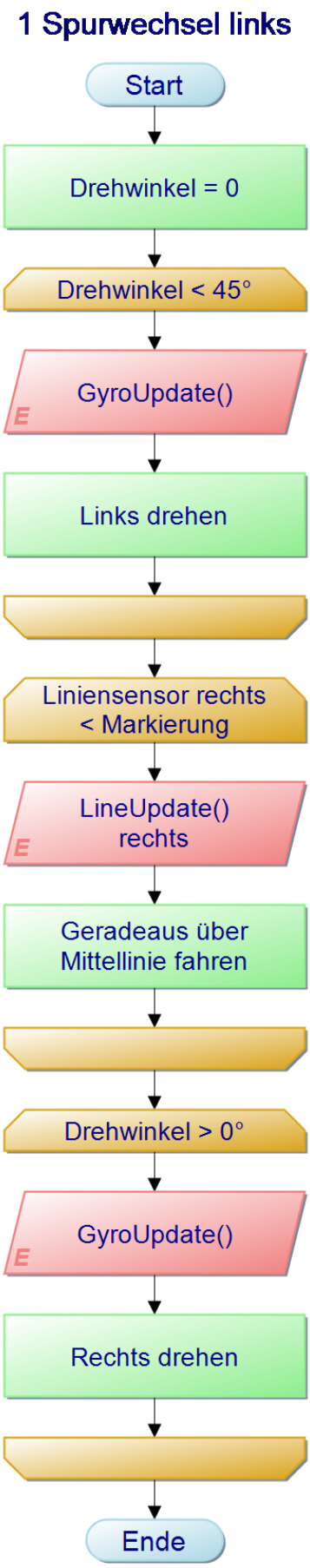


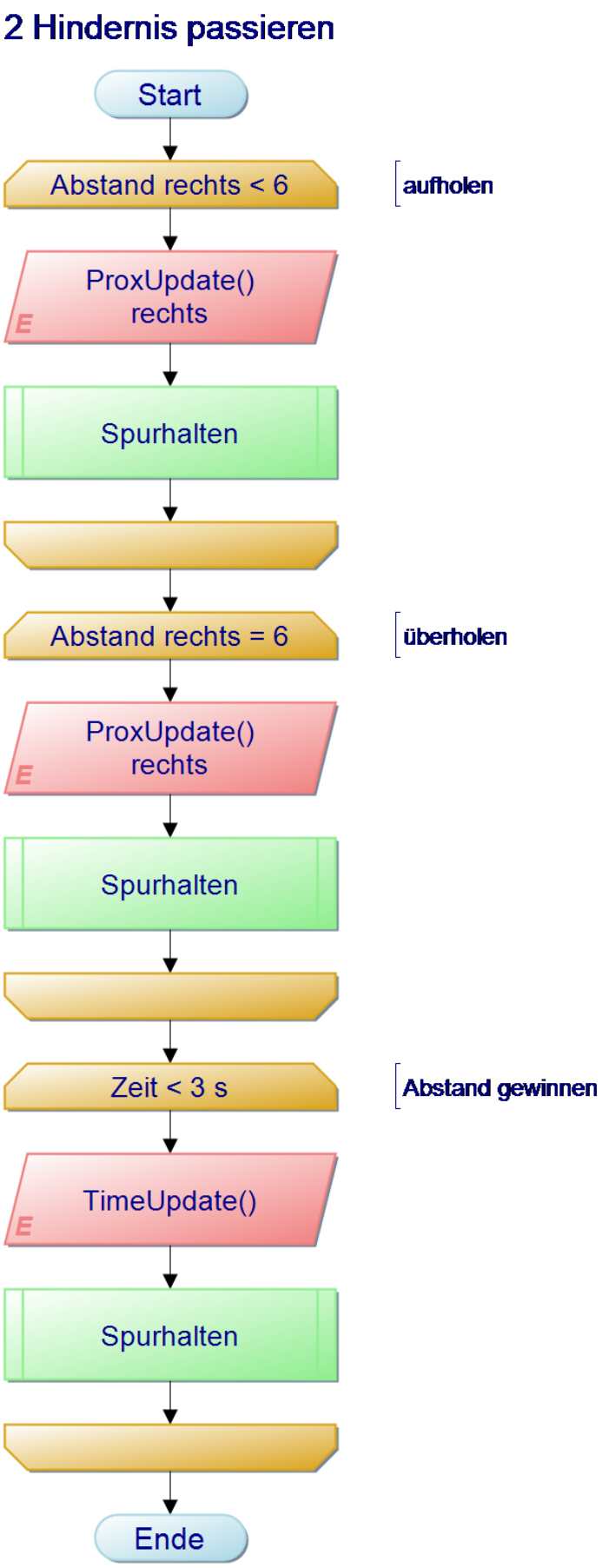
Anlage 1: Programmablaufplan

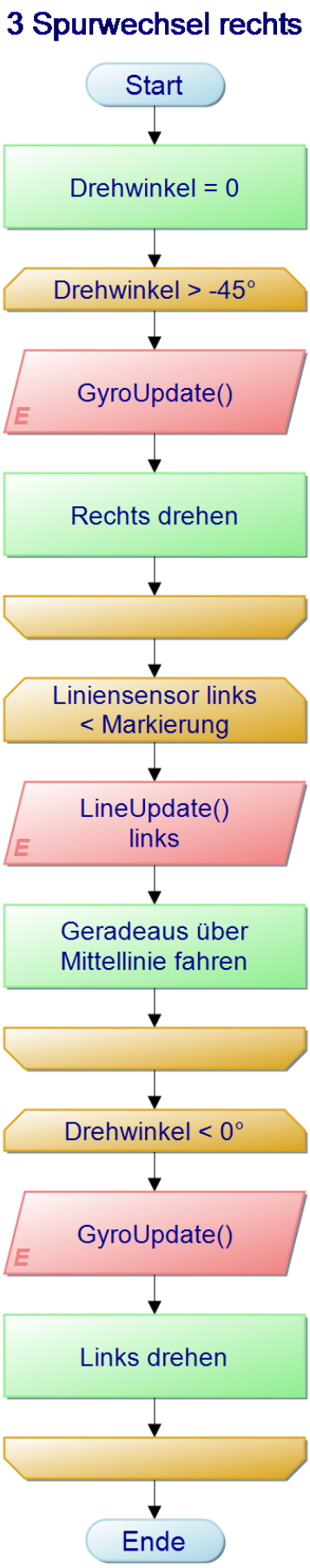


Anlage 1: Programmablaufplan

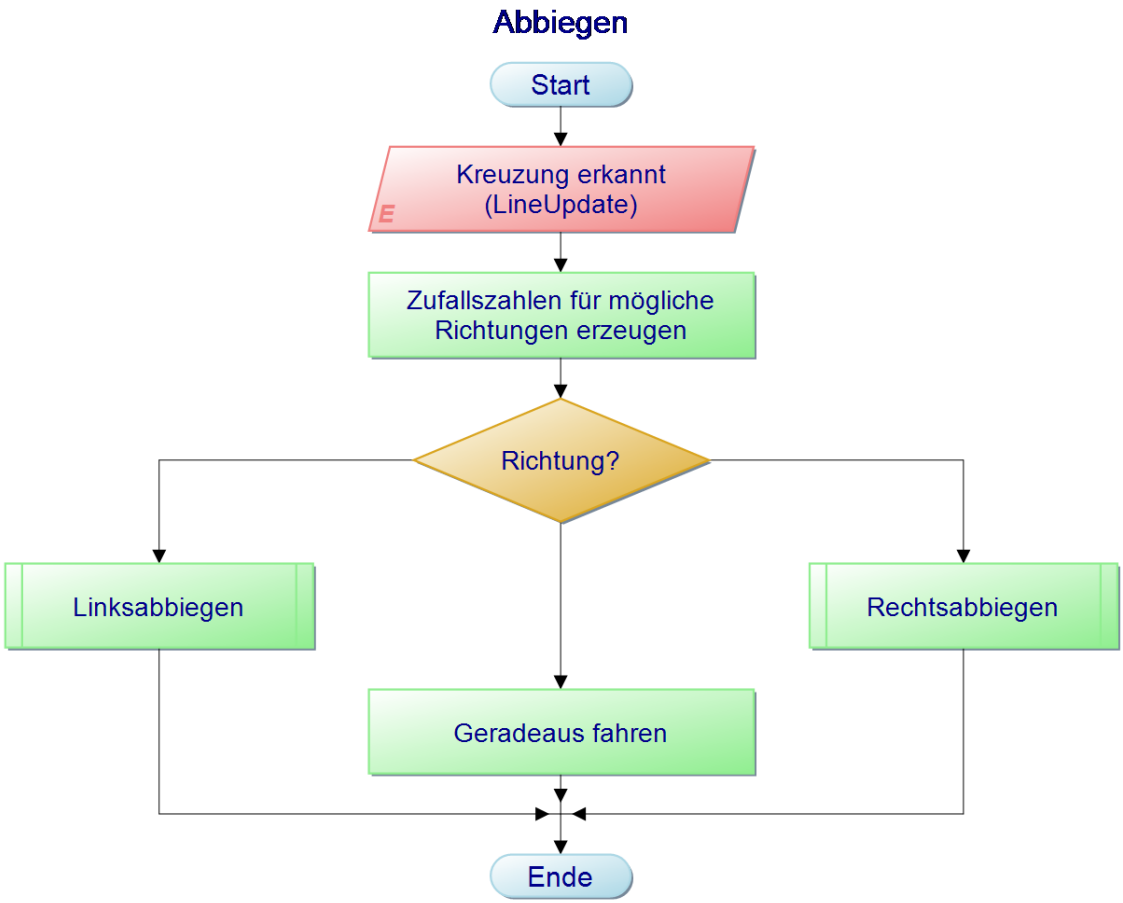


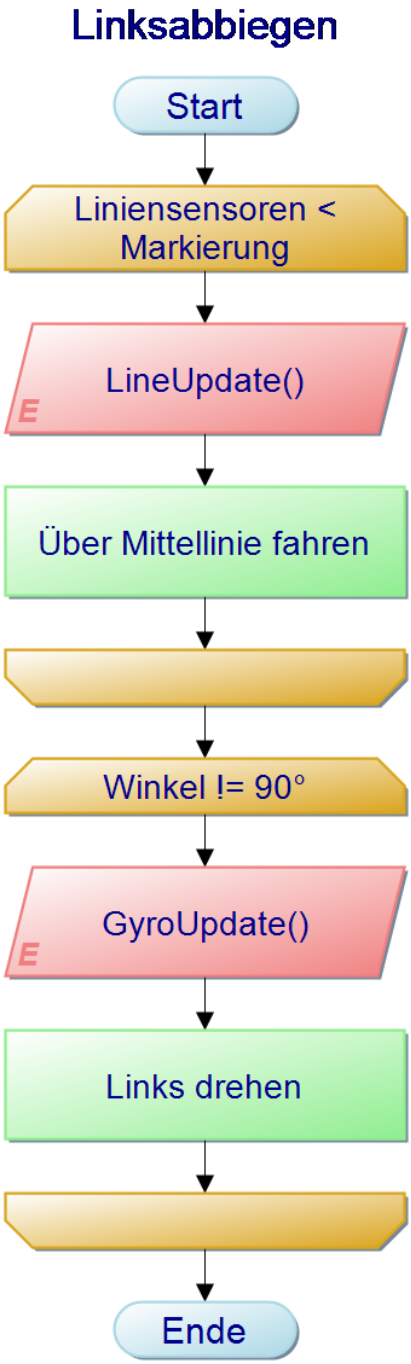

1 Spurwechsel links
Start
Drehwinkel = 0
Drehwinkel < 45°
E
GyroUpdate()
Links drehen
Liniensensor rechts
< Markierung
E
LineUpdate()
rechts
Geradeaus über
Mittellinie fahren
Drehwinkel > 0°
E
GyroUpdate()
Rechts drehen
Ende


2 Hindernis passieren
Start
Abstand rechts < 6
aufholen
ProxUpdate()
rechts
E
Spurhalten
Abstand rechts = 6
überholen
ProxUpdate()
rechts
E
Spurhalten
Zeit < 3 s
Abstand gewinnen
TimeUpdate()
E
Spurhalten
Ende

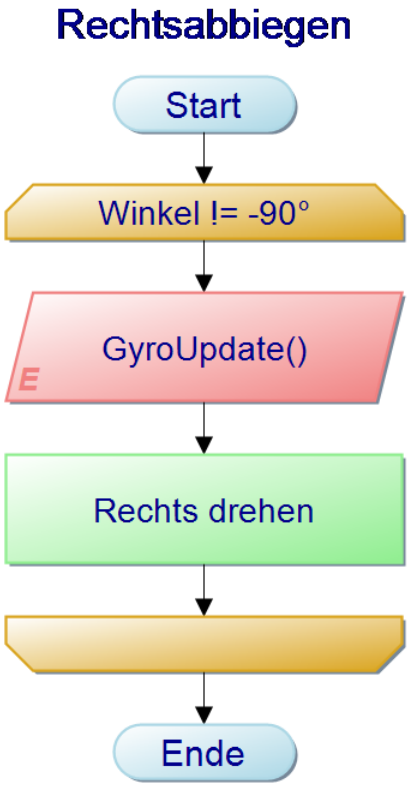

3 Spurwechsel rechts
Start
Drehwinkel = 0
Drehwinkel > -45°
E
GyroUpdate()
Rechts drehen
Liniensensor links
< Markierung
E
LineUpdate()
links
Geradeaus über
Mittellinie fahren
Drehwinkel < 0°
E
GyroUpdate()
Links drehen
Ende

Anlage 1: Programmablaufplan

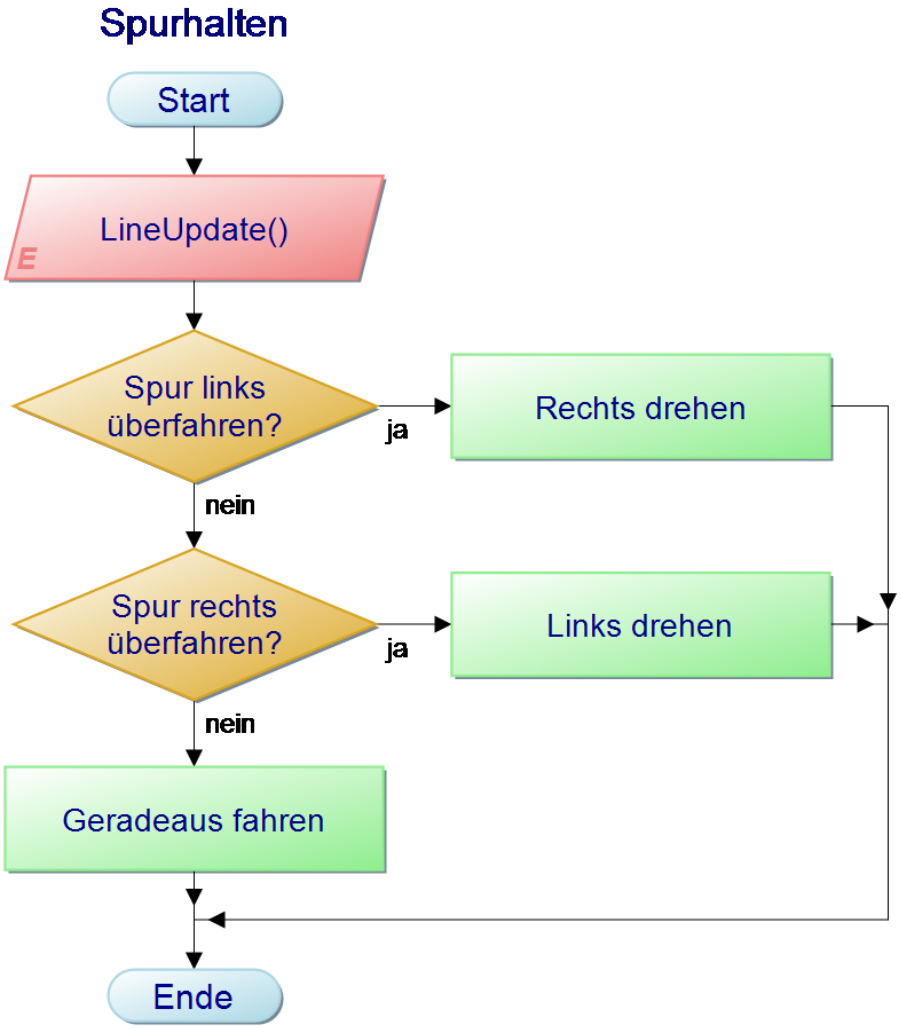



Linksabbiegen
Start
Liniensensoren < Markierung
E
LineUpdate()
Über Mittellinie fahren
Winkel != 90°
E
GyroUpdate()
Links drehen
Ende

Anlage 1: Programmablaufplan



Anlage 1: Programmablaufplan



Anlage 2: Hauptprogramm

```
//Verfassser: Felix Stange, 4056379, MET2016
//Datum: erstellt am 19.07.2017, zuletzt geändert am 18.05.2018
//Projektthema: Untersuchungen und Implementierung eines automatisierten
Fahrens mittels Pololu Zumo
/*Kurzbeschreibung: Der Zumo fährt automatisch zwischen 2 Linien ohne diese zu
überfahren mithilfe
der Liniensensoren (3), ähnlich einer Spurhalteautomatik (heller Belag und
dunkle Streifen).
Außerdem werden Kollisionen verhindert durch Nutzung der vorderen
Abstandssensoren. Kommt es
dennoch zu einer Kollision, wird diese durch den Beschleunigungssensor
(LSM303) erkannt.
Für den Überholvorgang werden die seitlichen Abstandssensoren und der
Drehbewegungssensor (L3G)
genutzt. Mithilfe der Quadraturencoder in den Motoren können Wegstrecken
vermessen werden.
Für die Filterung der Messwerte werden Medianfilter genutzt. Der Zumo
kommuniziert über ein
Bluetooth-Modul (HC-05) mit anderen Geräten. Die Kommunikation erfolgt seriell
('SERIAL' für USB,
'SERIAL1' für Bluetooth). Das LCD kann bei Bluetoothnutzung nicht verwendet
werden.*/

#include  <Wire.h>
#include  <Zumo32U4.h>                  //Copyright Pololu Corporation
(https://github.com/pololu/zumo-32u4-arduino-library)
#include  <MedianFilter.h>              //Copyright Phillip Schmidt
(https://github.com/daPhoosa/MedianFilter)

Zumo32U4ProximitySensors  proxSensors;  //Abstandssensoren
Zumo32U4LineSensors       lineSensors;  //Liniensensoren
Zumo32U4Motors            motors;       //Motoren
Zumo32U4ButtonA           buttonA;      //Taste A
Zumo32U4Encoders          encoders;     //Motorencoder
LSM303                    compass;      //Beschleunigungssensor x-Achse
L3G                       gyro;         //Drehbewegungssensor z-Achse

//Medianfilter geben mittleren Wert einer Reihe mit ungerader Anzahl aus
MedianFilter LineFilter0(3, 0);         //erstellen der Filter für
Liniensensoren
MedianFilter LineFilter1(3, 0);         //Fenstergröße: 3, Basiswerte: 0 0 0
MedianFilter LineFilter2(3, 0);

MedianFilter ProxFilter0(5, 0);         //erstellen der Filter für
Abstandssensoren
MedianFilter ProxFilter1(5, 0);         //Fenstergröße: 5, Basiswerte: 0 0 0 0
0
MedianFilter ProxFilter2(5, 0);
MedianFilter ProxFilter3(5, 0);
```

```
#define   MARKLINE0       150
#define   MARKLINE1       100
#define   MARKLINE2       120
#define   SIGN0           500
#define   SIGN1           300
#define   SIGN2           500

#define   MAXSPEED        400
#define   FULLSPEEDLEFT   106
#define   HALFSPEEDLEFT   54
#define   SLOWSPEEDLEFT   27
#define   FULLSPEEDRIGHT  100
#define   HALFSPEEDRIGHT  50
#define   SLOWSPEEDRIGHT  25

int16_t   lineValue[3];               //Liniensensoren
uint16_t  lineOffset[3];              //von links (0) nach rechts (2)

uint8_t   proxValue[4];               //Abstandssensoren v.l. (0) n.r. (3)

int16_t   encoderCounts[2];           //Anzahl der Umdrehungen
int16_t   driveRotation[2];           //von links (0) nach rechts (1)

int32_t   rotationAngle = 0;          //Drehwinkel
int32_t   turnAngle = 0;
int16_t   turnRate;
int16_t   gyroOffset;
uint16_t  gyroLastUpdate;

int16_t   compassRate;                //Beschleunigung
int16_t   compassOffset;
int16_t   compassLastUpdate;

uint16_t  lastUpdate = 0;             //Systemzeit
uint16_t  eventTime = 0;              //Zeitdifferenz
uint16_t  stopUpdate = 0;             //Systemzeit
uint16_t  stopTime = 0;               //Zeit seit letztem Manöver
float     eventSpeed = 1;             //vermindert die Geschwindigkeit bei
Manövern
int       btData = 0;                 //Gelesene Daten aus Bluetooth
bool      btBuffer = false;           //puffert Daten von btData
bool      stop = false;               //Sperrt Funktion Kontaktvermeiden

/*-----------------------------------------------------------------*/
```

Anlage 2: Hauptprogramm

```
//Setup Bluetoothverbindung
void BlueSetup()
{
  Serial1.begin(9600);                              //Initialisierung mit
Datengeschwindigkeit(Baud)
  Serial1.setTimeout(10);                           //verkürzt Serial(1).read
auf 10 ms statt 1000 ms
  if(Serial1.available()) Serial.println("Bluetoothverbindung hergestellt");
  if(Serial1.available() > 0) Serial1.read();       //Verwerfen der alten
Informationen aus dem Puffer
  Serial1.print(0);
}

//Setup Liniensensoren
void LineSetup()
{
  ledYellow(1);
  lineSensors.initThreeSensors();           //Initialisierung von 3 Sensoren
(max 5)

  //Kalibrierung
  uint32_t total[3] = {0, 0, 0};
  for(uint8_t i = 0; i < 120; i++)
  {
    if (i > 30 && i <= 90) motors.setSpeeds(FULLSPEEDLEFT, -FULLSPEEDRIGHT);
    else motors.setSpeeds(-FULLSPEEDLEFT, FULLSPEEDRIGHT);
    lineSensors.read(lineOffset);
    total[0] += lineOffset[0];
    total[1] += lineOffset[1];
    total[2] += lineOffset[2];
   lineSensors.calibrate();
  }
  motors.setSpeeds(0, 0);
  lineOffset[0] = total[0] / 120;
  lineOffset[1] = total[1] / 120;
  lineOffset[2] = total[2] / 120;
  ledYellow(0);

  // lineOffset[0] = 240;
  // lineOffset[1] = 120;
  // lineOffset[2] = 220;
}

//Setup Abstandssensoren
void ProxSetup()
{
  proxSensors.initThreeSensors();
}
```

Anlage 2: Hauptprogramm

```
//Setup Drehbewegungssensor
void GyroSetup()
{
  ledYellow(1);
  gyro.init();
//Initialisierung
  if(!gyro.init())
//Fehlerabfrage
  {
    //Fehler beim Initialisieren des Drehbewegungssensors
    ledRed(1);
    while(1)
    {
      Serial.println("Fehler Drehbewegungssensors");
      delay(5000);
    }
  }
  gyro.writeReg(L3G::CTRL1, 0b11111111);
//Ausgaberate 800Hz, Tiefpassfilter 100Hz
  gyro.writeReg(L3G::CTRL4, 0b00100000);                    //2000dps
Auflösung
  gyro.writeReg(L3G::CTRL5, 0b00000000);
//Hochpassfilter ausgeschaltet

  //Kalibrierung
  int32_t total = 0;
  for(uint16_t i = 0; i < 1024; i++)
  {
    while(!gyro.readReg(L3G::STATUS_REG) & 0x08);
//Fehlerabfrage
    gyro.read();
    total += gyro.g.z;
  }
  gyroOffset = total / 1024;

  gyroLastUpdate = micros();
  ledYellow(0);
}

//Setup Beschleunigungssensor
void CompassSetup()
{
  ledYellow(1);
  compass.init();
//Initialisierung
  if(!compass.init())
//Fehlerabfrage
  {
    //Fehler beim Initialisieren des Beschleunigungssensors
```

```
    ledRed(1);
    while(1)
    {
      Serial.println("Fehler Beschleunigungssensors");
      delay(5000);
    }
  }
  compass.enableDefault();

  //Kalibrierung
  int32_t total = 0;
  for (uint16_t i = 0; i < 1024; i++)
  {
    compass.readAcc();
    total += compass.a.x;
  }
  compassOffset = total / 1024;

  //compassLastUpdate = micros();
  ledYellow(0);
}

/*-------------------------------------------------------------------*/

void setup()
{
  //Initialisierung der Bluetoothverbindung
  BlueSetup();

  //Initialisierung und Kalibrierung der Sensoren
  //Serial1.println("Sensorkalibrierung");
  Wire.begin();
  delay(500);
  ProxSetup();
  LineSetup();
  GyroSetup();
  CompassSetup();

  //Zumo bereit zu starten
  //Serial1.println("Zumo bereit, starte mit A");
  ledGreen(1);
  buttonA.waitForButton();
  randomSeed(millis());
  delay(500);
  stopUpdate = millis();
  //Serial1.println("Start");
}

/*-------------------------------------------------------------------*/
```

Anlage 2: Hauptprogramm

```
//Update Zeitdifferenzen allgemein
void TimeUpdate()
{
  uint16_t m = millis();
  eventTime = m - lastUpdate;
}

//Update Zeit für Kontaktvermeiden
void StopUpdate()
{
  uint16_t m = millis();
  stopTime = m - stopUpdate;
}

void LoopTiming()           //by GuntherB & Doc_Arduino @ german Arduino Forum
(http://forum.arduino.cc/index.php?topic=292794.0)
{
  const int AL = 100;       // Arraylänge, NUR GERADE Zahlen verwenden!
  static unsigned long LoopTime[AL];
  static unsigned int Index=0, Messung=0, Min=0xFFFF, Max, Avg;

  if (Messung % 2 == 0)     // wenn Messung X gerade (0,2,4,6 usw.),
entspricht immer Anfang der Loop
    {
     LoopTime[Index] = millis();
     Messung++;
     Index++;
     return;                // Funktion sofort beenden, spart etwas Zeit
    }

  if (Messung % 2 == 1)     // wenn Messung X ungerade (1,3,5,7 usw.),
entspricht immer Ende der Loop
    {
     LoopTime[Index] = millis();
     LoopTime[Index-1] = LoopTime[Index] - LoopTime[Index-1];   // Loopdauer
einen Index niedriger einspeichern wie aktuell
     Messung++;
    }

  if (Index >= AL)          // Array voll, Daten auswerten
    {
     for (int i = 0; i<AL; i++)
       {
        Min = min(Min, LoopTime[i]);
        Max = max(Max, LoopTime[i]);
        Avg += LoopTime[i];
       }

     Avg = Avg / AL;
```

```
     Serial1.print(F("Minimal       "));Serial1.print(Min);Serial1.println("
ms  ");
     Serial1.print(F("Durchschnitt  "));Serial1.print(Avg);Serial1.println("
ms  ");
     Serial1.print(F("Maximal       "));Serial1.print(Max);Serial1.println("
ms  ");
     SerielleAusgabe();
     Min = 0xFFFF;
     Max = 0;
     Avg = 0;
     Messung = 0;
     Index = 0;
    }
}

//Funktion Serielle Ausgabe
void SerielleAusgabe()
{
  char report[200];
  snprintf_P(report, sizeof(report),
    PSTR("Abstand: %d %d %d %d   Linie: %d %d %d"),
    proxValue[0], proxValue[1], proxValue[2], proxValue[3],
    lineValue[0], lineValue[1], lineValue[2]);
  Serial1.println(report);
  snprintf_P(report, sizeof(report),
    PSTR("Weg: %d %d   Winkel: %d   Beschleunigung: %d"),
    driveRotation[0], driveRotation[1], rotationAngle, compassRate);
  Serial1.println(report);
}

//Update Abstandssensoren
void ProxUpdate()
{
  bool  done = false;
  int   state = LOW;
  while(done == false)
  {
    //wiederholt abfragen, ob bereits gesendet wird (Dauer je 1000 us bzw. 1
ms)
    if(pulseIn(20, state, 1000) == 0 && pulseIn(22, state, 1000) == 0 &&
pulseIn(4, state, 1000) == 0)
    //if(pulseIn(22, state, 1000) > 0)
    {
      //wenn nichts gesendet wird, selbst Messung durchführen
      proxSensors.read();
      proxValue[0] = ProxFilter0.in(proxSensors.countsLeftWithLeftLeds());
      proxValue[1] = ProxFilter1.in(proxSensors.countsFrontWithLeftLeds());
      proxValue[2] = ProxFilter2.in(proxSensors.countsFrontWithRightLeds());
      proxValue[3] = ProxFilter3.in(proxSensors.countsRightWithRightLeds());
```

```
      done = true;
    }
    else
    {
      //solange fremde Signale empfangen werden, Fehler ausgeben
      Serial.println("Fremdmessung erkannt");
    }
  }
}

//Update Liniensensoren
void LineUpdate()
{
  uint16_t  lineRaw[3];
  lineSensors.read(lineRaw);                                    //lese Messwerte
der Liniensensoren aus
  lineValue[0] = LineFilter0.in(lineRaw[0] - lineOffset[0]);  //ziehe
Offsetwerte von Messwerten ab und gib diese in Filter ein
  lineValue[1] = LineFilter1.in(lineRaw[1] - lineOffset[1]);  //erhält neue
mittlere Werte der Filter
  lineValue[2] = LineFilter2.in(lineRaw[2] - lineOffset[2]);
//"LineFilter.out" um gefilterte Werte auszugeben ohne neue Werte einzugeben
}

//Update Drehbewegungssensor
void GyroUpdate()
{
  gyro.read();                                  //Rohwert 10285 entspricht 90°
bzw.
  turnRate = gyro.g.z - gyroOffset;         //8,75mdps/LSB
  uint16_t m = micros();
  uint16_t dt = m - gyroLastUpdate;
  gyroLastUpdate = m;
  int32_t d = (int32_t)turnRate * dt;
  turnAngle += (int64_t)d * 14680064 / 17578125;
  rotationAngle = (((int32_t)turnAngle >> 16) * 360) >> 16;
}

//Update Beschleunigungssensor
void CompassUpdate()
{
  compass.read();                               //Rohwert 16384 entspricht 1g
(9,81m/s²) bzw. bei +/-2g Messbereich 0,61mg/LSB
  int16_t x = compass.a.x - compassOffset;
  compassRate = x - compassLastUpdate;
  compassLastUpdate = x;
}
```

```
//Update Encoder
void EncoderUpdate()
{
  encoderCounts[0] += encoders.getCountsAndResetLeft();
  driveRotation[0] = encoderCounts[0] / 910;                          //12cpr
(Motorwelle) * 75,81:1 (Getriebe) = 909,7cpr (Antriebszahnrad)
  encoderCounts[1] += encoders.getCountsAndResetRight();
  driveRotation[1] = encoderCounts[1] / 910;
}

/*-----------------------------------------------------------------*/

//Funktion Kontaktvermeiden
void Kontaktvermeiden()
{
  //Serial1.println("Kontaktvermeiden");
  Serial1.print(1);
  ledRed(1);
  motors.setSpeeds(0, 0);
  delay(1000);
  while(proxValue[1] == 0 || proxValue[2] == 0)
  {
    ProxUpdate();
    LineUpdate();
    motors.setSpeeds(-HALFSPEEDLEFT, -HALFSPEEDRIGHT);
    if(lineValue[0] > MARKLINE0 || lineValue[1] > MARKLINE1 || lineValue[2] >
MARKLINE2) break;
  }
  lastUpdate = millis();
  TimeUpdate();
  while(eventTime < 1000)
  {
    TimeUpdate();
    LineUpdate();
    motors.setSpeeds(-HALFSPEEDLEFT, -HALFSPEEDRIGHT);
    if(lineValue[0] > MARKLINE0 || lineValue[1] > MARKLINE1 || lineValue[2] >
MARKLINE2) break;
  }
  motors.setSpeeds(0, 0);
  compassLastUpdate = 0;
  compassRate = 0;
  CompassUpdate();
  stop = false;
  stopUpdate = millis();
  //Serial1.println("Vermeiden beendet");
  Serial1.print(0);
}
```

Anlage 2: Hauptprogramm

```
//Funktion Hindernisumfahrung
void Hindernisumfahren()
{
  //Serial1.println("Hindernisumfahren");
  Serial1.print(1);
  ledYellow(1);

  //Schritt 1: Spurwechsel links
  //links drehen
  turnAngle = 0;
  rotationAngle = 0;
  GyroUpdate();
  while(rotationAngle < 20)
  {
    GyroUpdate();
    LineUpdate();
    motors.setSpeeds(SLOWSPEEDLEFT, FULLSPEEDRIGHT);
  }
  GyroUpdate();
  while(rotationAngle < 45)
  {
    GyroUpdate();
    LineUpdate();
    motors.setSpeeds(SLOWSPEEDLEFT, FULLSPEEDRIGHT);
    if(lineValue[2] > MARKLINE2 && lineValue[2] < SIGN2) break;
  }

  //geradeaus über Mittellinie fahren
  LineUpdate();
  while(lineValue[2] < MARKLINE2)
  {
    LineUpdate();
    motors.setSpeeds(FULLSPEEDLEFT, FULLSPEEDRIGHT);
  }

  //rechts drehen
  GyroUpdate();
  while(rotationAngle > 10)
  {
    GyroUpdate();
    LineUpdate();
    if(lineValue[0] > MARKLINE0 && lineValue[0] < SIGN0)
motors.setSpeeds(FULLSPEEDLEFT, 0);
    else if(lineValue[2] > MARKLINE2 && lineValue[2] < SIGN2)
motors.setSpeeds(FULLSPEEDLEFT, FULLSPEEDRIGHT);
    else motors.setSpeeds(FULLSPEEDLEFT, SLOWSPEEDRIGHT);
  }

  //geradeaus fahren
```

```
motors.setSpeeds(FULLSPEEDLEFT, FULLSPEEDRIGHT);

//Gegenverkehr beachten
ProxUpdate();
//if(proxValue[1] == 6 && proxValue[2] == 6)
//{
  //Schritt 2: Hindernis passieren
  //Serial1.println("Aufholen");
  lastUpdate = millis();
  eventSpeed = 0.8;
  while(proxValue[3] < 6)
  {
    ProxUpdate();
    LineUpdate();
    Spurhalten();
    TimeUpdate();
    //Serial1.println(eventTime);
    if(eventTime > 3000) break;
  }
  //Serial1.println("Vorbeifahren");
  ProxUpdate();
  while(proxValue[3] == 6)
  {
    ProxUpdate();
    LineUpdate();
    Spurhalten();
  }
  //Serial1.println("Abstand gewinnen");
  lastUpdate = millis();
  TimeUpdate();
  while(eventTime < 3000)
  {
    LineUpdate();
    Spurhalten();
    TimeUpdate();
    //Serial1.println(eventTime);
  }
  eventSpeed = 1.0;
//}

//Schritt 3: Spurwechsel rechts
//rechts drehen
turnAngle = 0;
rotationAngle = 0;
GyroUpdate();
while(rotationAngle > -20)
{
  GyroUpdate();
  LineUpdate();
```

```
    motors.setSpeeds(FULLSPEEDLEFT, SLOWSPEEDRIGHT);
  }
  GyroUpdate();
  while(rotationAngle > -45)
  {
    GyroUpdate();
    LineUpdate();
    motors.setSpeeds(FULLSPEEDLEFT, SLOWSPEEDRIGHT);
    if(lineValue[0] > MARKLINE0 && lineValue[0] < SIGN0) break;
  }

  //geradeaus über Mittellinie fahren
  LineUpdate();
  while(lineValue[0] < MARKLINE0)
  {
    LineUpdate();
    motors.setSpeeds(FULLSPEEDLEFT, FULLSPEEDRIGHT);
  }

  //links drehen
  GyroUpdate();
  while(rotationAngle < -10)
  {
    GyroUpdate();
    LineUpdate();
    if(lineValue[2] > MARKLINE2 && lineValue[2] < SIGN2) motors.setSpeeds(0,
FULLSPEEDRIGHT);
    else if(lineValue[0] > MARKLINE0 && lineValue[0] < SIGN0)
motors.setSpeeds(FULLSPEEDLEFT, FULLSPEEDRIGHT);
    else motors.setSpeeds(SLOWSPEEDLEFT, FULLSPEEDRIGHT);
  }

  //geradeaus fahren
  //Serial1.println("Umfahren beendet");
  motors.setSpeeds(FULLSPEEDLEFT, FULLSPEEDRIGHT);
  stop = false;
  stopUpdate = millis();
  Serial1.print(0);
}

//Funktion Abbiegen
void Abbiegen()
{
  ledYellow(1);
  //Serial1.println("Abbiegen");
  Serial1.print(1);

  //Markierung analysieren
  LineUpdate();
```

```
  bool  leftCode;
//links => 1
  bool  rightCode;
//rechts => 2
  if((lineValue[0] > SIGN0 && lineValue[0] < 1600) && (lineValue[1] > SIGN1 &&
lineValue[1] < 800)) leftCode = true;
  else leftCode = false;
  if((lineValue[2] > SIGN2 && lineValue[2] < 1600) && (lineValue[1] > SIGN1 &&
lineValue[1] < 800)) rightCode = true;
  else rightCode = false;

  //Zufallszahl erzeugen
  uint8_t randy;
//geradeaus => 0
  if(leftCode == true && rightCode == true) randy = random(1, 3);          //1,
2
  else if(leftCode == true && rightCode == false) //randy = 1;
  {
    randy = random(0, 2);                                                  //0,
1
    //if(randy == 0) randy = random(0, 2);
//erhöht Wahrscheinlickeit abzubiegen
  }
  else if(leftCode == false && rightCode == true) //randy = 2;
  {
    randy = random(0, 2);                                                  //0,
1
    //if(randy == 0) randy = random(0, 2);
//erhöht Wahrscheinlickeit abzubiegen
    if(randy == 1) randy = 2;                                              //!1
=> 2
  }

  //links Abbiegen (von der Verbindungsstrecke)
  if(randy == 1 && rightCode == true)
  {
    //Serial1.println("links Abbiegen von der Verbindungsstrecke");
    //geradeaus zur Mittellinie fahren
    motors.setSpeeds(FULLSPEEDLEFT, FULLSPEEDRIGHT+10);
    lastUpdate = millis();
    TimeUpdate();
    while(eventTime < 1000)
    {
      TimeUpdate();
      motors.setSpeeds(FULLSPEEDLEFT, FULLSPEEDRIGHT+10);
    }
    LineUpdate();
    while(lineValue[0] < MARKLINE0 && lineValue[2] < MARKLINE2)
    {
```

```
      LineUpdate();
      motors.setSpeeds(FULLSPEEDLEFT, FULLSPEEDRIGHT+10);
    }

    //links drehen
    turnAngle = 0;
    rotationAngle = 0;
    GyroUpdate();
    while(rotationAngle < 90)
    {
      GyroUpdate();
      LineUpdate();
      if(lineValue[2] > MARKLINE2 && lineValue[2] < SIGN2) motors.setSpeeds(0,
FULLSPEEDRIGHT);
      else if(lineValue[0] > MARKLINE0 && lineValue[0] < SIGN0)
motors.setSpeeds(HALFSPEEDLEFT, FULLSPEEDRIGHT);
      else motors.setSpeeds(SLOWSPEEDLEFT, FULLSPEEDRIGHT);
    }

    //geradeaus fahren
    motors.setSpeeds(FULLSPEEDLEFT, FULLSPEEDRIGHT);
  }

  //links Abbiegen (vom Rundkurs)
  else if(randy == 1 && leftCode == true)
  {
    //Serial1.println("links Abbiegen vom Rundkurs");
    //links drehen
    motors.setSpeeds(SLOWSPEEDLEFT, FULLSPEEDRIGHT);
    turnAngle = 0;
    rotationAngle = 0;
    GyroUpdate();
    while(rotationAngle < 40)
    {
      GyroUpdate();
      motors.setSpeeds(SLOWSPEEDLEFT, FULLSPEEDRIGHT);
    }
    GyroUpdate();
    while(rotationAngle < 85)
    {
      GyroUpdate();
      motors.setSpeeds(SLOWSPEEDLEFT, FULLSPEEDRIGHT);
    }

    //geradeaus fahren
    motors.setSpeeds(FULLSPEEDLEFT, FULLSPEEDRIGHT);
    lastUpdate = millis();
    TimeUpdate();
    while(eventTime < 1000)
```

```
    {
      TimeUpdate();
      LineUpdate();
      motors.setSpeeds(FULLSPEEDLEFT, FULLSPEEDRIGHT);
      if(lineValue[2] > MARKLINE2 && lineValue[2] < SIGN2) break;
    }
    lastUpdate = millis();
    TimeUpdate();
    while(eventTime < 1000)
    {
      TimeUpdate();
      LineUpdate();
      Spurhalten();
    }
  }

  //rechts Abbiegen
  else if(randy == 2 && rightCode == true)
  {
    //Serial1.println("rechts Abbiegen");
    //rechts drehen
    motors.setSpeeds(FULLSPEEDLEFT, SLOWSPEEDRIGHT);
    turnAngle = 0;
    rotationAngle = 0;
    GyroUpdate();
    while(rotationAngle > -40)
    {
      GyroUpdate();
      LineUpdate();
      motors.setSpeeds(FULLSPEEDLEFT, SLOWSPEEDRIGHT);
      if(lineValue[0] > MARKLINE0 && lineValue[0] < SIGN0) break;
    }
    GyroUpdate();
    lastUpdate = millis();
    while(rotationAngle > -85)
    {
      TimeUpdate();
      GyroUpdate();
      LineUpdate();
      if(eventTime > 3000) break;
      if(lineValue[0] > MARKLINE0 && lineValue[0] < SIGN0)
motors.setSpeeds(FULLSPEEDLEFT, 0);
      //else if(lineValue[1] > MARKLINE1 && lineValue[1] < SIGN1)
motors.setSpeeds(-SLOWSPEEDLEFT, -SLOWSPEEDRIGHT);
      else if(lineValue[2] > MARKLINE2 && lineValue[2] < SIGN2)
motors.setSpeeds(FULLSPEEDLEFT, HALFSPEEDRIGHT);
      else motors.setSpeeds(FULLSPEEDLEFT, SLOWSPEEDRIGHT);
    }
  }
```

```
  //nicht Abbiegen, geradeaus fahren
  else
  {
    //Serial1.println("nicht Abbiegen");
    motors.setSpeeds(FULLSPEEDLEFT, FULLSPEEDRIGHT);
    lastUpdate = millis();
    TimeUpdate();
    while(eventTime < 1000)
    {
      TimeUpdate();
      LineUpdate();
      Spurhalten();
    }
  }
  stop = false;
  stopUpdate = millis();
  //Serial1.println("Abbiegen beendet");
  Serial1.print(0);
}

//Funktion Spurhalten
void Spurhalten()
{
  uint16_t StartTime = millis();
  uint16_t Update = millis();
  uint16_t Time = Update - StartTime;

  //linke Linie erreicht, nach rechts fahren
  if(lineValue[0] > MARKLINE0 && lineValue[0] < SIGN0)
  {
    ledYellow(1);
    //Serial1.println("Spur nach rechts korrigieren");
    motors.setSpeeds((int)FULLSPEEDLEFT/eventSpeed,
(int)SLOWSPEEDRIGHT/eventSpeed);
    while(Time < 100)
    {
      Update = millis();
      Time = Update - StartTime;
      LineUpdate();
      if(lineValue[2] > MARKLINE2) break;
    }
    stop = false;
    stopUpdate = millis();
  }

  //rechte Linie erreicht, nach links fahren
  else if(lineValue[2] > MARKLINE2 && lineValue[2] < SIGN2)
  {
    ledYellow(1);
```

```
    //Serial1.println("Spur nach links korrigieren");
    motors.setSpeeds((int)SLOWSPEEDLEFT/eventSpeed,
(int)FULLSPEEDRIGHT/eventSpeed);
    while(Time < 100)
    {
      Update = millis();
      Time = Update - StartTime;
      LineUpdate();
      if(lineValue[0] > MARKLINE0) break;
    }
    stop = false;
    stopUpdate = millis();
  }

  //normale Fahrt
  else
  {
    ledGreen(1);
    motors.setSpeeds((int)FULLSPEEDLEFT/eventSpeed,
(int)FULLSPEEDRIGHT/eventSpeed);
    stop = true;
  }
}

//Funktion Spurfinden
void Spurfinden()
{
  ledRed(1);
  //Serial1.println("Spurfinden");
  Serial1.print(1);
  lastUpdate = millis();
  while(lineValue[0] < MARKLINE0 && lineValue[2] < MARKLINE2)
//Zumo fährt solange rückwärst bis er wieder auf der Spur steht
  {
    TimeUpdate();
    LineUpdate();
    motors.setSpeeds(-FULLSPEEDLEFT, -FULLSPEEDRIGHT);
    if(eventTime > 3000) break;
  }
  stop = false;
  stopUpdate = millis();
  //Serial1.println("Spur gefunden");
  Serial1.print(0);
}

/*-----------------------------------------------------------------*/
```

Anlage 2: Hauptprogramm

```
void loop()
{
  //LoopTiming();                                  //Zykluszeit beträgt max. 20ms
im Idle
  ledGreen(0);
  ledYellow(0);
  ledRed(0);

  //Abfragen der Sensordaten
  ProxUpdate();
  EncoderUpdate();
  GyroUpdate();
  CompassUpdate();
  LineUpdate();
  TimeUpdate();
  StopUpdate();

  //Funktionsauswahl
  if(Serial1.available() > 0) btData = Serial1.read();
  if(btData == '1') btBuffer = true;
  else if(btData == '0') btBuffer = false;

  //verfügbare Funktionen bei laufenden Manövern
  if(btBuffer == true)
  {
    //Serial1.println("Verstanden");
    eventSpeed = 1.4;
    if(proxValue[0] == 6 || (proxValue[1] == 6 && proxValue[2] == 6))
    {
      lastUpdate = millis();
      TimeUpdate();
      while(eventTime < 1000)
      {
        TimeUpdate();
        motors.setSpeeds(0, 0);
      }
    }
    else if((stop == true && stopTime > 2000 && abs(compassRate) > 3500) ||
    (lineValue[0] > SIGN0 || lineValue[2] > SIGN2 || lineValue[1] >
MARKLINE1)) motors.setSpeeds(0, 0);
    else Spurhalten();
  }

  //verfügbare Funktionen im Normalfall
  else
  {
    eventSpeed = 1.0;
    if(stop == true && stopTime > 2000 && abs(compassRate) > 3500)
Kontaktvermeiden();
```

```
    else if(proxValue[1] == 6 && proxValue[2] == 6) Hindernisumfahren();
    else if(lineValue[0] > 1600 && lineValue[2] > 1600) motors.setSpeeds(0,
0);
    else if(((lineValue[0] > SIGN0 && lineValue[0] < 1600) && (lineValue[1] >
SIGN1 && lineValue[1] < 800)) ||
    ((lineValue[2] > SIGN2 && lineValue[2] < 1600) && (lineValue[1] > SIGN1 &&
lineValue[1] < 800))) Abbiegen();
    else if(lineValue[1] > MARKLINE1 && lineValue[1] < SIGN1) Spurfinden();
    else Spurhalten();
  }
  //LoopTiming();
}
```