Projektseminar Echtzeitsysteme

Ausarbeitung vom Team OCP
Proseminar eingereicht von
Christian Rose, Tajas Ruschke, Andreas Schaefer, Rainer Wahler
am 7. April 2015

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachgebiet Echtzeitsysteme

Elektrotechnik und Informationstechnik (FB18)

Zweitmitglied Informatik (FB20)

Prof. Dr. rer. nat. A. Schürr
Merckstraße 25
64283 Darmstadt

www.es.tu-darmstadt.de

Gutachter: Prof. Dr. rer. nat. A. Schürr
Betreuer: M.Sc. Geza Kulcsar

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Szenarien . . . 1
1.1.1 Fahren mit seitlicher Abstandsregelung . . . 1
1.1.2 Erkennen und Fahren einer Kurve . . . 1
1.1.3 Paralleles Einparken . . . 2
1.1.4 Ferngesteuertes Fahren . . . 2
1.1.5 Befahren einer Kreuzung mit Beachtung anderer Verkehrsteilnehmer . . . 3
1.2 Eingesetzte Hardware . . . 3
1.3 Projektmanagement . . . 4
1.3.1 Definition der Verantwortlichkeiten . . . 4
1.3.2 Projektplan . . . 5
1.3.3 Zusammenarbeit und Projektverwaltung . . . 5

2 Implementierung **7**

2.1 Architektur . . . 7
2.2 Tasks . . . 8
2.2.1 WallDistanceControl . . . 8
2.2.2 CurveDetect . . . 8
2.2.3 DriveCurve . . . 8
2.2.4 ParkingSpaceDetect . . . 9
2.2.5 ParkInSlot . . . 9
2.2.6 RemoteController . . . 9
2.2.7 CrossroadDetect . . . 9
2.2.8 Car2CarCom . . . 10
2.2.9 DriveRandomDirection . . . 10
2.2.10 SwitchInput . . . 10
2.3 Modi . . . 10
2.3.1 wall-curve-Modus . . . 11
2.3.2 parking-Modus . . . 11
2.3.3 remote-controlled-Modus . . . 11
2.3.4 crossroad-Modus . . . 11
2.4 Android Applikation . . . 12
2.4.1 JoystickView . . . 12
2.4.2 Bluetooth Kommunikation . . . 12
2.4.3 `Joystick.java` . . . 12
2.4.4 `CrossActivity.java` . . . 14
2.4.5 `BtMessage.java` . . . 14

3 Fazit **15**
3.1 Ergebnisse 15
3.1.1 Fahren mit seitlicher Abstandsregelung 15
3.1.2 Erkennen und Fahren einer Kurve 15
3.1.3 Paralleles Einparken 15
3.1.4 Ferngesteuertes Fahren 16
3.1.5 Befahren einer Kreuzung mit Beachtung anderer Verkehrsteilnehmer 16
3.2 Gewonnene Erkenntnisse 16
3.3 Ausblick 17

A PID-Regelung **19**

B Aufgabenverteilung und Zeitplan **20**

1 Einleitung

In diesem Kapitel sollen kurz die zu lösenden Szenarien des Projektseminars vorgestellt werden. Zusätzlich werden Informationen zum Projektmanagement wie die Organisation im Team oder die Zuordnung der Verantwortlichkeiten geliefert. Für detailliertere Informationen zum Projektseminar oder den gestellten Aufgaben sei auf die offiziellen Dokumente des Projektseminars verwiesen.

1.1 Szenarien

Die während des Projektseminars zu bearbeitenden Aufgaben sind in einzelne Szenarien untergliedert. Jedes Szenario wird durch eine Reihe von Akzeptanzkriterien definiert, die erfüllt sein müssen um eine erfolgreiche Bearbeitung des Szenarios zu bestätigen. So ist es möglich den Fortschritt des Projekts jederzeit zu überprüfen.

1.1.1 Fahren mit seitlicher Abstandsregelung

Das Fahrzeug wird an einer Wand platziert und soll dieser automatisch folgen. Dazu muss eine seitliche Abstandsregelung implementiert werden, die sicherstellt, dass das Fahrzeug einen definierten Abstand zur Wand einhält. Generell kann dabei nicht davon ausgegangen werden, dass es sich um eine glatte Wand handelt. Beim Test im Gebäude werden Hindernisse wie z.B. Türrahmen zu Sprüngen in der Abstandsmessung führen. Die Abstandsregelung muss demnach so ausgelegt werden, dass Sprünge entsprechend ausgeregelt werden. Außerdem ist der anfängliche Abstand zur Wand nicht vorgegeben, sodass sich das Fahrzeug zu Beginn an den Soll-Abstand annähern muss.

Akzeptanzkriterien:

1. Das Fahrzeug wird parallel zu einer Wand platziert und folgt auf Befehl dieser
2. Anfänglich muss sich das Fahrzeug vom Startabstand dem Soll-Abstand annähern
3. Eine seitliche Abstandsregelung sorgt dafür, dass das Fahrzeug immer mit gleichem Abstand zur Wand fährt
4. Unebenheiten werden entsprechend ausgeregelt

1.1.2 Erkennen und Fahren einer Kurve

Hierbei handelt es sich um eine Erweiterung des Szenarios "Fahren mit seitlicher Abstandsregelung". Während der Fahrt entlang der Wand soll das Fahrzeug erkennen, ob es sich auf eine Kurve zubewegt und diese selbstständig nehmen. Eine Kurve ist hierbei ein Abknicken der Wand, die Abstandsmessung erkennt somit einen sehr großen Sprung.

Nachdem die Kurve ordnungsgemäß durchfahren wurde soll das Fahrzeug erneut durch eine seitliche Abstandsregelung der Wand folgen. Hierbei ist anzumerken, dass durch den begrenzten Lenkwinkel und dem daraus resultierenden Kurvenradius es nicht

möglich ist, am Ende der Kurve einen entsprechend nahen Abstand zur Wand sicherzustellen. Aus diesem Grund muss sich das Fahrzeug ähnlich zum Start der Geradeausfahrt dem Soll-Abstand erneut annähern.

Akzeptanzkriterien:

1. Das Fahrzeug erkennt automatisch eine Kurve und leitet diese ein
2. Am Ende der Kurve folgt das Fahrzeug erneut der Wand mit seitlicher Abstandsregelung
3. Es muss sichergestellt werden, dass sich das Fahrzeug am Ende der Kurve erneut dem Soll-Abstand annähert

1.1.3 Paralleles Einparken

Bei diesem Szenario soll das Fahrzeug selbständig eine entsprechend breite Parklücke auf der rechten Seite detektieren und mit möglichst wenig Zügen in diese Lücke manövrieren. Hierzu muss die Parklücke ausgemessen und anhand der Messung entschieden werden, ob die Lücke zum Einparken eine ausreichende Länge hat. Ist die Lücke groß genug, fährt es in die Startposition zum Einparken. Es fährt dann selbstständig in die Lücke ohne andere Autos zu rammen. Aufgrund des bereits erwähnten begrenzten Lenkwinkels ist das Einparken in drei Zügen je nach Größe der Lücke nicht möglich.

Akzeptanzkriterien:

1. Das Fahrzeug sucht automatisch eine Parklücke auf der rechten Seite und vermisst diese
2. Anhand der Länge der Parklücke wird entschieden, ob die Parklücke zum parallelen Einparken ausreicht
3. Wurde eine ausreichend große Parklücke detektiert, parkt das Fahrzeug automatisch ein wobei es möglichst parallel zur Wand zum stehen kommen sollte
4. Die Anzahl der benötigten Korrekturzüge sollte möglichst gering gehalten werden.
5. Es dürfen keine Hindernisse berührt werden

1.1.4 Ferngesteuertes Fahren

Über einen frei wählbaren Kommunikationskanal erhält der Benutzer die Möglichkeit die Bewegung des Fahrzeugs zu kontrollieren. Dabei sollen die üblichen Funktionen eines einfachen ferngesteuerten Autos gegeben sein. Die Fernsteuerungseinheit gibt den Lenkwinkel und die zu fahrende Geschwindigkeit vor und das Fahrzeug übernimmt diese ohne Kontrolle von möglichen Kollisionen oder Ähnlichem. Um das Fahrzeug

fernsteuern zu können, muss dem Benutzer eine Auswahl des entsprechenden Fahrzeugs in geeigneter Form dargestellt werden.

Akzeptanzkriterien:

1. Der Benutzer erhält die Möglichkeit ein gewünschtes Fahrzeug zur Fernsteuerung auszuwählen
2. Die Anwendung verbindet sich automatisch mit dem Fahrzeug und ermöglicht somit die Fernsteuerung
3. Durch eine geeignete Eingabemethode erhält der Anwender die Möglichkeit das Fahrzeug fernzusteuern. Er kann damit Lenkwinkel und zu fahrende Geschwindigkeit vorgeben.

1.1.5 Befahren einer Kreuzung mit Beachtung anderer Verkehrsteilnehmer

Zwei Autos sollen an eine Kreuzung heranfahren und sich an die Regel rechts vor links halten. Dabei kann jedes Fahrzeug an jeder Position stehen und in jede Richtung weiterfahren. Es muss eine Kommunikation unter den Teilnehmern stattfinden, welche die Postion und die gewünschte Fahrtrichtung überträgt. Anschließend können die Autos ihre Fahrt regelkonform fortsetzen.

Akzeptanzkriterien:

1. Die Fahrzeuge fahren an die Kreuzung heran und warten 3 Sekunden auf andere Verkehrsteilnehmer
2. Falls kein zweites Auto ankommt, wird die Kreuzung einfach überquert
3. Steht ein weiteres Fahrzeug an der Kreuzung wird Position und gewünschte Fahrtrichtung übertragen
4. Die Fahrzeuge fahren entsprechend der geltenden Verkehrsregeln weiter

1.2 Eingesetzte Hardware

Zum Einsatz kam das Student-Model-Car von Spansion (ehem. Fujitsu). Dieses wurde bereits um einige Sensoren und Kommunikationsmodule erweitert. Folgende Module stehen zur Verfügung:

- Ultraschallsensoren: Abstandsmessung vorne, links und rechts
- Liniensensoren: Erkennung der Untergrundfarbe
- Lenkservo: Lenkwinkel über API wählbar
- Streckensensor: zurückgelegter Weg über Radsensor messbar

- Motorcontroller: Fahrgeschwindigkeit vorwärts und rückwärts wählbar
- Bluetoothmodul: Kommunikation über Bluetoothstandard
- Amber-Funkmodul: Kommunikation im ISM-Band

1.3 Projektmanagement

Das Projektseminar Echtzeitsysteme ist als Gruppenarbeit konzipiert. Um erfolgreich in einer Gruppe die gestellte Aufgabe bearbeitet zu können ist es wichtig, eine gute Organisation und klare Strukturen zu haben. Dies ist für größere Projekte mit einer Vielzahl von Projektteilnehmern essentiell, aber auch für diese Projektarbeit mit nur vier Teilnehmern durchaus von Bedeutung. Aus diesem Grund soll nun ein kurzer Überblick über das Projektmanagement gegeben werden, welches mit zur erfolgreichen Bearbeitung des Projektseminars geführt hat.

1.3.1 Definition der Verantwortlichkeiten

Direkt zu Beginn des Projekts wurde den einzelnen Teilnehmern feste Verantwortlichkeiten zugeteilt, sodass wichtige Aufgaben eine klare Zuweisung erhalten können. Nachfolgend sind die festgelegten Verantwortlichkeiten mit einer kurzen Beschreibung sowie der zugeteilten Person aufgelistet.

1. Organisation
    - Kommunikation mit den Projektverantwortlichen
    - Verantwortlich für die Einhaltung von Terminen und Absprachen
    - Allgemeine Organisation des Projekts
    - Zuständig: Christian Rose
2. Test Management
    - Organisation der Systemtests
    - Überprüfung ob alle Akzeptanzkriterien eines Szenarios erfüllt sind
    - Dokumentation und Weitergabe von erkannten Fehlern
    - Zuständig: Tajas Ruschke
3. Release Management
    - Freigabe von stabilen Softwareständen
    - Organisation der zu bearbeitenden Software-Features
    - Setup und Wartung des Versionsverwaltungssystems
    - Zuständig: Rainer Wahler
4. Zeit Management
    - Planung der Arbeitsphasen

- Planung der Projektmeilensteine
- Organisation der Teamtreffen
- Zuständig: Andreas Schaefer

Es ist wichtig darauf hinzuweisen, dass die Vergabe einer Verantwortlichkeit nicht bedeutet, dass die entsprechende Person die gesamte Arbeit in diesem Bereich übernehmen muss. Vielmehr ist die Person dafür verantwortlich dafür zu sorgen, dass alle Arbeiten in diesem Bereich erfolgreich durchgeführt werden, wobei das gesamte Team an der Durchführung beteiligt ist. Daher gibt es auch keine spezielle Verantwortlichkeit "Entwicklung", da hier alle Teilnehmer beteiligt sind und eine Organisation bereits durch die anderen Verantwortlichkeiten stattfindet.

1.3.2 Projektplan

Direkt zu Beginn des Projektseminars wurde ein Projektplan erstellt, der als Grundlage für die gesamte Bearbeitungszeit diente. Hierzu wurden Meilensteine und Deadlines definiert und Bearbeitungszeiträume für die einzelnen Szenarien vorgesehen. Großzügige Pufferzeiten sollten Verzögerungen abfangen, sodass der Projektabschluss nicht gefährdet wird. Zusätzlich wurden Abwesenheiten bzw. Verhinderungen wie zum Beispiel die Vorbereitung auf Prüfungen erfasst, sodass diese direkt in den Projektablauf eingearbeitet werden konnten.

Zur Erstellung des Projektplans wurde die freie Software GanttProject [1] eingesetzt. Diese ermöglicht eine einfache und schnelle Erstellung des Projektplans, zusätzlich kann der Fortschritt des Projekts direkt in den Projektplan eingetragen werden. Der Projektplan wurde während der Projektlaufzeit stetig überprüft und bearbeitet. So konnten Abweichungen direkt erkannt werden und auf diese entsprechend reagiert werden.

Der erstellt Projektplan ist diesem Bericht im Anhang B beigefügt.

1.3.3 Zusammenarbeit und Projektverwaltung

In diesem Abschnitt werden die aus unserer Sicht wichtigsten Aspekte der Zusammenarbeit im Team sowie der Projektverwaltung aufgezeigt.

1. Regelmäßige Teamtreffen

   Gemeinsames Arbeiten ist deutlich produktiver als die separate Bearbeitung von Aufgaben. Probleme können in der Gruppe besprochen werden und Absprachen über das weitere Vorgehen werden nicht verzögert. Zusätzlich motiviert ein fester Termin jeden etwas für das Projekt zu machen.

   Als Ergänzung zu den regelmäßigen Treffen kann sich jeder Teilnehmer Zeiten einteilen, an denen er selbstständig am Projekt arbeitet.

2. Versionsverwaltungssystem

   Ein Versionsverwaltungssystem soll die Entwickler bei der Arbeit unterstützen und eine reibungslose Zusammenarbeit ermöglichen. Die Rechnerbetriebsgruppe der

[1] http://www.ganttproject.biz/

TU-Darmstadt ermöglicht jedem Student den Zugriff auf ein Projektverwaltungssystem [2]. In diesem System wurde für das Projektseminar ein Git-Repository für alle Teilnehmer eingerichtet, sodass ein gemeinsames Versionsverwaltungssystem genutzt werden konnte.

[2] https://scm.rbg.informatik.tu-darmstadt.de/

2 Implementierung

Dieses Kapitel liefert Informationen zur Software-Architektur sowie den einzelnen Tasks und Modi, die für den Abschluss der Szenarien entwickelt wurden. Hierbei wird sich vor allem auf den Verwendungszweck beziehungsweise die Aufgabe der Tasks und Modi bezogen.

2.1 Architektur

Die grundlegende Architektur des Programms basiert auf einem übergeordneten "Controller" Task, welcher zwischen den verschiedenen Modi umschaltet. Dieser "behaviour" Task ist in die vier Klassen "wall curve", "parking", "remote controlled" und "crossroad" unterteilt. Jede dieser Klassen für sich integriert die Verwaltungslogik, welche für den jeweiligen Modus benötigt wird. Die eigentliche Abarbeitung der gewünschten Aufgabe geschieht dann in verschiedenen untergeordneten Tasks, welche von der Logik der übergeordneten Klasse gestartet, beendet und ausgewertet werden. Tasks werden beim ersten Aufruf über die Methode "os_registerProcessStack(..)" gestartet und durch "os_suspendTask(..)" schlafen gelegt. Bei erneuter Aktivierung werden die Tasks mit "os_resumeTask(..)" wieder aufgeweckt.

In Abbildung 2.1 ist eine graphische Übersicht über die Hierarchie des Programmaufbaus dargestellt.

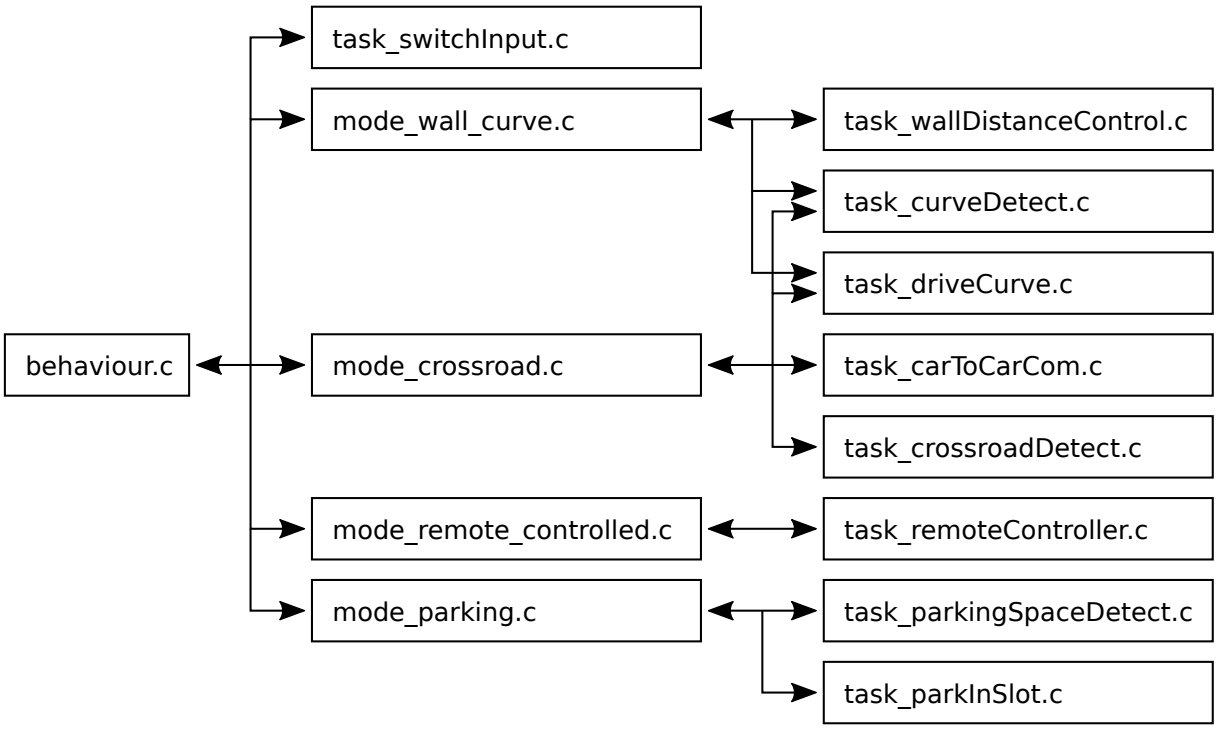


Abbildung 2.1: Hierarchie des Programms

2.2 Tasks

Tasks bilden die Grundstruktur unserer Software-Architektur und lassen sich als nebenläufige Prozesse interpretieren. Jeder Task hat dabei eine bestimmte Aufgabe und liefert Informationen mittels Statusinformationen an andere Programmteile. Teilweise werden Tasks für verschiedene Modi wiederverwendet, was diese Aufteilung hinsichtlich der Code-Wiederverwendung sinnvoll erscheinen lässt.

2.2.1 WallDistanceControl

WallDistanceControl implementiert einen PD-Regler, wobei der proportionale Anteil der gemessene Abstand zur rechten Seite der Wand ist. Für den differenziellen Anteil werden zwei Messungen im zeitlichen Abstand des Tick-Intervalls vorgenommen und die Steigung zur Wand berechnet. Beide Ergebnisse werden mit Verstärkungsfaktoren verrechnet um den neuen Lenkwinkel für das Fahrzeug zu erhalten und den vorgegebenen Abstand zur Wand halten zu können. Nach dem Aktivieren des Tasks wird der aktuelle Abstand zur Wand ermittelt und mit dem Soll-Abstand verglichen. Weichen diese voneinander ab wird die Entfernung für den PD-Regler kontinuierlich an den Soll-Wert angenähert. Dieses Verfahren ermöglicht es das Auto in einem "beliebigen" Abstand zur Wand aufzustellen ohne den PD-Regler aus dem Tritt zu bringen. Das Auto nähert sich dann sukzessive der gewünschten Entfernung zur Wand an. Die Verstärkungsfaktoren wurden experimentell bestimmt, wobei diese von einer Java Applikation über AMBER-Funk eingestellt werden können. Ein Test mit einem zusätzlichen integralen Anteil, also einem PID-Regler, hat Probleme bereitet und führte zu Instabilitäten. WallDistanceControl wurde als Task in unserer Programmstruktur implementiert und kann vom Controller "behaviour" gestartet und beendet werden.

2.2.2 CurveDetect

Im "wall-curve-Modus", während der Task WallDistanceControl aktiv ist, läuft nebenbei auch der Task CurveDetect. Dieser überwacht kontinuierlich den Abstand zur Wand und versucht zu erkennen, ob ein Quergang aufgetaucht ist, wofür ein Grenzwert hinterlegt wurde. Überschreitet die Entfernung zur Wand diesen Grenzwert wird angenommen, dass sich unmittelbar neben dem Auto keine Wand mehr befindet. Zur Sicherheit gegen eine fehlerhafte Messung wird nach 50ms erneut geprüft, ob der Grenzwert immer noch überschritten ist. Wird dieser Zustand erkannt teilt der Task dem "behaviour" Controller dies mit.

2.2.3 DriveCurve

DriveCurve ist dafür zuständig an einer Kreuzung in eine beliebige Richtung abzubiegen oder über diese drüber zu fahren. Der Task wird unter anderem nach dem Erkennen einer Kurve im "wall-curve-Modus" aktiviert um den Abbiegevorgang erfolgreich durchzuführen. Ebenfalls Verwendung findet der Task im "crossroad-Modus", bei welchem nach definierten Verkehrsregeln über die Kreuzung gefahren werden soll. Intern wird mit Hilfe der zurückgelegten Entfernung, gemessen über den Radencoder, gearbeitet.

Dafür wurde experimentell ermittelt, nach welcher Entfernung bei maximalem Lenkeinschlag eine 90° Kurve gefahren wurde. Sobald die Kurve durchfahren wurde wird dem "behaviour" Controller dies mitgeteilt und wieder der PD-Regler aktiviert um der Wand zu folgen. Bedingt durch den geringen Lenkwinkel befindet sich das Auto nach der Kurvenfahrt weiter weg von der Wand als die gewünschte Soll-Entfernung für den PD-Regler. Dieses wird dann durch den Regler automatisch wieder ausgeglichen durch eine schrittweise Annäherung (siehe 2.2.1).

2.2.4 ParkingSpaceDetect

Wenn der Task ParkingSpaceDetect aktiv ist fährt das Auto geradeaus bis der Abstandssensor eine beginnende Parklücke detektiert. Das Auto fängt an die zurückgelegte Strecke zu messen bis die Breite der Parklücke ausreichend ist oder ein zweites Auto detektiert wird. Der Task meldet dann dem Controller "behaviour" die Werte für die ausgemessene Parklücke in Breite und Länge zurück falls diese groß genug ist und andernfalls wird weiter nach einer Parklücke gesucht.

2.2.5 ParkInSlot

Der Task ParkInSlot erhält vom Controller "behaviour" die Daten für eine bereits ausgemessene Parklücke und beginnt dann mit dem Einparkvorgang. Der Vorgang ist als Automat implementiert, wobei mehrere Zustände durchlaufen werden. Zunächst fährt das Auto ein Stück nach vorne um besser in die Parklücke zu kommen. Jetzt fährt das Auto mit vollem Einschlag nach Rechts bis zur Hälfte der Parklückenlänge unter Berücksichtigung der Autoabmessungen. Danach wird das Auto mit vollem Einschlag nach Links wieder die gleiche Fahrstrecke zurücklegen. Jetzt fährt das Auto wieder die halbe Parklücke nach vorne mit dem maximalen Lenkwinkel nach rechts und die andere Hälfte nach vorne mit dem maximalen Lenkwinkel nach links. Dieser Vorgang wird wiederholt bis der Abstand zwischen Auto und Wand klein genug ist. Danach wird der Einparkvorgang beendet und dem Controller "behaviour" gemeldet.

2.2.6 RemoteController

Im Task RemoteController wird lediglich die Bluetoothnachricht des richtigen Typs angemeldet und die Motoren sicherheitshalber gestoppt. Wenn dann eine Bluetoothnachricht mit neuen Werten für Geschwindigkeit und Richtung eintrifft, werden diese jeweils im Auto gesetzt. Da der Joystick beim Loslassen die Zwischenwerte bis einschließlich 0 ebenfalls an die onMoved() Methode weitergibt, wird das Fahrzeug so auch nach dem Loslassen gestoppt.

2.2.7 CrossroadDetect

CrossroadDetect ist ein sehr einfach gehaltener Task, welcher im "crossroad-Modus" parallel zum WallDistanceControl und Car2CarCom Task läuft. Es wird dabei kontinuierlich der Liniensensor, welcher sich am vorderen Ende des Autos befindet, abgefragt und ausgewertet. Sobald dieser eine Linie am Boden, welche zuvor mit Hilfe von Klebeband an

der Kreuzung erstellt wurde, erkennt, wird ein Signal an den "behaviour" Controller gemeldet.

2.2.8 Car2CarCom

Der Task Car2CarCom erledigt die Hauptarbeit bei der Entscheidung, welches Fahrzeug wann und in welche Richtung fahren darf. Initial wird über eine Android App per Bluetooth den Fahrzeugen, welche an dem Kreuzungs-Szenario teilnehmen, ihre Fahrtrichtung und Position an der Kreuzung mitgeteilt. Sobald alle Fahrzeuge ihre Konfiguration erhalten haben fahren diese bis zur Kreuzung und warten dort kurz. Die Fahrzeuge teilen dann mit Hilfe des AMBER-Funkmoduls ihre Anwesenheit an der Kreuzung und deren Fahrtrichtung mit. Nun ermitteln alle Fahrzeuge, anhand der festgelegten Vorfahrtsregeln, welches als erstes fahren darf. Ist dieses Fahrzeug über die Kreuzung gefahren sendet es per Funk ein Signal das die Kreuzung wieder frei ist. Nun startet das zweite Fahrzeug, basierend auf den gültigen Vorfahrtsregeln.

2.2.9 DriveRandomDirection

Der Task DriveRandomDirection wurde ursprünglich dafür entwickelt, um eine zufallsbasierte Entscheidung zu treffen, in welche Richtung eine Kreuzung durchfahren werden sollte. Um eine einfache Quelle für die benötigten Zufallszahlen zu bekommen, welche eine halbwegs zufällige Entscheidung ermöglicht, wurde der Radencoder verwendet. Nachdem das Auto üblicherweise schon eine unbekannte Strecke vor der Kreuzung gefahren ist kann diese Strecke als Basis des Zufallsgenerators dienen. Da nur drei mögliche Fahrtrichtungen zu unterscheiden sind wurde die Zufallszahl mit Hilfe des Modulo Operators ermittelt (x = strecke % 3).

2.2.10 SwitchInput

Der Task SwitchInput überwacht die zwei Taster auf dem Auto und ermöglicht das Umschalten in verschiedene Verhaltensweisen im Controller "behaviour". Im Startzustand wird kein Task ausgeführt und das Auto befindet sich im Ruhezustand. Wird das Auto in den nächsten Modus geschaltet wird die Zahl auf dem Hex Display um eins inkrementiert und der entsprechende Modus aktiviert. In Zustand 1 wird der wall-curve-, in Zustand 2 parking-, in Zustand 3 remote-controlled- und in Zustand 4 der crossroad-Modus im Controller aktiviert. Im Folgenden werden die Modi genauer beschrieben.

2.3 Modi

Ein Modus aggregiert mehrere Tasks zu einem komplexen Programmablauf, um so die Akzeptanzkriterien eines Szenarios zu erfüllen. Hierbei werden die Steuerung des Programmflusses vorgenommen, konkreter es werden Tasks in Abhängigkeit von Statusinformationen gestartet oder gestoppt. Die Abarbeitung der entsprechenden Aufgabe wird dann von einem Task übernommen. Implementiert wurden die verschiedenen Modi mittels einer State-Machine, wobei die Transitionen durch Informationen von laufenden Tasks gesteuert werden.

2.3.1 wall-curve-Modus

In diesem Modus wird WallDistanceControl ausgeführt, also der Abstand zur Wand beim Geradeausfahren gehalten. Falls eine Kurve von dem Task CurveDetect detektiert wird, wird eine Rechtskurve gefahren, indem der Task DriveCurve mit dem Parameter "right" aufgerufen wird.

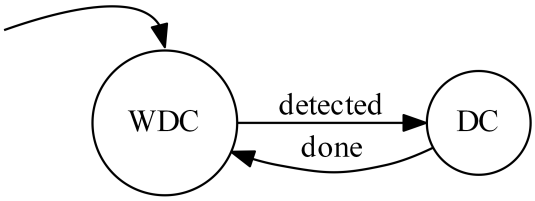


Abbildung 2.2: Zustandsdiagramm des Modus "Wall Curve".

2.3.2 parking-Modus

Im parking-Modus wird der Task ParkingSpaceDetect ausgeführt und nachdem dieser erfolgreich beendet wurde wird der Task ParkInSlot aufgerufen. Ist dieser Task beendet geht das Auto in den Ruhezustand über.

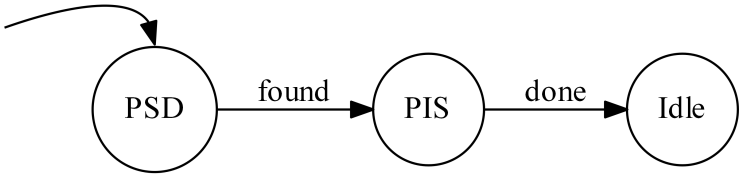


Abbildung 2.3: Zustandsdiagramm des parking-Modus.

2.3.3 remote-controlled-Modus

Im remote-controlled-Modus wird der Task RemoteController ausgeführt, welcher eine Callback-Funktion für die Bluetooth Schnittstelle registriert und auf eine Verbindung durch ein Telefon mit unserer Android Applikation wartet. Nachdem eine Verbindung aufgebaut wurde sorgt es dafür, dass empfangene Nachrichten auf die Autosteuerung umgesetzt werden. Dies umfasst die Geschwindigkeit und den Lenkwinkel des Autos.

2.3.4 crossroad-Modus

Im crossroad-Modus navigiert das Auto mittels Car-to-Car Kommunikation über eine Kreuzung. Dazu wird zunächst wieder eine Callback-Funktion auf der Bluetooth Schnittstelle registriert und das Auto wartet zunächst auf Instruktionen über unsere Android Applikation. Nachdem alle Autos die Fahrtrichtung und Position einprogrammiert bekommen haben kann ein Startsignal über die Android Applikation gesendet werden. Nun fahren alle Autos im wall-curve-Modus los bis eine Kreuzung detektiert wird. Dies

wird über den Liniensensor und einer Linie an jeder Straße vor der Kreuzung realisiert. Nun sendet jedes Auto nach dem Detektieren eine Broadcast Nachricht um seine Präsenz anzuzeigen und gibt seine Fahrtrichtung und Position an der Kreuzung bekannt. Wird keine Nachricht von einem anderen Auto empfangen fährt das Auto nach 3 Sekunden Wartezeit los. Wird jedoch eine Nachricht empfangen müssen die Autos nach den Vorfahrtsregeln die Kreuzung überqueren. Das Auto mit Vorfahrt fährt also über die Kreuzung und signalisiert den anderen Autos, dass es die Kreuzung gerade blockiert. Nach dem Überqueren der Kreuzung wird die Kreuzung freigegeben und das andere Auto kann losfahren.

2.4 Android Applikation

Für die Android-App wurde das Android-SDK verwendet. Die App besteht aus mehreren Klassen, die alle eine entsprechende Teilfunktionalität bereitstellen. Die Bluetooth Kommunikation wurde in weiten Teilen aus dem Bluetooth Chat Beispiel[Dev15] übernommen. Für die Steuerung wurde ein JoystickView Widget von mobile-anarchy-widgets[wid15] verwendet.

2.4.1 JoystickView

Das JoystickView Widget stellt bereits die benötigte Funktionalität zum Steuern des Autos bereit. Um auf die Bewegung des Sticks zu reagieren wird ein `JoystickMovedListener` implementiert. Dort wird lediglich das `onMoved`-Interface implementiert. Eine Richtungs- oder Geschwindigkeitsänderung muss nur übertragen werden, wenn der Joystick bewegt wird. Lässt man ihn los, bewegt sich der Joystick automatisch in die Mitte zurück und das `onMoved`-Ereignis wird zuletzt mit dem Wertepaar (0,0) aufgerufen. Damit bleibt das Auto nach dem Loslassen des Sticks auf jeden Fall stehen.

Nach dem Hinzufügen des Quelltextes kann der Joystick einfach dem Layout hinzugefügt und beliebig positioniert werden. In Abbildung 2.4 wird die Ansicht der App nach ihrem Aufruf gezeigt.

2.4.2 Bluetooth Kommunikation

Aus dem BluetoothChat-Beispiel lassen sich die für den Aufbau und die Verwendung einer Bluetoothverbindung nötigen Routinen ableiten. Es wird die Funktionalität zum Verbinden und Pairen neuer Geräte zur Verfügung gestellt und eine RFCOMM-Verbindung aufgebaut. Diese emuliert eine einfache serielle Schnittstelle zwischen Empfänger und Sender.

Die Auswahl des Bluetoothgerätes erfolgt über die DeviceList-Activity. Dies ist in Abb. 2.5 gezeigt.

2.4.3 `Joystick.java`

Diese Klasse stellt die Funktionalität des Hauptbildschirms bereit. Dort kann der Joystick bedient und eine Verbindung mit dem Auto hergestellt werden. Außerdem lässt sich der Kreuzungsmodus starten.

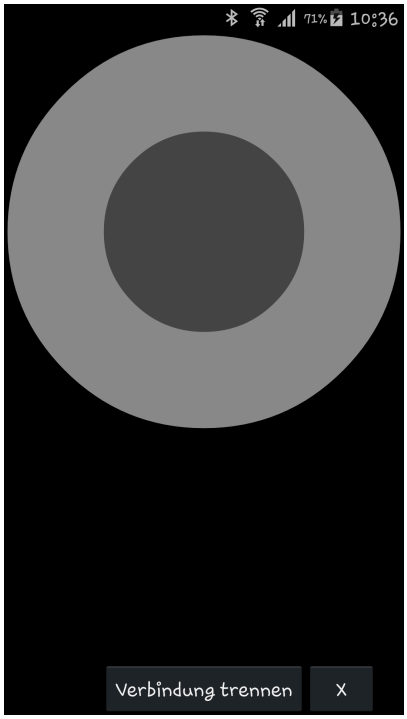


Abbildung 2.4: Joystick-Activity der AndroidApp

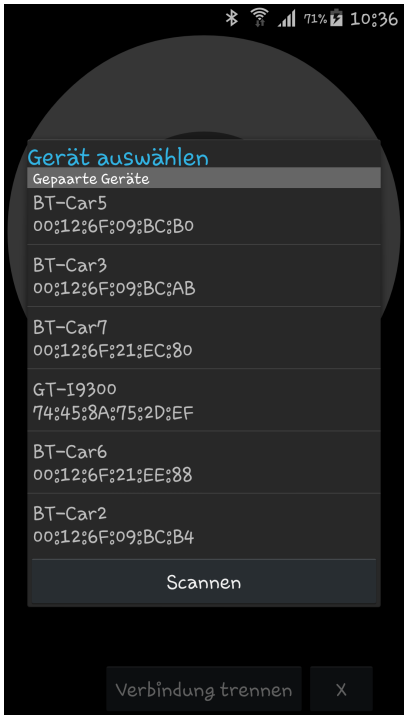


Abbildung 2.5: DeviceList mit derzeit gepaarten Geräten

2.4.4 `CrossActivity.java`

Hier wird die Möglichkeit zur Übertragung der Positionen und Richtungen für das Verhalten an der Kreuzung gegeben. Dabei besteht die Verbindung nur zu einem Fahrzeug, das die Informationen dann an die anderen beteiligten Autos weitergibt. Die Positionen an der Kreuzung sind dabei von oben gesehen im Uhrzeigersinn durchnummeriert. Die App bietet die Möglichkeit für jede Position eine CarID sowie die gewünschte Fahrtrichtung auszuwählen. Darüberhinaus gibt es einen Senden-Knopf, der dafür sorgt, dass die Informationen an die Autos weitergegeben werden. Der Starten-Knopf lässt die Autos anschließend losfahren. Dies ist in Abb 2.6 zu sehen.

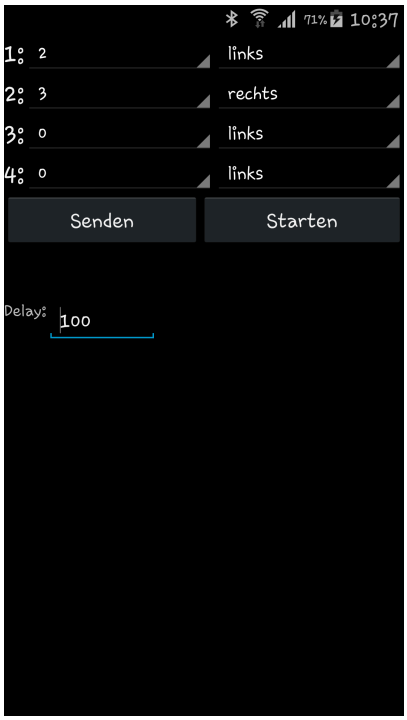


Abbildung 2.6: CrossRoad-Activity

2.4.5 `BtMessage.java`

Da in der API ein Protokoll auf dem reinen Bytestream aufgesetzt ist, welches das Verwalten der Nachrichten übernimmt, ist dieses Nachrichtenschema auch in der Android-App implementiert. Eine Bluetoothnachricht kann also über den `BtMessage`-Konstruktor angelegt und mit Informationen gefüllt werden.

3 Fazit

In diesem Kapitel sollen die Ergebnisse kurz dargestellt und ein Fazit zur Projektarbeit gezogen werden. Unter anderem werden gewonnene Erkenntnisse angeführt und mögliche Verbesserungen in einem Ausblick aufgezeigt.

3.1 Ergebnisse

Um die Ergebnisse präsentieren zu können, werden diese nach den einzelnen Szenarien sortiert aufgeführt.

3.1.1 Fahren mit seitlicher Abstandsregelung

Für das Szenario "Fahren mit seitlicher Abstandsregelung" wurde eine PD-Regelung auf Basis des seitlichen Ultraschallsensors entwickelt. Die Reglerparameter wurden experimentell bestimmt, ohne das ein besonderer Wert auf Reglerentwurf oder Parameterwahl gelegt wurde. Dies war für diese Aufgabe auch nicht nötig. Die Regelung ist ausreichend gut gedämpft, sodass das System bei Störungen nicht aufschwingt und trotzdem Änderungen ausreichend schnell nachregelt. Bei den Tests im Flur des Universitätsgebäudes funktionierte die Regelung gut. Bei größeren Lücken an der Wand wie zum Beispiel im Bereich des Fahrstuhls gab es die Problematik, dass das Fahrzeug zu weit in die Lücke herein regelte, sodass es am Ende die Lücke nicht mehr verlassen konnte. Für das Projektseminar wird dies jedoch außer acht gelassen.

3.1.2 Erkennen und Fahren einer Kurve

Für das Szenario "Erkennen und Fahren einer Kurve" wurde die PD-Regelung um eine Funktion zur Erkennung einer Kurve erweitert. Die Detektion erfolgt durch die Erkennung eines relativ großen Sprungs im seitlichen Abstand. Das Fahren der Kurve wird über einen konstanten Lenkwinkel realisiert. Das Ende der Kurve wird durch den Radsensor bestimmt und danach die PD-Regelung wieder aktiviert. Durch den geringen Lenkwinkel des Fahrzeugs ist nur ein beschränkter Kurvenradius befahrbar. Dennoch funktioniert das Kurven-Fahren gut genug, um das Fahrzeug auf einem Rundkurs im Gebäude fahren zu lassen.

3.1.3 Paralleles Einparken

Für das Szenario "Paralleles Einparken" wurde ein Verfahren entwickelt, um automatisch eine Parklücke auszumessen und in diese zu fahren. Das Ausmessen der Parklücke erfolgt durch die Erkennung eines Sprungs im seitlichen Abstand und die Messung durch den Radsensor. Das Einparken erfolgt dann mit einem selbst entwickelten Einparkverhalten.

Wurde die Parklücke groß genug gewählt, so ist ein Einparken innerhalb eines Zuges möglich. Bei kleineren Parklücken müssen teilweise mehrere Korrekturzüge durchgeführt werden. Trotzdem ist das Verfahren in sofern gut geeignet, da es nicht auf einzelne Lücken trainiert wurde.

3.1.4 Ferngesteuertes Fahren

Für das Szenario "Ferngesteuertes Fahren" wurde eine Android App entwickelt, die als Steuereinheit für das Fahrzeug fungiert. Der Benutzer kann direkt den Lenkwinkel und die Geschwindigkeit des Fahrzeugs vorgeben. Der auf dem Fahrzeug dafür integrierte Task reagiert zeitnah auf die Befehle, sodass die Steuerung des Fahrzeugs einfach und intuitiv erfolgt.

3.1.5 Befahren einer Kreuzung mit Beachtung anderer Verkehrsteilnehmer

Für das Szenario "Befahren einer Kreuzung mit Beachtung anderer Verkehrsteilnehmer" wurden verschiedene Teilaufgaben gelöst. Die Erkennung der Kreuzung wurde mittels des Liniensensors realisiert, die Kommunikation zwischen den Fahrzeugen erfolgt mittels eines selbst entwickelten Protokolls über die AMBA-Schnittstelle. Eine Kontrolllogik entscheidet über die Vorfahrtsregeln und koordiniert die einzelnen Fahrzeuge. Die Logik ist dabei dezentral gehalten, es gibt also keine zentrale Einheit, sondern jedes Fahrzeug führt selbst die erforderlichen Berechnungen aus.

3.2 Gewonnene Erkenntnisse

Das gesamte Projekt lies sich insgesamt nicht ganz so umsetzen, wie wir das ursprünglich erwartet hatten. Es war klar, dass wir uns in die Syntax von FreeRTOS erst einarbeiten mussten und auch die Hardware musste erst kennen gelernt werden. Aber wir rechneten nicht mit Problemen in der bestehenden Hardware und Software. Glücklicherweise konnten wir zu allen diesen "Herausforderungen" eine Lösung finden um das Projekt erfolgreich abzuschließen.

Nachfolgend ist eine unvollständige Liste der zusätzlichen "Herausforderungen", welche wir als Team zu meistern hatten:

- Compiler Fehler bei Vergleichen zwischen vorzeichenbehafteten 32Bit Variablen
- Radsensor lieferte, je nach Auto, unterschiedlich zuverlässige Werte
- Ein Auto besaß keinen Liniensensor
- Manche Autos kommunizierten nicht zuverlässig über AMBER
- Manche Autos sind nur sehr langsam gefahren im Vergleich zu anderen
- Teilweise nicht funktionierende AMBER-USB-Sticks

Die meisten Hardware-Probleme konnten wir nur durch tauschen der Autos oder USB-Sticks umgehen. Den Compiler-Fehler haben wir umgangen indem wir den Vergleich mit 16Bit Variablen durchführten. Dafür mussten wir den Code etwas anpassen um die nötigen 32Bit Berechnungen weiterhin durchführen zu können.

Alles in allem kann man sagen, dass das Projekt vermutlich ähnliche Hürden und Herausforderungen gestellt hat, wie es später in der Industrie wohl auch sein wird. Auch dort werden vermutlich immer wieder unvorhergesehene Dinge auftreten, mit welchen

man sich auseinandersetzen muss. Unter diesem Blickwinkel kann man abschließend ruhigen Gewissens sagen, dass das Projekt eine lehrreiche Erfahrung für das spätere berufliche Umfeld war.

3.3 Ausblick

Während des Projektseminars wurde nach Verbesserungsmöglichkeiten gefragt. In diesem Zusammenhang wurde ein Konzept erstellt, wie die verwendete Hardware mit Standard Software und Hardware erweitert werden kann, um komplexere Aufgaben zu bewältigen und besser handhabbar zu sein. Ein wichtiger Teil hierzu ist die Verwendung von ROS[3] als Software-Framework, sowie die Verwendung eines PCs mit ausreichender CPU für aufwendigere Rechenoperationen. Somit wird es auch möglich Aufgaben wie Bildverarbeitung auf den Fahrzeugen zu integrieren.

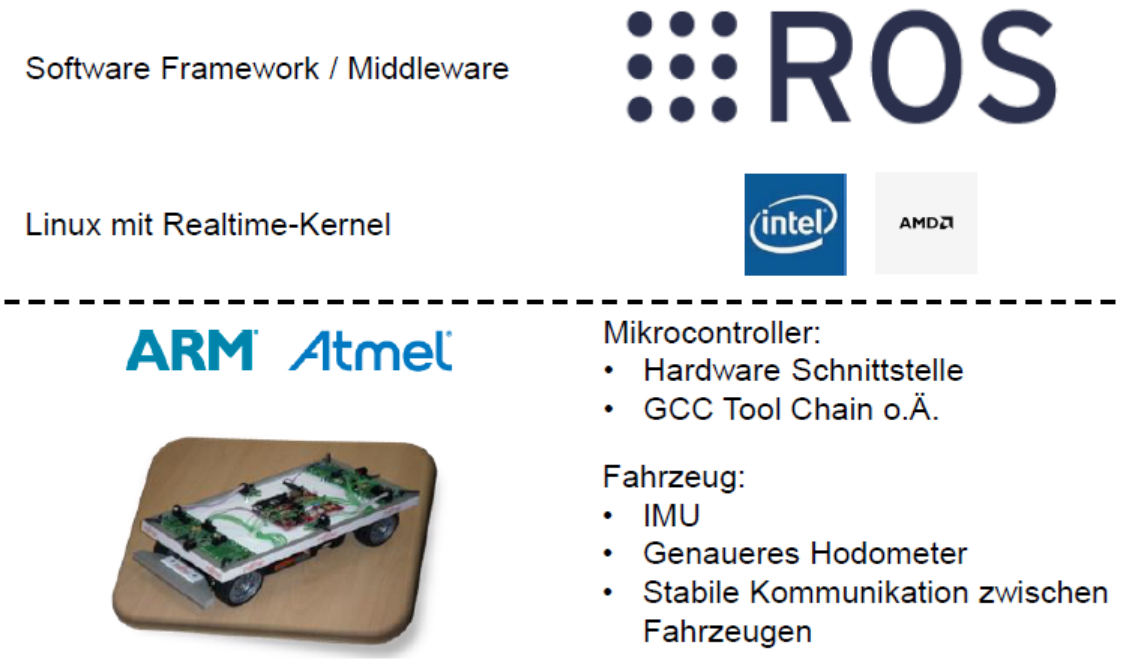


Abbildung 3.1: Konzept zur Erweiterung der Fahrzeuge aus dem Abschlussbericht

[3] http://www.ros.org

Literatur

[Dev15] DEVELOPERS, Android: *BluetoothChat Sample*. http://developer.android.com/samples/BluetoothChat/index.html, 2015. – [Online; accessed 22-März-2015]

[wid15] WIDGETS mobile-anarchy: *JoystickView Widget*. https://code.google.com/p/mobile-anarchy-widgets/wiki/JoystickView, 2015. – [Online; accessed 22-März-2015]

A PID-Regelung

Die Aufgabe eines Reglers ist die selbstständige Beeinflussung einer Regelgröße $y(t)$ mittels einer Stellgröße $u(t)$, sodass die Regelgröße einer vorgegebenen Führungsgröße $w(t)$ entspricht. In diesem Fall muss der Lenkeinschlag des Fahrzeugs so geregelt werden, dass der Abstand des Fahrzeugs zur Wand einem bestimmten Wert entspricht. Durch Störungen von außen kommt es zu einer Abweichung von der Führungsgröße, die der Regler möglichst schnell und stabil ausgleicht. In der Praxis wird dafür häufig ein PID-Regler verwendet, der seinen Namen den enthaltenen proportionalen, integralen und differenzialen Anteilen verdankt. Ebenso sind andere Regler wie ein reiner P-Regler denkbar, jedoch wird in diesem Rahmen ein Fokus auf die praktische Implementierung und nicht auf einen möglichst umfassenden Reglerentwurf gelegt. Dafür sei auf einschlägige Fachliteratur aus dem Bereich Regelungstechnik verwiesen.

Wie bereits erwähnt besteht ein PID-Regler aus drei verschiedenen Anteilen. Diese Anteile entscheiden über das spätere Regelverhalten und können jeweils über einen eigenen Parameter angepasst werden. Der proportionale Anteile erzeugt einen zur Regelabweichung linearen Anteil. Der integrale Anteil ist abhängig vom Fehler, der bei der Regelung erzeugt wird. Dazu wird die Regelabweichung aufsummiert und der Anteil steigt, je länger eine Abweichung zur Führungsgröße vorliegt. Der differenzielle Anteil wiederum ist abhängig von der zeitlichen Änderung der Regelabweichung. Hierbei führen große Sprünge in der Reglerabweichung zu einem größeren Anteil als kleine Sprünge.

Mit diesen Grundlagen könnte ein PID-Regler in Software erzeugt werden, der für die Regelung des Wandabstands verwendet werden kann.

B Aufgabenverteilung und Zeitplan

Projektseminar Echtzeitsysteme - Team OCP

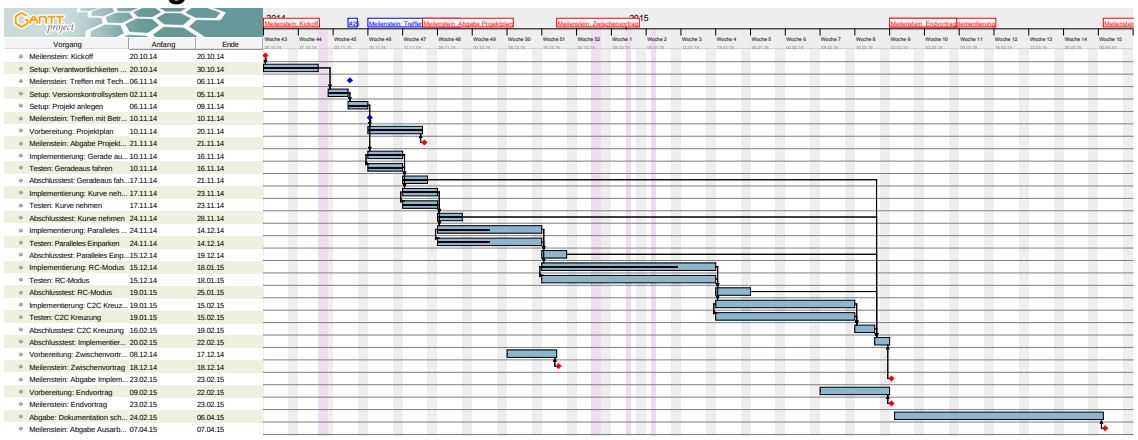
07.04.2015

2

Vorgänge

Vorgang	Anfang	Ende
Meilenstein: Kickoff	20.10.14	20.10.14
Setup: Verantwortlichkeiten klären	20.10.14	30.10.14
Meilenstein: Treffen mit Technischem Betreuer	06.11.14	06.11.14
Setup: Versionskontrollsystem	02.11.14	05.11.14
Setup: Projekt anlegen	06.11.14	09.11.14
Meilenstein: Treffen mit Betreuer	10.11.14	10.11.14
Vorbereitung: Projektplan	10.11.14	20.11.14
Meilenstein: Abgabe Projektplan	21.11.14	21.11.14
Implementierung: Gerade ausfahren	10.11.14	16.11.14
Testen: Geradeaus fahren	10.11.14	16.11.14
Abschlusstest: Geradeaus fahren	17.11.14	21.11.14
Implementierung: Kurve nehmen	17.11.14	23.11.14
Testen: Kurve nehmen	17.11.14	23.11.14
Abschlusstest: Kurve nehmen	24.11.14	28.11.14
Implementierung: Paralleles Einparken	24.11.14	14.12.14
Testen: Paralleles Einparken	24.11.14	14.12.14
Abschlusstest: Paralleles Einparken	15.12.14	19.12.14
Implementierung: RC-Modus	15.12.14	18.01.15
Testen: RC-Modus	15.12.14	18.01.15
Abschlusstest: RC-Modus	19.01.15	25.01.15
Implementierung: C2C Kreuzung	19.01.15	15.02.15
Testen: C2C Kreuzung	19.01.15	15.02.15
Abschlusstest: C2C Kreuzung	16.02.15	19.02.15
Abschlusstest: Implementierung	20.02.15	22.02.15
Vorbereitung: Zwischenvortrag	08.12.14	17.12.14
Meilenstein: Zwischenvortrag	18.12.14	18.12.14
Meilenstein: Abgabe Implementierung	23.02.15	23.02.15
Vorbereitung: Endvortrag	09.02.15	22.02.15
Meilenstein: Endvortrag	23.02.15	23.02.15
Abgabe: Dokumentation schreiben	24.02.15	06.04.15
Meilenstein: Abgabe Ausarbeitung	07.04.15	07.04.15

Gantt-Diagramm



Vorgang	Anfang	Ende
Meilenstein: Kickoff	20.10.14	20.10.14
Setup: Verantwortlichkeiten ...	20.10.14	30.10.14
Meilenstein: Treffen mit Tech...	06.11.14	06.11.14
Setup: Versionskontrollsystem	02.11.14	05.11.14
Setup: Projekt anlegen	06.11.14	09.11.14
Meilenstein: Treffen mit Betr...	10.11.14	10.11.14
Vorbereitung: Projektplan	10.11.14	20.11.14
Meilenstein: Abgabe Projekt...	21.11.14	21.11.14
Implementierung: Gerade au...	10.11.14	16.11.14
Testen: Geradeaus fahren	10.11.14	16.11.14
Abschlusstest: Geradeaus fah...	17.11.14	21.11.14
Implementierung: Kurve neh...	17.11.14	23.11.14
Testen: Kurve nehmen	17.11.14	23.11.14
Abschlusstest: Kurve nehmen	24.11.14	28.11.14
Implementierung: Paralleles ...	24.11.14	14.12.14
Testen: Paralleles Einparken	24.11.14	14.12.14
Abschlusstest: Paralleles Einp...	15.12.14	19.12.14
Implementierung: RC-Modus	15.12.14	18.01.15
Testen: RC-Modus	15.12.14	18.01.15
Abschlusstest: RC-Modus	19.01.15	25.01.15
Implementierung: C2C Kreuz...	19.01.15	15.02.15
Testen: C2C Kreuzung	19.01.15	15.02.15
Abschlusstest: C2C Kreuzung	16.02.15	19.02.15
Abschlusstest: Implementier...	20.02.15	22.02.15
Vorbereitung: Zwischenvortr...	08.12.14	17.12.14
Meilenstein: Zwischenvortrag	18.12.14	18.12.14
Meilenstein: Abgabe Implem...	23.02.15	23.02.15
Vorbereitung: Endvortrag	09.02.15	22.02.15
Meilenstein: Endvortrag	23.02.15	23.02.15
Abgabe: Dokumentation sch...	24.02.15	06.04.15
Meilenstein: Abgabe Ausarb...	07.04.15	07.04.15