LEGO Racers

Studienarbeit

im Themengebiet LEGO Mindstorms

an der Fakultät für Technik
im Studiengang Informationstechnik

an der
DHBW Ravensburg

Verfasser:	Lara Kroesen (9377505)
	Jens Müller (6881478)
	Jan Herkommer (3218132)
Wiss. Betreuer:	Peter Firmkaes
Abgabedatum:	11.01.16 (erster Stand)

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Projektziel

Das Ziel des Lego Racers Projekts ist es, mithilfe von LEGO Mindstorm Bausteinen autonom fahrende Rennwagen zu realisieren. Diese sollen eine vorgegebene Strecke in möglichst kurzer Zeit zu befahren. Es musss möglich sein, eine Strecke mit mehreren Fahrzeugen kollisionsfrei zu durchfahren. Zusätzlich soll eine Möglichkeit bestehen, die Fahrzeuge per Smartphone zu steuern um verschiedene Modi zu ermöglichen. Ziel ist zunächst der Bau eines Fahrzeuges, das auf optimales Befahren der Strecke ausgelegt ist. Anschließend soll das Projekt um weitere Fahrzeuge erweitert werden, die die Funktionalitäten des Prototyps übernehmen und zusätzlich Kollisionsvermeidung ermöglichen sollen.

2 Entwurfsphase

2.1 Software-Requirements

2.1.1 High Level Requirements

Requirement-ID	Beschreibung
HLR-01	Autonomes Befahren einer vorgegebenen Strecke (ohne andere Autos)
HLR-02	Kollisionsvermeidung und Kollisionshandling zwischen den Fahrzeugen und Hindernissen
HLR-04	Verschiedener Rennmodi müssen existieren und über eine Smartphone App konfigurierbar sein
HLR-05	Smartphone App zur Bedienung der Fahrzeuge
HLR-06	Realisierung von Fahrsicherheitssystemen

2.1.2 Low Level Requirements

Requirement-ID	Beschreibung
HLR-01	
LLR-01-1	Erkennen der vorgegebenen Strecke
LLR-01-1-1	Erkennen von geraden Strecken und das damit verbundene Erkennen der Grenze und Richtung der geraden Strecke
LLR-01-1-2	Erkennen von Kurven, erkennen des Kurveneintrittspunktes sowie des Austrittpunktes. Berechnung der optimalen Linie und maximalen Geschwindigkeit anhand der gegebenen Daten.
LLR-01-1-3	Speichern der durchfahrenen Strecke um weitere Umläufe zu optimieren. Analyse der Rückmeldungen der Fahrzeugsysteme und Events wie z.B. abkommen von der Strecke, Berechnung optimaler Parameter für bestimmte Kurven o.Ä..
LLR-01-1-4	Messung der Umlaufzeit, Rückmeldung an Smartphone/Ausgabe auf Display
LLR-01-2	Erkennen von Störungen
LLR-01-2-1	Erkennen von Hindernissen auf der Strecke und Berechnung passender Ausweichmöglichkeiten
LLR-01-2-2	Erkennen von Traktionsproblemen wie z.B. rutschen in Kurven und Berechnung geeigneter Gegenmaßnahmen bzw. Wiedereinordnung auf Strecke
LLR-01-2-3	Erkennen der Fahrtrichtung, im Falle eines Fehlers muss das Fahrzeug wenden um in den normalen Fahrtmodus zurückzukehren

HLR-02	
LLR-02-1	Detektierung von Fahrzeugen in der Nähe und Berechnung der Richtungsänderung zur Kollisionsvermeidung falls Kollisionsgefahr besteht
LLR-02-2	Im Falle einer Kollision muss der Kontakt zum anderen Fahrzeug abgebrochen werden und das Fahrzeug wieder in den normalen Fahrmodus übergehen
LLR-02-3	Überholmanöver müssen kollisionsfrei möglich sein
HLR-04	
LLR-04-1	Rennmodi müssen anhand von Parametern einstellbar sein
LLR-04-2	Freie Fahrt: Das Fahrzeug fährt den Rundkurs bestmöglich ab, ohne spezielle Regelungen
LLR-04-2	Qualifikation: Fahrzeug hat eine vorgegebene Anzahl von Runden und fährt bestmögliche Rundenzeit. Die beste Rundenzeit wird entsprechend ausgegeben. Bei mehreren Fahrzeugen kann automatisch die Startposition eingenommen werden.
LLR-04-3	Rennen: Rennen wird mit einem Signal gestartet. Fahrzeuge fahren gegeneinander, nach einer bestimmten Anzahl von Runden wird das Rennen beendet.
LLR-04-3-1	Startsignal: Extern ausgelöstes Signal, welches gleichzeitig bei allen Fahrzeugen den Start auslöst
LLR-04-3-2	Reifenabnutzung: Im Laufe eines Rennens kann, wenn so konfiguriert. das Fahrzeug langsamer werden. Mit einem Boxenstopp(durch Befehl von außen Ausgeführt) kann die ursprüngliche Geschwindigkeit wieder hergestellt werden. Der Befehl kann entweder vor dem Rennen eingegeben (Strategie, Boxenstopp in Runde X) oder während dem Rennen per Befehl durch die Smartphone App ausgelöst werden.
LLR-04-4	Crash Race: Kollisionsabfrage abgeschaltet, Autos sollen sich rammen
HLR-05	
LLR-05-1	Smartphone-App muss in der Lage sein, Kontakt zum Fahrzeug aufzunehmen
LLR-05-2	App muss die Möglichkeit bieten, die Rennmodi auszuwählen
LLR-05-3	App muss die Rückmeldungen vom Fahrzeug wie z.B. Rundenzeiten erhalten und anzeigen können
LLR-05-4	App soll in der Lage sein, das Fahrzeug manuell zu steuern

HLR-06	
LLR-06-1	ABS soll realisiert werden, um blockieren der Reifen beim anbremsen zu verhindern. Außerdem soll das System bei der Optimierung der Rundenzeiten eingesetzt werden.
LLR-06-2	ESP soll realisiert werden, um schleudern des Fahrzeugs bei unterschiedlichen Untergründen zu vermeiden. Außerdem soll das System bei der Optimierung der Rundenzeiten eingesetzt werden.
LLR-06-3	Es soll eine Traktionskontrolle realisiert werden, um Haftungsverlust bei Beschleunigung zu vermeiden. Außerdem soll das System bei der Optimierung der Rundenzeiten eingesetzt werden.

2.2 Streckengestaltung

Die Strecke wird dem Fahrzeug mithilfe von Farbcodierungen mitgeteilt.

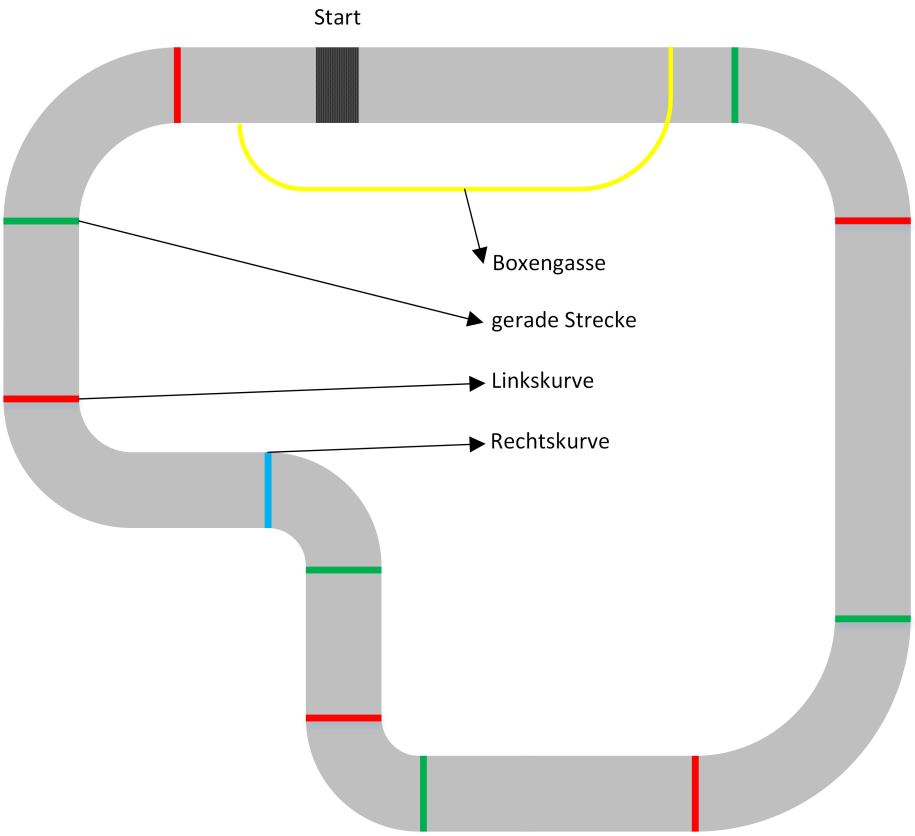

Abb. 2.1: Erste Version der Rennstrecke

Ursprünglich war geplant, dass rote Linien für Linkskurven stehen, blaue für Rechtskurven und grüne Linien eine Gerade bzw. ein Ende des aktuellen Segments (z.B. Boxengasse) symbolisieren. Die Gelbe Linie symbolisiert die Boxengasse. Die Außenlinie sollte durch eine einfarbige Fläche symbolisiert werden, da dies die Rückkehr auf die Strecke bei versehentlichem Verlassen der Strecke erleichtert.

Nach ersten Farbsensortests und Vorüberlegungen zum Ablauf der Steuerung fiel jedoch die Entscheidung, die Farbcodierungen deutlich zu vereinfachen. Die Außenlinien der Strecke sollten durch weiße Linien (z.B. Klebeband) verdeutlicht werden. Dadurch lässt sich eine beliebige Rennstrecke einfach konstruieren, in dem die Strecke mit weißem Klebeband auf schwarzem Untergrund abgeklebt wird. Dies erschwert zwar nach versehentlichem Verlassen der Strecke die Rückkehr, ist allerdings deutlich leichter und flexibler umzusetzen und spiegelt eher die Realität wieder. Des weiteren reicht es, wenn die Kurven-Eingänge und -Ausgänge mit einer Farbe markiert werden, egal ob Links- oder Rechtskurve. Die Erkennung der Kurven erfolgt dann durch ein sogenannte Einführungsrunde, in der alle Kurven erfasst werden. Außerdem kann das Auto durch diese Linien

kontrollieren, wie es momentan ausgerichtet ist.

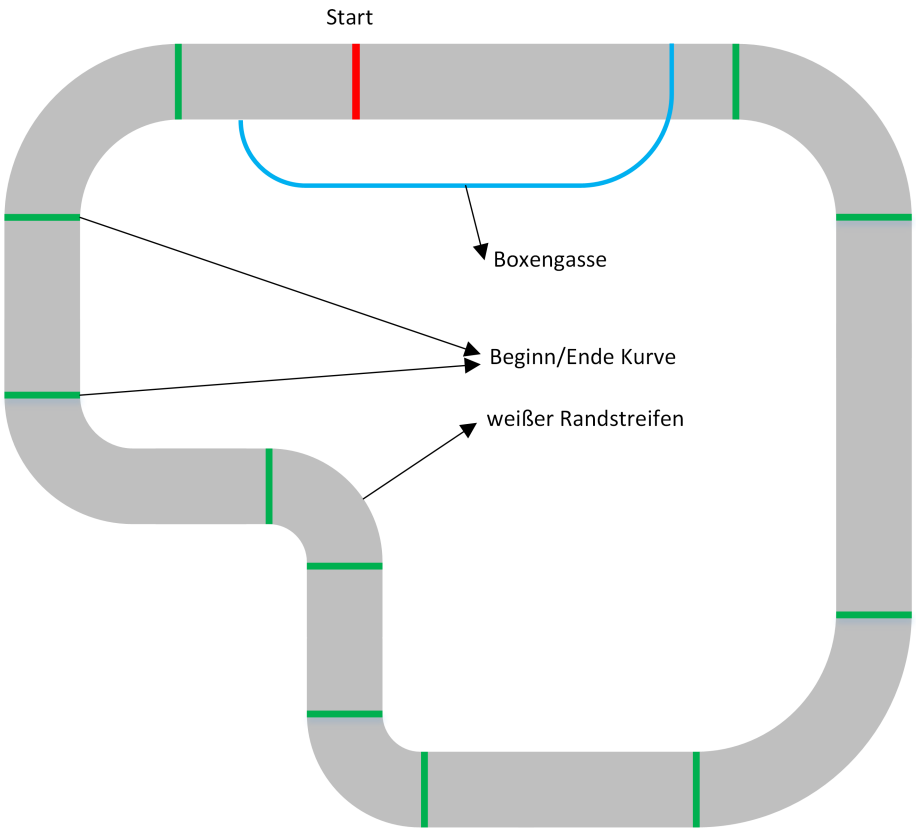

Abb. 2.2: Verbesserte Version der Rennstrecke

Eventuell kann später getestet werden, ob die Kurvenmarkierungen sogar ganz weggelassen werden können. Dies erfordert jedoch eine noch genauere, softwarebasierte Aufzeichnung der Strecke, was zum jetzigen Zeitpunkt noch nicht bewertet werden kann. Boxengasse sowie Start/Ziel werden weiterhin farbig markiert. Somit ergeben sich folgenden Linien:

Start/Ziel-Linie:	rot
Beginn/Ende-Kurve-Lini:	grün
Boxengasse:	blau
Fahrbahnmarkierung:	weiß

2.3 Fahrzeuganforderungen

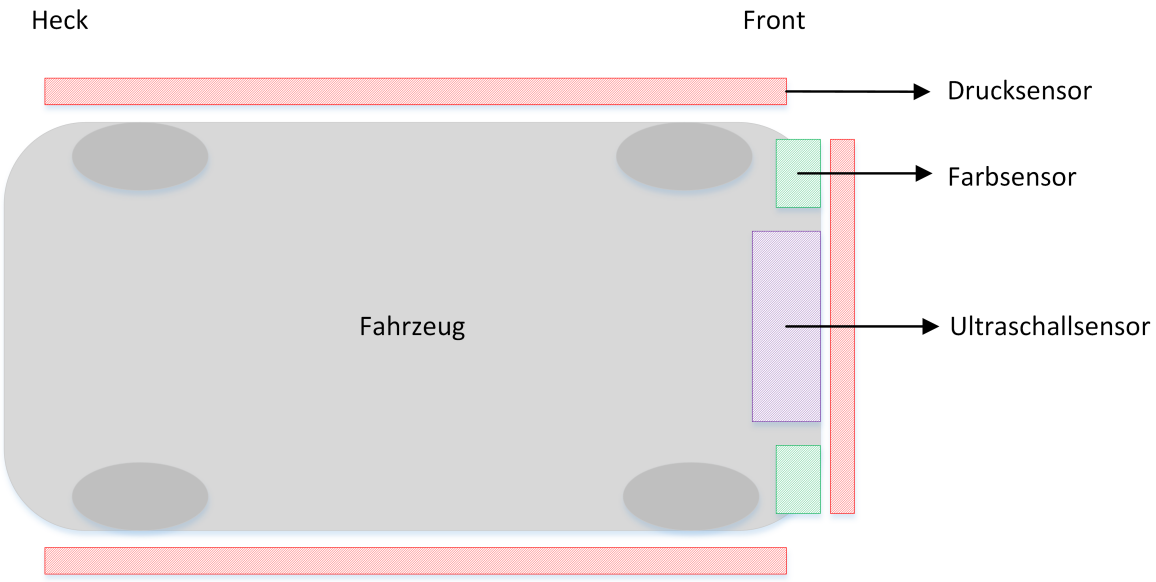

Abb. 2.3: Sensoren am Fahrzeug

Das Fahrzeug sollte an den Seiten und vorne Drucksensoren haben. Diese dienen zur Erkennung von Kollisionen und entsprechenden Reaktionen. Alternativ zu den Drucksensoren kann ein Ultraschallsensor an der Front des Fahrzeugs der Kollisionsvermeidung mit Hindernissen wie beispielsweise anderen Fahrzeugen dienen. Am Heck kann auf einen Sensor verzichtet werden, da dort im Falle einer Annäherung das auffahrende Fahrzeug die Situation erkennen würde.

2.4 Aktivitätsdiagramm

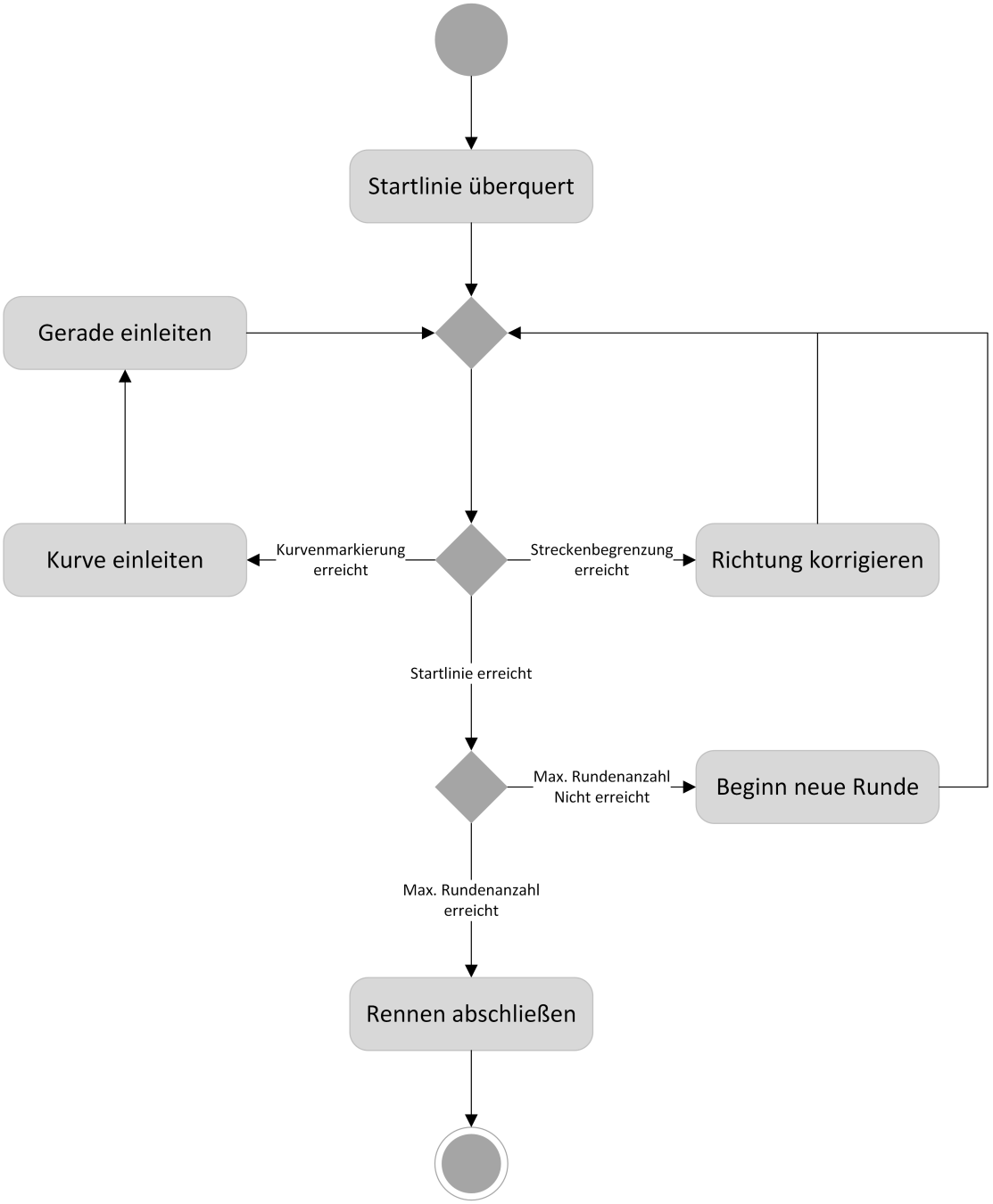

Abb. 2.4: Aktivitätsdiagramm

2.5 Software-Architektur

Die fünf High-Level-Requirements werden jeweils durch ein eigenes Package abgebildet. Der Kern des Projekts, *HLR-1*, stellt das autonome Befahren der Rennstrecke dar. Die anderen vier High-Level Requirements sind Erweiterungspackages, die je nach zeitlicher Entwicklung zusätzlich implementiert werden können.

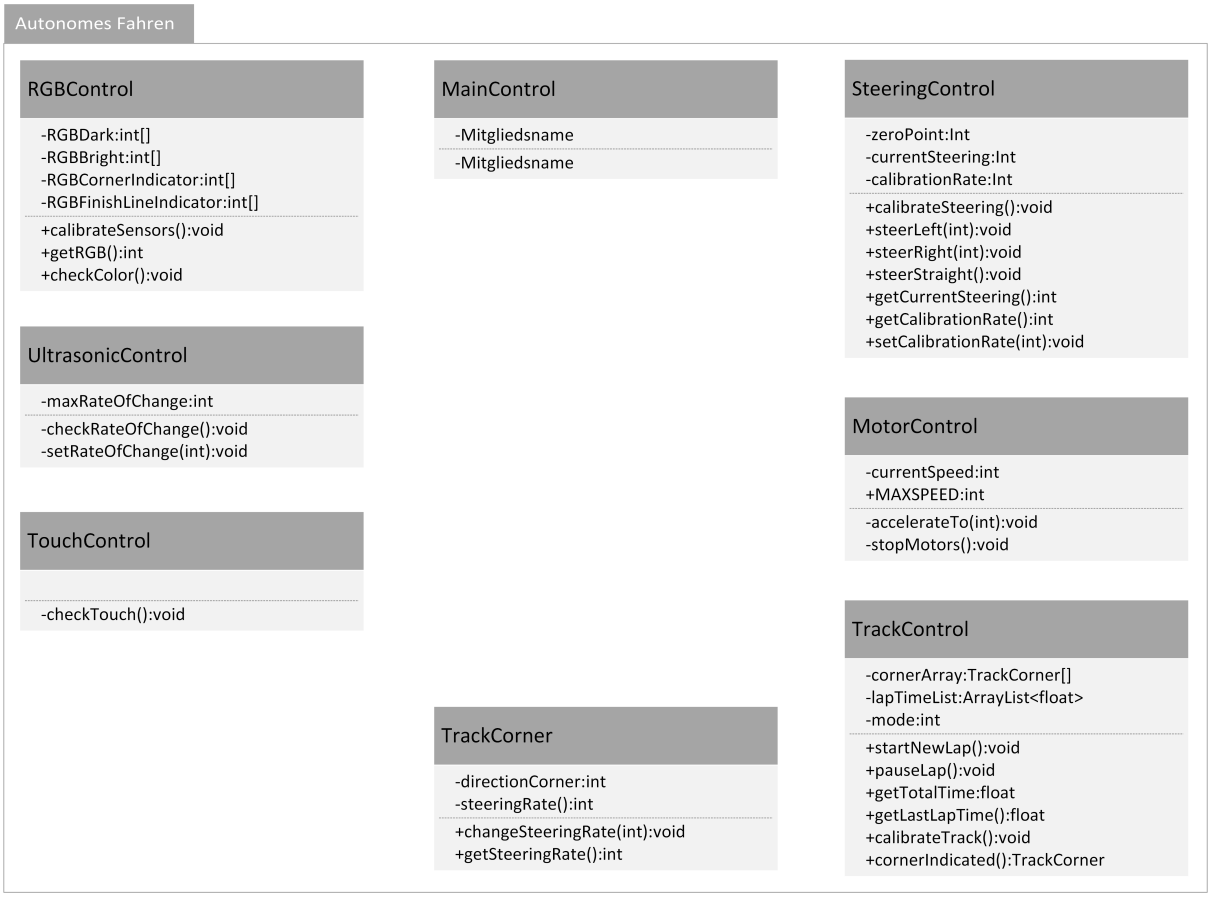

Abb. 2.5: Klassendiagramm

Das Package *Autonomes Fahren* besteht aus einer Hauptklasse *MainControl*, die Sensorik und Aktorik kombiniert und in Echtzeit das Auto steuert. Sie greift auf einige Nebenklassen zu, die zum Einen für die Steuerung der Motoren zuständig sind, und zum Anderen die Sensoren auswerten. Dadurch lässt sich der Ablauf deutlich vereinfachen und es ist möglich, einzelne Klassen auszutauschen (z.B. die Lenkung) ohne den gesamten Ablauf zu ändern. Damit besteht sogar die Möglichkeit, Autos unterschiedlicher Bauart (die den Spezifikationen entsprechen) zu steuern.

Darüber hinaus gibt es eine Klasse, die die Rennstrecke verwaltet. Sie zeichnet in der Einführungsrunde die gefahrene Strecke möglichst genau auf. Auf diese Streckenübersicht greift die *MainControl* im anschließenden Rennen zu. In Kombination mit den Sensordaten wird die Streckenübersicht auch während dem Rennen ständig verbessert um so eine möglichst optimale Linie zu fahren.

3 Erste Umsetzungsphase

3.1 Entwicklungsentscheidungen

Bei dem System handelt es sich um ein Echtzeitsystem, das jedoch keine harten Deadlines einhalten muss. Generell charakterisiert ein Echtzeitsystem das rechtzeitige Eintreffen eines Ergebnisses. Im konkreten Fall muss das Fahrzeug auf die Streckeneigenschaften reagieren und Kollisionen mit anderen Fahrzeugen vermeiden oder gewollt herbeiführen. Daher sind gewisse Zeitfenster in der Streckenkalkulation einzuhalten, sodass das Fahrzeug nicht von der Fahrbahn abweicht. Zudem sollte bei der Erkennung einer möglichen Kollision das Signal zu einem bestimmten Zeitpunkt in der Recheneinheit ankommen, sodass es dort abgefangen und darauf reagiert werden kann. Diese Bedingungen sind mit keinen harten Echtzeitbedingungen verbunden, d.h. bei Nichteinhalten wird kein großer Schaden verursacht. Diese Betrachtungen lassen auf den Einsatz einer Programmiersprache schließen, die keine harten Echtzeitbedingungen unterstützen muss. Java reicht für den Anwendungsfall völlig aus.

Als Entwicklungsumgebung kommt Eclipse Mars zum Einsatz. Dabei ist zu beachten, dass die 32bit Version von Eclipse, JDR und JDK Vorraussetzung für ein erfolgreiches Übertragen auf den NXT ist.

Für die Erstellung eine umfassende Dokumentation wird der Web-Editor GoogleDocuments genutzt, sodass das gesamte Team parallel an den Dokumenten arbeiten kann. Als Versionsverwaltungstool der Software soll GitHub zum Einsatz kommen.

3.2 Sensortests

3.2.1 Farbsensor

Als Farbsensoren werden RGB-Sensoren verwendet. Da die Farbe abhängig vom einfallenden Licht und der Entfernung des Sensors zum Untergrund ist, haben wir speziell auf den konkreten Anwendungsfall abgestimmte Sensortests vorgenommen. Vor dem Start eines Rennens soll auf Basis der gemessenen Werte eine Sensor-Kalibrierung vorgenommen werden, sodass sich die Software im Fahrzeug auf die aktuell herrschenden Lichtbedingungen einstellen kann.

Eine erste Messreihe ergab folgende Rot-, Grün- und Blauwerte für die aufgelisteten farbigen Untergründe:

Farbe des Untergrunds	**Blauwert**	**Grünwert**	**Rotwert**
rot (Trennblatt)	101	920	260
grün (Trennblatt)	126	193	111
gelb (Trennblatt)	185	278	260
blau (Trennblatt)	250	202	125
schwarz (Schaumstoffplatte)	350	460	598
weiß (Kreppband)	220	240	250

Wie man anhand der Messreihe sehen kann, können einige Werte nicht korrekt sein. Da bei erneutem Messen jedoch ähnliche Werte auftraten, haben wir uns für eine Umgehung des Problems durch eine Softwarelösung entschieden. Eine genauere Betrachtung der Werte lässt den Schluss zu, dass alle Werte zwischen 255 und 300 den Wert 255 annehmen sollten. Alle Werte, die größer als 300 sind, sollten den Wert 0 annehmen. Durch eine solche Umsetzung werden die Farben entsprechend richtig dargestellt:

Farbe des Untergrunds	**Blauwert**$_{\text{(korrigiert)}}$	**Grünwert**$_{\text{(korr.)}}$	**Rotwert**$_{\text{(korr.)}}$
rot (Trennblatt)	101	0	255
grün (Trennblatt)	126	193	111
gelb (Trennblatt)	185	255	255
blau (Trennblatt)	250	202	125
schwarz (Schaumstoffplatte)	0	0	0
weiß (Kreppband)	220	240	250

Die meisten Farben lassen sich laut Testreihe relativ gut unterscheiden. Lediglich der Unterschied zwischen gelb und weiß (in dem Fall Kreppband) ist sehr gering. Deshalb werden gelbe Linien nicht verwendet (siehe Abschnitt *Streckengestaltung*).

3.2.2 Drucksensor

Die Drucksensoren dienen der frühzeitigen Kollisionserkennung. Anhand eines Drucktests stellte sich heraus, dass der Sensor ein Hindernis erst bei vollständig eingedrücktem Sensorfeld erkennt. Leichter Druck löst keine Zustandsänderung aus. Dies könnte ein Problem bei der späteren Umsetzung darstellen, da ein spezieller Aufbau nötig wird, um eine bevorstehende Kollision zu erkennen, bevor sie tatsächlich eintritt.

3.2.3 Ultraschallsensor

Der Ultraschallsensor dient der Hinderniserkennung. Es soll die gesamte vor dem Fahrzeug liegende Fläche abgedeckt werden, sodass bei einem Rennen mit mehreren Fahrzeugen Kollisionen vermieden werden können.

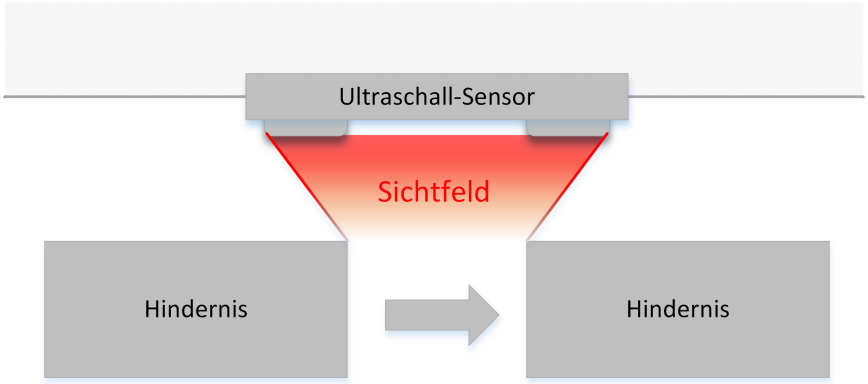

Abb. 3.1: Sichtfeld eines Ultraschallsensors

Mit Hilfe eines etwa 4 cm hohen Hindernisses wurde das Sichtfeld des Ultraschallsensors ermittelt. Dabei wurde der Sensor etwa auf der Höhe befestigt, auf der er sich auch später im Fahrzeug befinden soll. Durch langsames Vorbeischieben des Hindernisses parallel zum Sensor konnte der Bereich festgestellt werden, innerhalb dem der Sensor das Hindernis erkennt. Die vom Sensor erkannten Werte wurden auf dem Display des NXT angezeigt. Ein schlagartiger Wechsel auf eine bedeutend kleinere Zahl war ein Indiz für ein Hindernis.

Bei diesem Versuchsaufbau stellte sich heraus, dass das Sichtfeld des Ultraschall-Sensors recht eingeschränkt ist und nur einen kleinen Bereich der davorliegenden Fläche wahrgenommen wird. Ein einzelner Ultraschall-Sensor ist für die Hinderniserkennung eines Fahrzeugs somit nicht geeignet.

4 Reflexion des ersten Standes

Wir begannen mit dem Projekt im Oktober 2015. Die Entscheidung des Themas fiel recht schnell, wir waren uns einig, dass es sich als Studienarbeitsthema hervorragend eignet. Es kann beliebig erweitert werden und bietet eine Reihe an Möglichkeiten, Erlerntes zu vertiefen und Neues zu lernen. Während die Grundsteuerung der Fahrzeuge recht schnell umgesetzt und ein erster Stand bereits nach wenigen Wochen erreicht war, erfordern komplexeren Überlegungen eine genaue Planung. Daher haben wir uns einige Zeit mit einem Entwurf der Software-Architektur beschäftigt, um diese möglichst einfach erweiterbar zu gestalten. Auf Basis des aktuellen Projektplans haben wir fehlende Bauteile bestellt, die auch bereits eingetroffen sind.

Der Umstand, dass für erfolgreiches Übertragen auf den NXT spezielle Versionen von Java und Eclipse nötig sind, brachte einiges an Arbeitsaufwand mit sich. Bis die Entwicklungsumgebung bei allen Teammitgliedern wie vorgesehen funktionierte, dauerte es einige Wochen. Das schränkte den Arbeitsfortschritt zunächst erheblich ein.

Bisher haben wir uns etwa wöchentlich einmal für ein paar Stunden zusammengesetzt und an dem Projekt weitergearbeitet. Die Dokumentation erfolgt außerhalb dieser Zeiten online über ein geteiltes Dokument, das jedes Teammitglied bearbeiten kann. Das Projekt bietet die Möglichkeit, ab einem gewissen Stand einzeln an den Fahrzeugen weiterzuarbeiten. Denn Ziel sind mehrere Fahrzeuge, die gegeneinander antreten. So kann auch innerhalb des Teams ein Wettbewerb stattfinden, wer das beste Fahrzeug baut und programmiert. Generell funktioniert die Teamarbeit in unserem Projekt sehr gut, da sich jeder für das Projekt interessiert und verantwortlich fühlt.

Die genaue Verteilung der Sensoren auf dem Fahrzeug (siehe Kapitel *Fahrzeuganforderungen*) wird aktuell diskutiert. Unter anderem steht die Frage der Verwendung eines Multiplexers im Raum, um so mehr Sensoren an den NXT anschließen und in der Software verwenden zu können.

Selbständigkeitserklärung

Ich versichere hiermit, dass wir unsere Studienarbeit mit dem Thema

LEGO Racers

selbständig verfasst und keine anderen als die angegebenen
Quellen und Hilfsmittel benutzt haben.

Ort, Datum	Unterschrift Lara Kroesen
Ort, Datum	Unterschrift Jens Müller
Ort, Datum	Unterschrift Jan Herkommer