Projektseminar Echtzeitsysteme

Projektseminar Echtzeitsysteme
Proseminar eingereicht von
Luka Sabljic, Daniel Alte, Julian Nix, Felix Euteneuer
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: Géza Kulcsár, MSc

ES-B-0060

Erklärung zum Proseminar

Hiermit versichere ich, das vorliegende Proseminar selbstständig und ohne Hilfe Dritter angefertigt zu haben. Gedanken und Zitate, die ich aus fremden Quellen direkt oder indirekt übernommen habe, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und wurde bisher nicht veröffentlicht.

Ich erkläre mich damit einverstanden, dass die Arbeit auch durch das Fachgebiet Echtzeitsysteme der Öffentlichkeit zugänglich gemacht werden kann.

Darmstadt, den 7. April 2015

(J. Walker)

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Aufgaben

Im Rahmen des Projektseminars sollten verschiedene Aufgaben bearbeitet werden, die in den folgenden Unterkapiteln aufgezählt werden.

1.1.1 Abstandsregeltempomat

Grundlage für alle weiteren Aufgaben ist das Geradeausfahren und das Abbiegen. Dies musste mit der Abstandssensorik und einer geeigneter Regelung implementiert werden. Bonusaufgabe war der Entwurf einer Regelung, die jeden beliebigen Winkel zur Wand fahren und auspendeln kann.

1.1.2 Einparken

Ziel war es mithilfe von Selbstortung und Regelung das Fahrzeug in eine beliebig große Parklücke fahren zu lassen. Bei einer zu kleinen Lücke sollte der Vorgang abgebrochen werden, um im Anschluss eine neue Parklücke zu suchen. Die Implementierung des parallelen Einparkens war die Grundaufgabe. Als Bonusaufgabe konnte zusätzlich das rückwärts Einparken implementiert werden.

1.1.3 RC-Modus

Durch eine drahtlose Schnittstelle sollte es möglich sein ein Fahrzeug fernzusteuern.

1.1.4 Car2Car

Die Aufgabe bei der Car2Car-Kommunikation bestand darin, zwei Autos miteinander kommunizieren zu lassen. Sie sollten an einer Kreuzung nach vordefinierten Regeln die Vorfahrt untereinander ausmachen können.

1.1.5 Zusatzaufgaben

Zusätzlich zu den vorgegebenen Themen wurden eine Reihe von Aufgaben bearbeitet, die nicht Teil der regulären Aufgabenstellungen waren.

Car2Car
Als experimentelle Aufgaben konnte das Kommunizieren mit mehr als 2 Autos ermöglicht werden.

Ausweichen
Inspiriert von den Aufgaben der letzten Jahren wurde die Implementierung eines dynamisches Ausweichmanöver angesetzt.

1.2 Aktoren / Sensorik

1.2.1 Ultraschallmodul SRF08

Die über den I2C Bus betriebene Ultraschallsensoren sind fähig Entfernungen bis zu 6 Metern abzutasten. Im Rahmen unseres Projektseminares wurde diese auf 3 Meter begrenzt, um eine möglichst kleine Fehlerrate bei der Abtastung zu gewährleisten. Softwaremäßig war es möglich die Resultate in Zentimeter, in Zoll oder in der Dauer zwischen Senden und Empfangen des Ultraschallsignals in μs ausgeben zu lassen. Da durch die Messung der Zeit eine größere Genauigkeit erreicht werden kann, wurde mithilfe der zurückgegeben μs ein genaueres Ergebnis in Millimetern errechnet. Die Entfernung ergibt sich durch

$$\text{Entfernung} = \frac{\Delta t}{2 \cdot 3\frac{ms}{m}} \cdot$$

Δt ist die Laufzeit des Ultraschallsignals gemessen in μs. Die $3\frac{ms}{m}$ ergeben sich aus der Invertierung der Schallgeschwindigkeit (in $\frac{m}{s}$) für eine Temperatur von 20° C. Das Ergebnis muss außerdem durch zwei geteilt werden, da Hin- und Rückweg des Schalls gemessen werden.

1.2.2 ADA-IR-LINE-SENSOR

Der am unteren, vorderen Ende des Autos liegende IR-Line-Sensor besteht aus 5 Infrarotsensoren, die in dem Intervall 0-1024 arbeiten. Dabei steht 0 für die maximale Helligkeit und 1024 für die maximal Dunkelheit. Die im Sourcecode vorhandene Implementierung wurde diesbezüglich soweit optimiert (Methode „Line_really_found“ in der Datei linesensor.c), dass der gemessene Wert der Sensoren genutzt wird, um zu entscheiden, ob ein schwarzes Klebeband von mindestens 3 Sensoren erkannt wurde. Da die schwarzen Klebebänder im Durchschnitt eine Intensität von 850 aufwiesen, wurde der Wert 800 als Minimum für die Erkennung eines Klebebandes eingestellt.

1.2.3 Servo-Motor

Der Servomotor besteht aus einem kleinen Elektromotor, einer starren Übersetzung und einem Sensor für die Ermittlung der Winkelstellung. Er wird benutzt um den Winkel der Lenkachse zu verstellen. Aufgrund der unausweichlichen Abnutzung der mechanischen Teile bzw. des Potentiometers und des Getriebes, wurde eine starke Abweichung der Winkelstellung zwischen Ist- und Sollwerte zwischen den verschiedenen Autos festgestellt. Ein ID gebundener Offset innerhalb der „drive.c“ Datei verhalf zu einer Besserung. Die jeweilige tatsächliche Nullstellung der verschiedenen Autos wurde so manuell im Code eingebunden. Dies hatte den Vorteil, dass alle Autos gleiche Reaktionen im Bezug auf die Lenkung aufweisen konnten. Ein Nachteil bestand darin, dass aufgrund der großen Unterschiede zwischen den Fahrzeugen, nur noch eine effektiver Range von -75 zu 75 statt des vorherigen Range von -100 bis 100 gewährleistet werden konnte. Weitere Abnutzungen während den 4 Monaten des Projektseminars ließen die anfänglich gesetzten Werte inkorrekt werden. Dadurch mussten diese nachträglich korrigiert werden.

1.2.4 Bluetooth

RC-Modus Android App

Zur Fernsteuerung des Autos dient eine Android App. Sie schickt per Bluetooth einen ByteBuffer an das Auto. Der ByteBuffer hat folgendes Format:

START = 2	START = 139	SRC-ID	DEST-ID	MSG-TYPE	PRIORITY	?	ARRAY-LENGTH	MOTOR	SERVO	STATE	STOP = 3	STOP = 3	STOP = 3	STOP = 3	STOP = 3	STOP = 3

Im Coderahmen ist schon eine Empfangsmethode implementiert. Somit lassen sich per Bluetooth standardmäßig MotorSpeed und ServoStellung steuern. Ein zusätzliches Feature der Android-App ist das manuelle Wechseln zwischen Zuständen. Wichtig hierfür ist Array-Length. Der Wert in Array-Length definiert wie viele Elemente vor den Stoppbytes noch folgen. Standardmäßig sind es 2 weitere Elemente für Motor und Servo. Die Android App verschickt ein weiteres Element, um den Zustand zu steuern. Es ist nicht möglich negative Zahlen zu versenden. Aus diesem Grund wird der Motorwert mit 43 und der Servowert mit 10 addiert. Im Code des Autos werden diese Werte wieder subtrahiert. Somit wird das versenden vorzeichenbehafteter Zahlen umgangen.

Empfängerseite RC-Modus

In der Datei AndroidBTControl.c wird die Methode motorControlTest_bt abgeändert:

```
1  uint8_t lastRcIn = 9;
2
3  void motorControlTest_bt(wireless_interface_t wiif,
4                           wirelessMessage_t* msg) {
5          if(msg->data[2] == 5){
6                  Drive_SetMotor(msg->data[0] - 43);
7                  Drive_SetServo((msg->data[1] - 10) * 10);
8          }
9          if(lastRcIn != msg->data[2]){
10                 RcInput(msg->data[2]);
11         }
12         lastRcIn = msg->data[2];
13 }
```

Standardmässig hat das Array data[] 2 Elemente. Das dritte Element wird dazu benutzt, den gewünschten Zustand zu übertragen. So kann der Nutzer zusätzlich zum Fernsteuern, das Auto auch in andere Zustände versetzen.

2 Implementierung

2.1 Wallfollower

Die Implementierung des Wallfollowers stellte die erste Aufgabe des Projektseminars dar, auf der alle weiteren Aufgabenteile und Funktionen des Autos beruhten. Darum war es um so wichtiger bei ihr ein sehr gutes Ergebnis zu erzielen. Das Auto sollte

so geregelt werden, dass es mit Hilfe des rechten Ultraschallsensors möglichst gerade mit einem vorgegebenen Abstand an einer Wand entlang fahren kann. Die erste Idee war eine lineare Implementierung. Nach deren Umsetzung stellte sich heraus, dass das Auto anfing in immer größeren Bögen um den vorgegebenen Abstandswert zu schwingen bis es schließlich die Wand berührte. Die darauf aufbauende zweite Idee, war die Implementierung eines Reglers realisiert durch eine Funktion dritten Gerades. Das Resultat dieser Implementierung war eine funktionierende Regelung, die zwar noch von Schwingungen gezeichnet war, aber ihren Zweck erfüllte. Im Anschluss wurden wir auf das Prinzip des PID-Regler aufmerksam, der sich als unser endgültig benutzter Regler herausstellte, da er das gewünschte Fahrverhalten erbrachte.

PID-Regler

Der PID-Regler besteht aus drei Gliedern. Dem Proportionalglied P, dem Differentialglied D und dem Integralglied I. Diese werden durch die jeweilige Multiplikation mit dem Faktor p, d und i gewichtet addiert und ergeben die Regelgröße, die der Servolenkung übermittelt wird. Das P-Glied vergleicht die Abweichung des Istwerts von dem Sollwert. Der Differentialteil reagiert auf die Änderungsgeschwindigkeit. Er vergleicht die Abweichung des aktuellen Messwerts zum vorherigen Messwert. Der Integralanteil wirkt durch die Summe der letzten Abweichungen zwischen Ist- und Sollwert auf die Stellgröße ein. Allgemein hat der PID-Regler die allgemeine Form

$$y(t) = p \cdot e(t) + d \cdot \frac{de(t)}{dt} + i \cdot \int_0^t e(\tau)\, d\tau,$$

wobei $y(t)$ die Stellgröße und $e(t)$ die Differenz zwischen Ist- und Sollwert darstellt. Die Gewichtung der Faktoren wurde durch empirisches Einstellen gewonnen. Der D-Anteil wurde dabei mit der größten und der I-Anteil mit einer verschwindend geringen Gewichtung bemessen.

Türen

Der Flur des Fachgebiets Echtzeitsysteme diente als Testumgebung. An seinen Wänden sind Türen die ca. 3 cm heraus- und Türrahmen, die ca. 5 cm hineinragen. Diese Gegebenheiten erschwerten das geradeaus Fahren, da der Ultraschallsensor auf ihrer Höhe immer mit kleinen Sprüngen in den Messwerten zurechtkommen musste. Es wurde versucht, die kleinen Sprünge in der Implementierung zu berücksichtigen und aus dem Regelverhalten zu eliminieren. Die Anpassung führte leider nicht zu dem gewünschten Ergebnis. Teils wurden die Sprünge erkannt und teils wurden sie übersehen. Eine Mittelung der Messwerte durch einen Ringpuffer brachte auch keinen Vorteil, sondern nur eine zeitliche Verzögerung in der Regelung. Da sich keine Verbesserung einstellte, wurde die Idee der Tür- und Türrahmenerkennung verworfen. Durch Erhöhen des D-Anteils und Verringern des P-Anteils wurde ein gutes Ergebnis im Bezug auf die Reaktion der Türrahmen erreicht.

Genauigkeit

Die Umstellung der Messung der Abstandssensoren von *cm* auf *mm* verschaffte zusätzliche Genauigkeit in dem Verhalten des PID-Reglers.

2.2 Rechtsabbiegen / Abzweigungen

Eine Gablung ist wie folgt definiert: Jede Kurve und jede Art von Kreuzungen sind Gablungen. In einem Array werden die gewünschten Richtungen für die nächsten 4 Gablungen gespeichert. Diese können im Quellcode angepasst werden. Wenn das Auto eine Gablung erkennt, so prüft es, ob es in die gewünschte Richtung abbiegen kann. Zur Verfügung stehen links=0, geradeaus=1 und rechts=2 . Wenn das Auto eine Links- oder Rechtskurve erkennt, wechselt es in den Gablungszustand.

Folgende Signale werden auf 1 gesetzt, wenn das jeweilige Ereignis zutrifft.

KannLinks = 1 , wenn das Auto eine Linkskurve erkennt.
kannRechts = 1, wenn das Auto eine Rechtskurve erkennt.
kannVorne = 1, wenn der Abstand nach vorne ausreichend groß ist.

Abhängig von den Abzweigmöglichkeiten und der geplanten Abzweigung wird nun bestimmt, in welche Richtung das Auto abbiegt. Aus dem Kurvenradius lässt sich der Umfang eines Viertelkreises berechnen. Dieser beträgt 1178mm. Wenn das Auto bei vollem Ausschlag mehr als 1178mm, oder 1000mm geradeaus gefahren ist, ist die Kurve abgeschlossen. Danach wechselt das Auto wieder in den „Fahren“-Zustand. Zur Veranschaulichung dient das Zustandsdiagramm in Abbildung 2.3 am Ende dieses Kapitels.

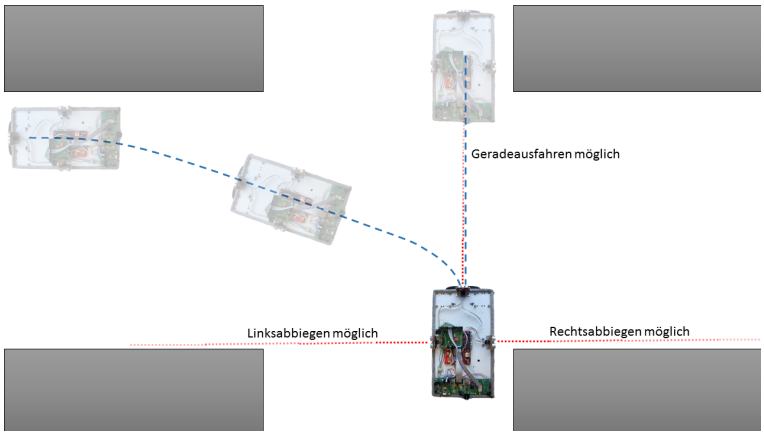

Abbildung 2.1: Abzweigung

Der Oben beschriebene Fall deckt 90° Kurven ab. Kleinere Kurven können sogar schon mit dem PID-Regler fehlerfrei gefahren werden. Bei größeren Kurven fährt das Auto erst die 90° und anschließend, setzt der PID-Regler ein. (Siehe Kapitel WallFollower)

Im folgenden sehen sie das reine Rechtsabbiegen im genauen Detail.

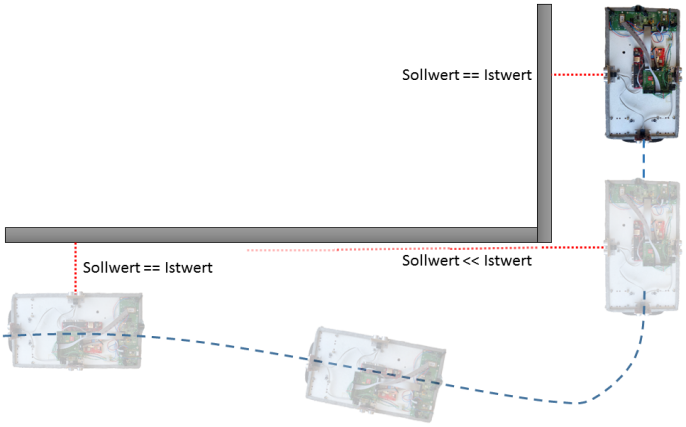

Abbildung 2.2: Rechtsabbiegen

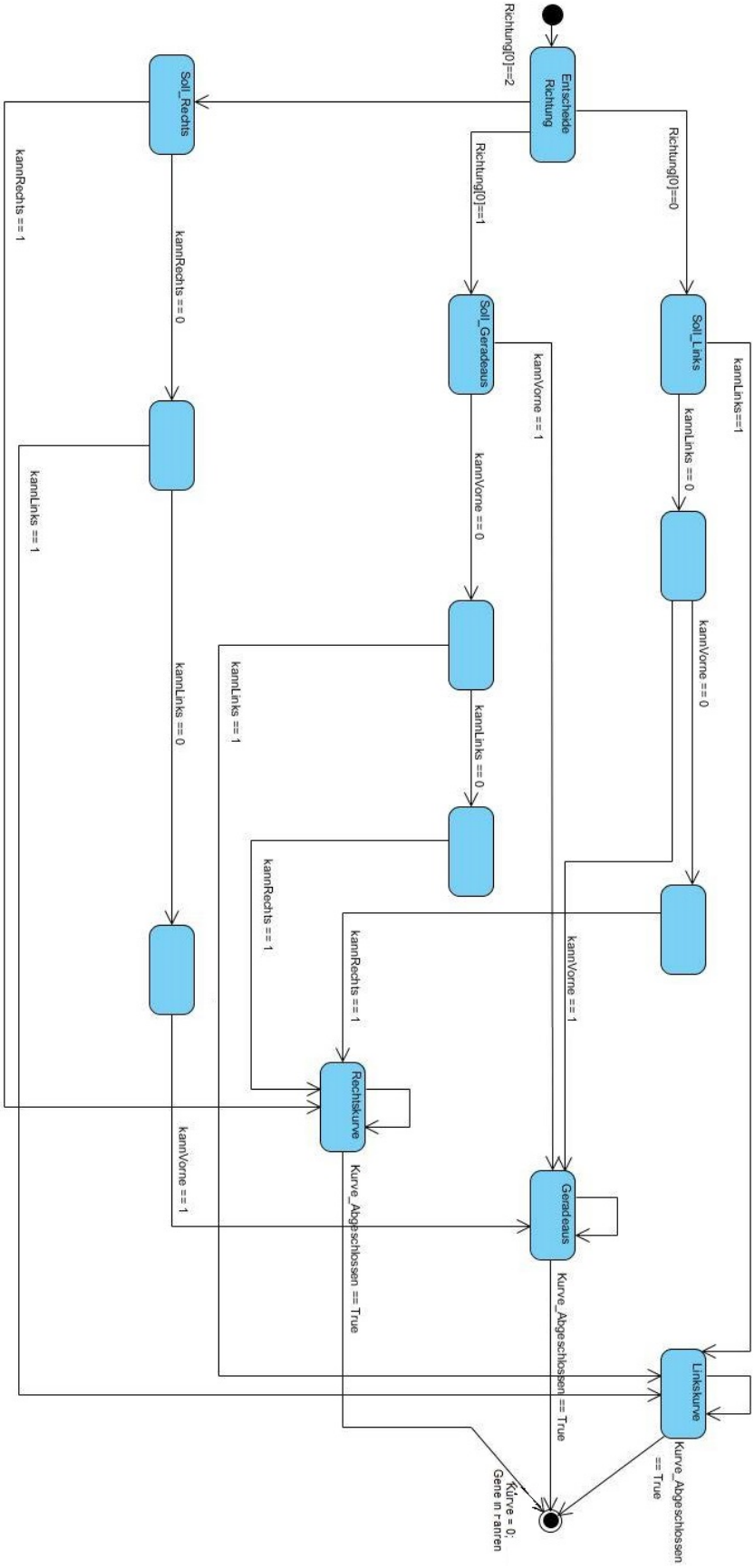

Abbildung 2.3: Kurvenentscheidung des Autos

2.3 Paralleles Einparken

Bei dem Vorgang des parallelen Einparkens muss das Auto autonom in eine parallel zur Fahrtrichtung befindliche Parklücke einparken. Abbildung 2.4 veranschaulicht diesen Vorgang und soll dem besseren Verständnis dienen.

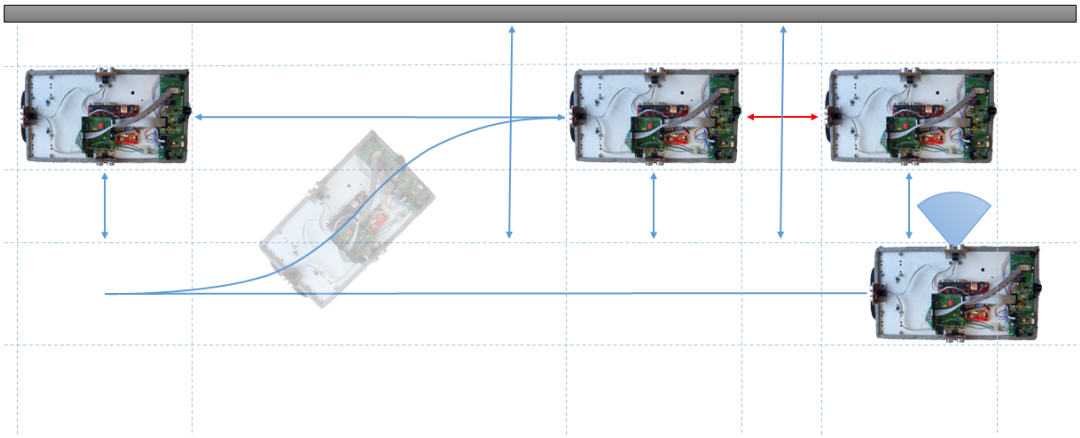

Abbildung 2.4: Modell der Vorgehensweise beim parallelen Einparken

In unserem Szenario herrscht Rechtsverkehr und es muss in der Regel in eine auf der rechten Straßenseite befindliche Straßenlücke eingeparkt werden. Aus diesem Grund wird für die Berechnungen der rechte Ultraschallsensor genutzt. Ausgangspunkt des Einparkszenarios ist der Wallfollower Modus, in dem das Auto mit Hilfe des rechten Abstandsensors geradeaus an der rechts von ihm liegenden Wand entlang fährt. Dabei ist der Abstand zwischen dem Auto und der Wand festgelegt und so gewählt, dass an parkenden Autos vorbeigefahren werden kann ohne Rücksicht nehmen zu müssen. Im Wallfollower Modus wird der PID-Regler genutzt, um das Auto in der Spur zu halten. Zu Beginn muss ein Auto auf der rechten Seite durch einen Sprung in den Abstandsmesswerten erkannt werden. Sobald ein weiterer Sprung in den Abstandsmesswerten das Ende des auf der rechten Seite stehenden Autos signalisiert, fängt das fahrende Auto an, die ab jetzt gefahrene Strecke auszumessen. Wird ein weiteres Auto erkannt werden die Schritte erneut durchlaufen und eine neue Messung wird gestartet. Wenn bei dem Messvorgang die zum Einparken benötigte Länge erreicht wird, sind nachfolgende Autos nicht mehr relevant und der Einparkvorgang wird gestartet. Dazu fährt das Auto wegen seines großen Kurven Radius eine bestimmte Strecke weiter. Das darauf folgende rückwärts Fahren besteht aus zwei Teilen. Erst wird voll nach links eingeschlagen und dann voll nach rechts. Wie weit das Auto rückwärts fahren muss wurde aus dem Abstand zwischen dem Auto und der rechten Wand in paralleler Ausgangsposition hergeleitet. Zum Schluss des Parkvorgangs, also nachdem diese Strecke gefahren wurde, hält das Auto an und der Einparkvorgang ist beendet. In Abbildung 2.5 sind die Zustände des parallelen Parkszenarios grafisch dargestellt.

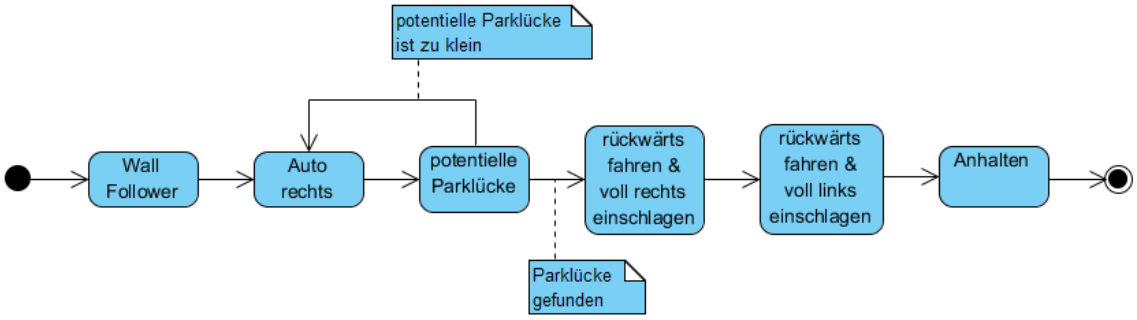

Abbildung 2.5: Zustandsdiagramm des parallelen Einparkvorgangs

2.4 Rückwärts Einparken

Bei dem Vorgang des rückwärts Einparkens hingegen muss das Auto autonom in eine rechtwinkelig zur Fahrtrichtung befindliche Parklücke einparken. Abbildung 2.6 veranschaulicht diesen Vorgang.

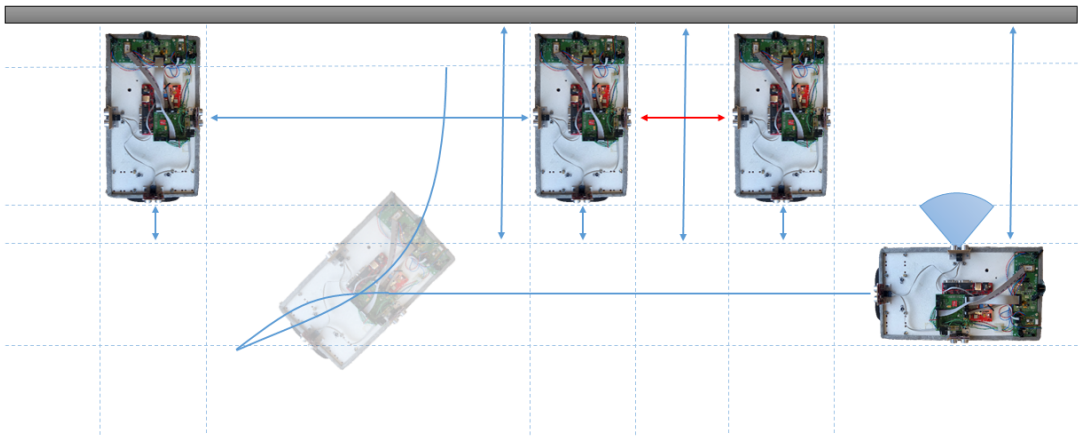

Abbildung 2.6: Modell der Vorgehensweise beim Rückwärts Einparken

Die Funktionsweise beim rückwärts Einparken ist der des parallelen Einparkens sehr ähnlich. Der Ausgangspunkt ist wieder der Wallfollower Modus und der Abstand zur Wand, der von dem PID-Regler geregelt wird, ist wieder so groß, dass an parkenden Autos problemlos vorbeigefahren werden kann. Die Vorgehensweise des Suchens einer Parklücke ist identisch mit der beim parallelen Einparken. Einzig und allein der Abstand zwischen den parkenden Autos muss im Code angepasst werden. Aus diesem Grund wird hier auf die Erklärung der Funktionsweise der Detektion einer Parklücke nicht näher eingegangen. Der Blick wird nun auf den Zeitpunkt nach dem Fund einer Parklücke gelegt. Da der Kurvenradius des Autos sehr groß ist, muss im ersten Schritt voll links eingeschlagen werden und ein weiteres Stück geradeaus gefahren werden. Beim darauffolgenden rückwärts Fahren wird dann voll nach rechts eingeschlagen. Der zurückgelegte Weg wurde aus dem Abstand zwischen dem Auto und der rechten Wand in paralleler Ausgangsposition hergeleitet und durch Praktische Tests leicht korrigiert.

Nachdem das Auto diesen Weg zurückgelegt hat stoppt es und das Programm terminiert. In Abbildung 2.7 sind die Zustände des Parkszenarios beim rückwärts Einparken grafisch dargestellt.

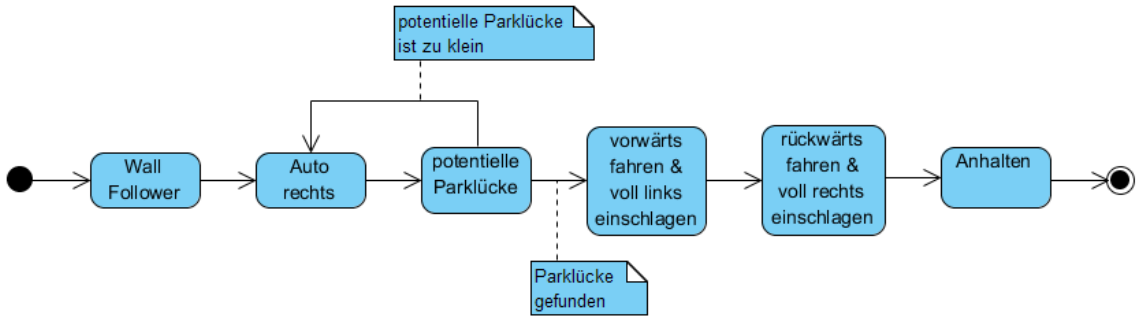

Abbildung 2.7: Zustandsdiagramm des parallelen Einparkvorgangs

2.5 Car2Car

Grundlage der Implementierung entsprang der Idee, ein erweiterbares System zu programmieren, welches erlaubt infinite Anzahl von Fahrzeugen zu organisieren. Jedes Fahrzeug besitzt eine globales Array mit Information über die Reihenfolge der Fahrzeuge, welche an der Kreuzung stehen. Das System kann soweit erweitert werden, dass alle gängigen Scheduling Prozesse theoretisch implementiert werden könnten. Das Team entschloss sich zuerst den Prozess nach First-in-First-out zu programmieren, um die grundlegende Funktion zu testen. Wie beim RC-Modus, musste zuerst ein Listener - in unserem Falle die Methode „subC2C“ - implementiert werden, welches sich bei Aufruf subscribed und auf GPS Signale mit dem richtigen Protokoll wartet. Im folgenden Abschnitt können sie den Ablauf nach FIFO Prinzip beobachten.

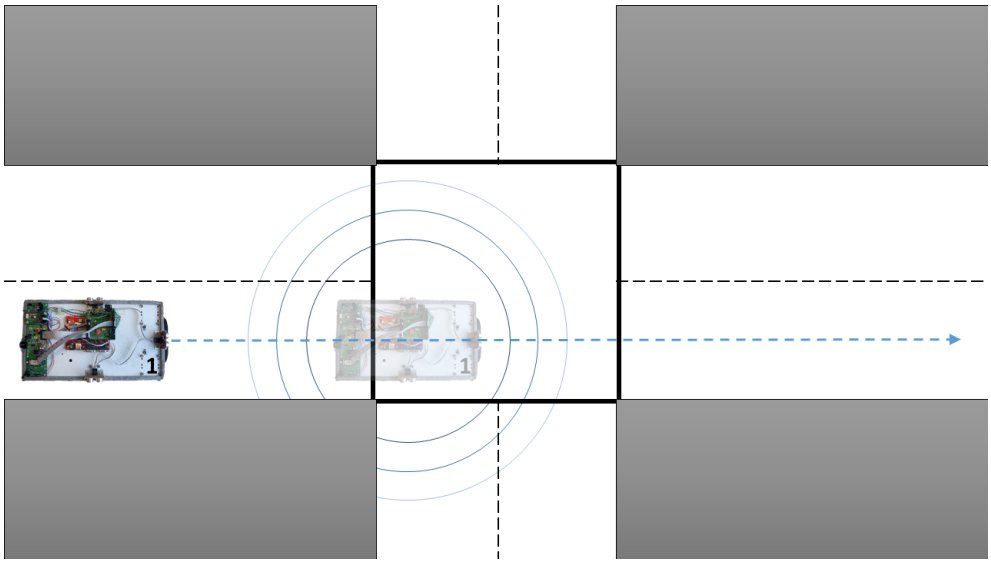

Abbildung 2.8: Auto 1 erreicht Kreuzung

Das erste Fahrzeug gelangt an die Kreuzung, nimmt folglich kein Signal eines anderen Fahrzeuges wahr und nimmt sich die Vorfahrt. Ab diesem Moment ist das Auto auf seiner lokalen Liste auf dem ersten Platz, diese Information wird mit folgendem Protokoll, sowie mithilfe der Broadcast-Funktion des GPS-Moduls im Umkreis versendet.

```
1 mess.destinationId=255; //255 == Broadcast an alle
2 mess.priority=1;
3 mess.sourceId=carid;
4 mess.messageType=4;
5 mess.dataLength=2;
6 mess.data[0]=carid;
7 mess.data[1]=x;
```

Da der Vorgang der Informationsübergabe bis zum Verlassen der Kreuzung anhält kann hier garantiert werden, das alle Fahrzeuge, die die Kreuzung betreten wollen diese Information erhalten. Ab diesem Moment tritt die 2te Phase ein. Ankommende Autos nehmen das Signal mithilfe der SubC2C und der Stop Methode auf und setzen alle auftretenden Informationen auf ihre lokale Liste, in diesem Falle, die des ersten Fahrzeuges. Sobald Fahrzeug 2 an der Kreuzungsmarkierung angekommen ist, wird dieses aufgrund der besetzten Kreuzung und der Information eines noch weiteren existenten Fahrzeuges, welches an der ersten Stelle der lokalen Liste steht, zum stehen kommen. Der Informationsabfang , ob das fahrende Auto die Kreuzung verlässt, wird stetig geprüft.

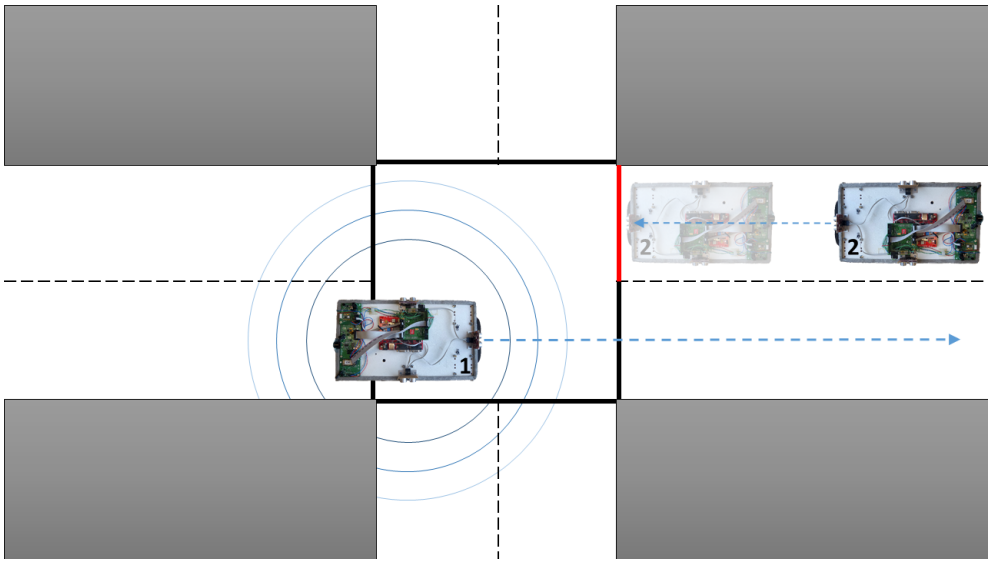

Abbildung 2.9: Auto 2 erreicht Kreuzung

Nun setzt Auto 2 sich an 2ter Stelle dieser Liste und sendet diese Information ebenfalls mit „SendAround“ im Broadcast herum, um andere Autos von der Existenz zu informieren. Es wird ebenfalls geprüft, dass sich das Auto nicht mehrfach auf der Liste befinden kann. Um nun die Idee mit 3 Autos zu erklären kommt ein weiteres Auto an die Kreuzung.

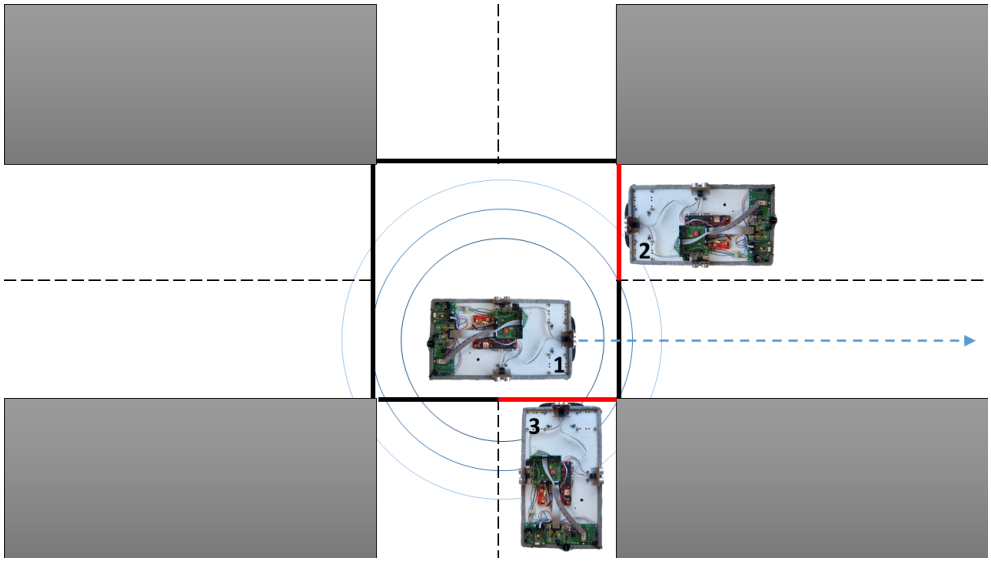

Abbildung 2.10: Auto 3 erreicht Kreuzung

Dieses hat nun zusätzlich die Information, dass sich ein weiteres Auto an der Kreuzung befindet. Würde ab hier mit einem anderen Scheduling Verfahren als FIFO gearbeitet werden, könnte bei der Ankuft des dritten Autos eine lokale Umverteilung ad-hoc stattfinden. So würde das Auto, welches beispielsweise eine höhere ID als das bereits stehende Auto hat, vor diesem beginnen sich in Bewegung zu setzen.

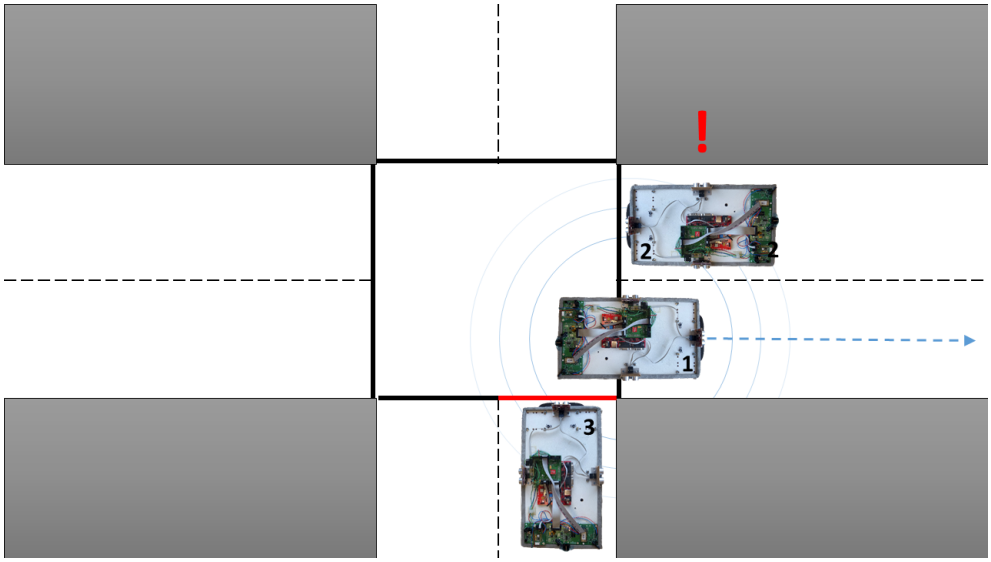

Abbildung 2.11: Auto 1 verlässt die Kreuzung

Auto 1 hat nun die Kreuzung verlassen. Die Information des Verschwindens wird ab dem Punkt, wo das Auto den 2ten Streifen überfahren hat, wiederum im Broadcast an alle Autos gesendet. Nun beginnt die Umverteilung im FIFO-Verfahren: Jedes Auto registriert das Verschwinden und die lokalen Listen werden aktualisiert. Dies passiert, indem der Arrayinhalt an der Stelle 0 mit der Stelle 1 (und so weiter) ersetzt wird.

So rückt jedes Auto eine Stelle auf und überprüft ob es nun an Stelle 0 des Arrays steht. Ist dieser Fall eingetreten, darf das Auto sich die Vorfahrt nehmen, ändert seine im Broadcast verteilte Information von stehend auf „belege Kreuzung“ und sendet dies wiederum herum.

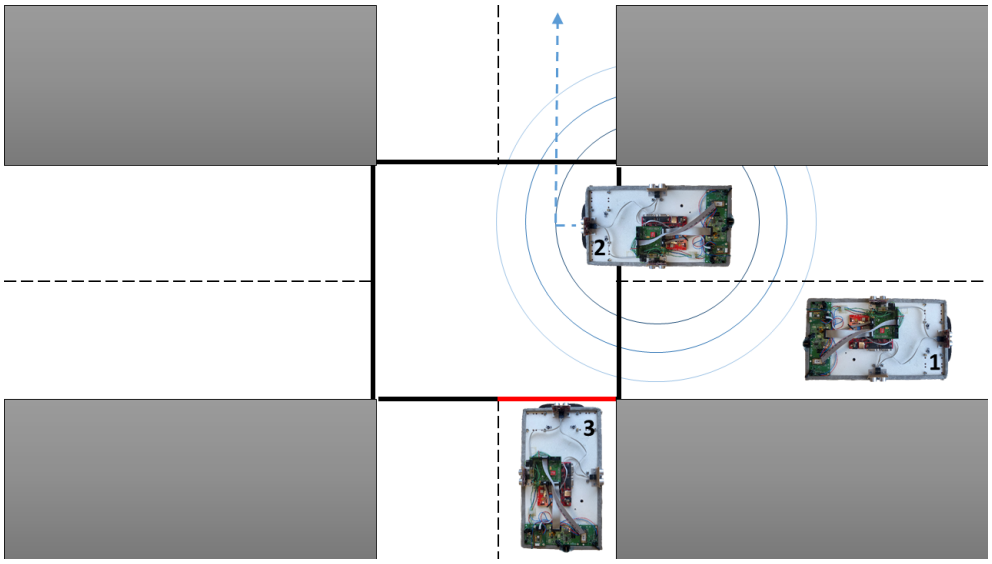

Abbildung 2.12: Auto 1 fährt über die Kreuzung

Auto 3 belegt nun nach Auto 2 die 2te Stelle jenes lokalen Arrays. Falls ein neues Auto ankommen würde, würde dieses sich hinter Auto 3 anreihen, da dieses System, aufgebaut wie ein Peer-to-Peer Netzwerk, immer die aktuellsten Informationen beinhaltet.

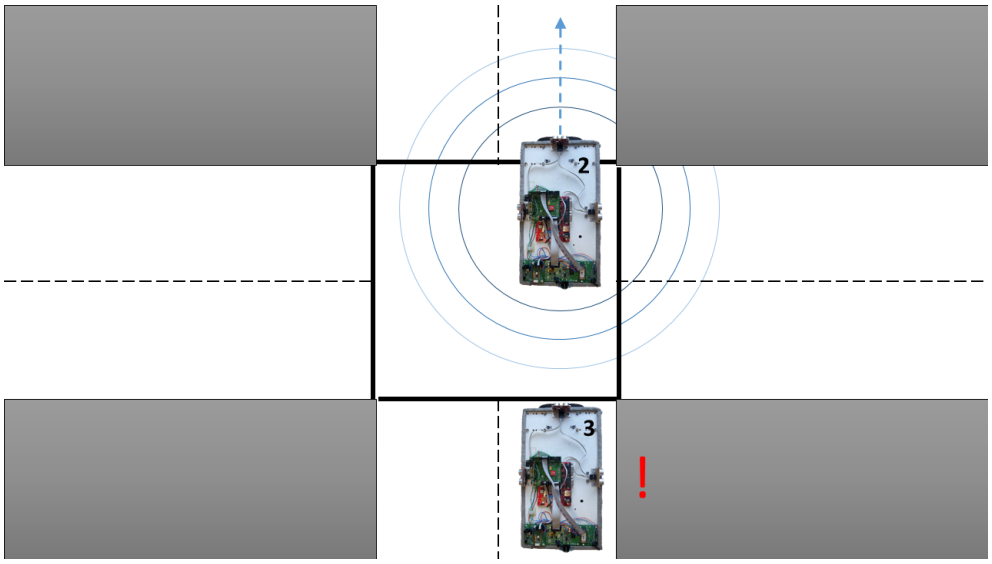

Abbildung 2.13: Auto 2 verlässt die Kreuzung

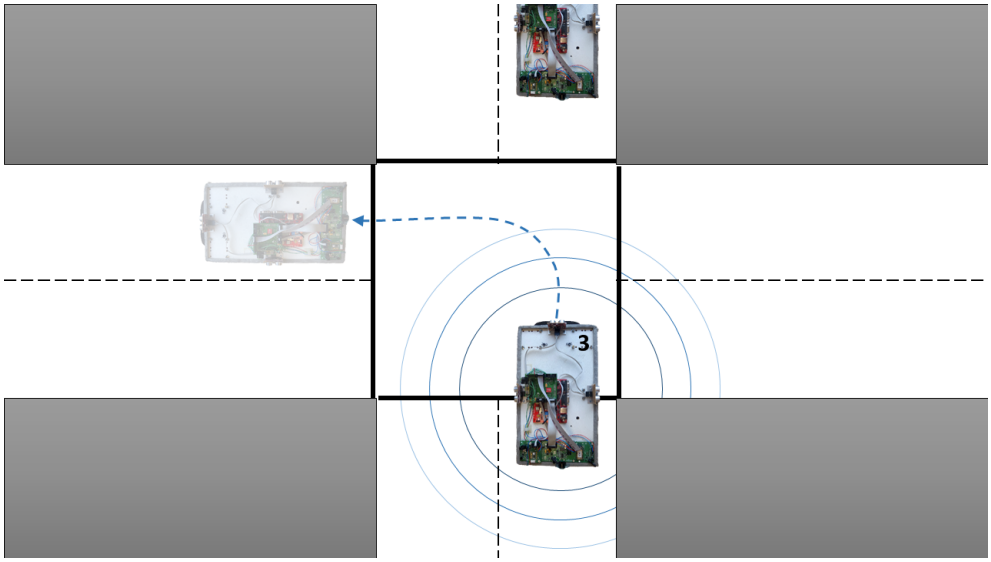

Abbildung 2.14: Auto 3 fährt über die Kreuzung

Nach dem Verlassen jedes Fahrzeuges, wird mithilfe der memset-Funktion die Prioritätsliste geleert, sodass ein 2tes ankommen an einer Kreuzung stattfinden kann, ohne das System resetten zu müssen. Alle Funktionen, vorrangig PID-Regler und Abbiegen funktionieren hier unabhängig von dem Car-2-Car Prozess und werden im Falle des stehenden Autos lediglich de- und dann wieder reaktiviert. Zwischen der Ankunft zweier Autos muss nach den Messungen der Gruppe nur eine halbe Sekunde liegen, um einen Reibungslosen ablauf zu gewährleisten.

2.6 Ausweichen / Spurwechsel

Der Vorgang des Ausweichens bzw. des Spurwechsels ist eine untergeordnete Funktion des PID-Reglers. Dieser wird, sobald der Frontsensor einen zu geringen Abstand registriert, deaktiviert und das Ausweichmanöver gestartet. Das Ausweichmanöver läuft prinzipiell in drei Stufen ab.

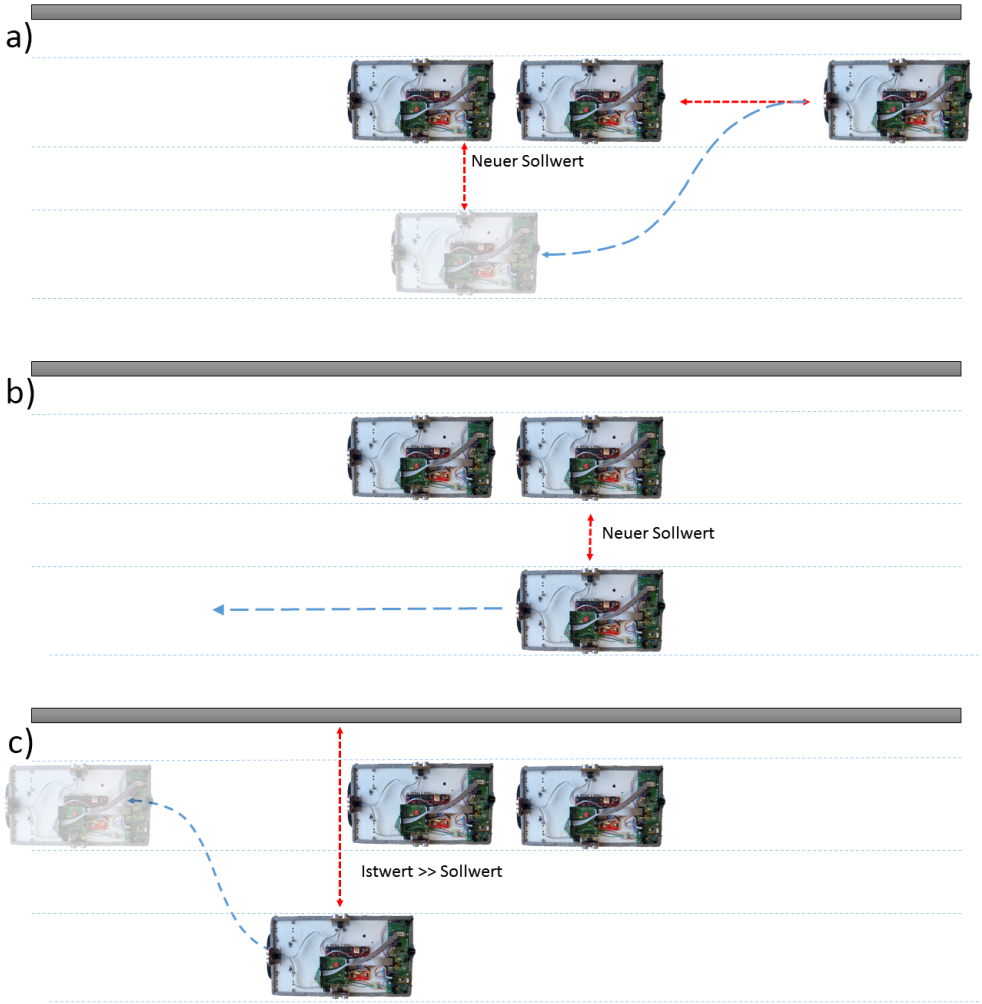

Abbildung 2.15: Ausweichvorgang. a) ausscheren nach links b) Auto neben Hindernis c) einscheren nach rechts

Anhand der Fahrzeuggeschwindigkeit wird der Reglersollwert solange von der Wand weg korrigiert, bis der Frontsensor keinen zu kleinen Abstand mehr registriert. Hierbei wird mit Hilfe einer einfachen Formel

$$\text{Regler\_set\_sollwert}(400 + 300 * \text{counter});$$

und mehreren Iterationen gearbeitet. Der konstante Wert 400 beschreibt hier den Grundabstand zur Wand und kann beliebig geändert werden. 300*counter sind hier die Stufen in Millimetern, die das Fahrzeug korrigiert.

Ist das Hindernis auf der rechten Seite nicht mehr im Sichtfeld, so wird nicht weiter gelenkt. Es wird darauf gewartet, dass das zu umfahrende Hindernis mit dem rechten Ultraschallsensor aufgenommen wird. Ab diesem Punkt wird der PID-Regler auf 100mm angeglichen, um nach der Umfahrung schnell wieder in die Normalspur zu kommen.

Ist die Umfahrung erfolgreich und meldet der rechte Ultraschallsensor das Verschwinden des Hindernisses, so wird der PID-Regler wieder auf Normalabstand gesetzt. Sonderfall des Ausweichmechanismusses ist, wenn der Abstand nach vorne zu klein ist um Auszuweichen. Extremfall ist das Befahren einer Sackgasse. Hier wird die Motorgeschwindigkeit negiert und der Servo, wegen des Rechtsverkehrs, auf eine postive Zahl gesetzt. Das bedeutet, dass das Fahrzeug nach rechts ausschert. Das anschließenden Zurückfahren wird vom Frontsensor begrenzt, denn sobald der Abstand nach vorne groß genug ist, wird versucht nach links zu fahren und so einen U-Turn zu vollenden. Gelingt dies, so wird der PID-Regler wieder regulär eingeschaltet. Andernfalls wird der Prozess so lange wiederholt, bis der U-Turn vollendet ist.

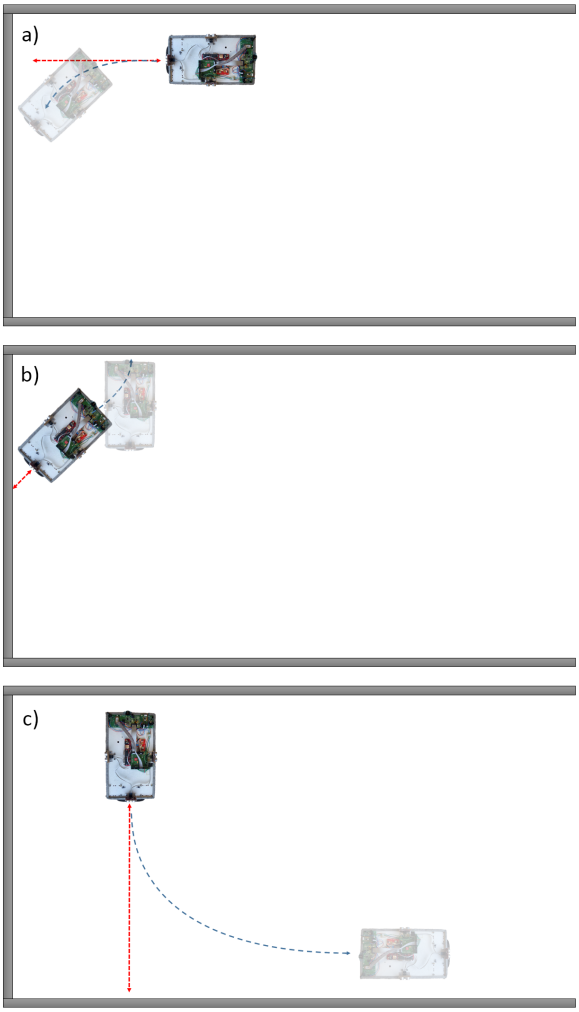

Abbildung 2.16: U-Turn

3 Zustandsautomat des Autos

Die Teilfunktionen des Autos werden von einem Main-Task gesteuert. Dieser ruft abhängig vom aktuellen Zustand die dazugehörige Teilfunktion auf. Der Startzustand ist die Teilfunktion "Fahre In einem Maintask ist der Zustandsautomat implementiert. Dieser Steuert das gesamte Auto und wird 10 Mal pro Sekunde aufgerufen. Der Zustandsautomat wird durch ein Switchcase und ein getNaechstenZustand(aktuellerZustand); realisiert. Beide befinden sich in einer While-Schleife. Zuerst gibt "getNachstenZustand(aktuellerZustand);"den nächsten Zustand zurück. Anschließend wechselt die SwitchcaseAnweisung in den nächsten Zustand. Die Transitionen werden also in getNaechstenZustand(); behandelt. Die Zustände im Switchcase. Für eine bessere Übersicht wird getNaechstenZustand() in ein eigenes C-Dokument ausgelagert. GetNaechstenZustand(); braucht nämlich auch eine SwitchCase Anweisung, da sich die Transitionen verschiedener Zustände unterscheiden können.

Zustände:
Folgende Zustände modellieren das Verhalten des Autos:

Fahren:
Im Zustand Fahren passt das Auto den Abstand nach rechts an den Sollwert an und fährt die Wand entlang.
Transition...
...in Gablung:
Wenn das Auto eine Rechts- oder Linkskurve erkennt.
...in Ausweichen:
Wenn das Auto ein Hindernis vorne erkennt, *...in RC-Modus:*
Wenn in der App der passende Button gedrückt wird.
...in Parken parallel:
Wenn in der App der passende Button gedrückt wird.
...in Parken Rückwärts:
Wenn in der App der passende Button gedrückt wird.

Gablung:
Das Auto fährt entweder eine 90° Linkskurve, eine 90° Rechtskurve oder geradeaus.
Transition...
...in Fahren:
Wenn der Kurvenvorgang abschlossen ist. Entweder hat das Auto einen Viertelkreis zurückgelegt, oder ist 1m geradeaus gefahren.
...in Ausweichen:
Wenn ein Hindernis von dem vorderen Ultraschallsensor detektiert wird.

Ausweichen:
Das Auto weicht einem Hindernis aus.
Transition...
...in Fahren:
Wenn das Ausweichmanöver beendet ist.

Parken_Parallel:
Das Auto parkt parallel ein.
Transition
Terminiert nach Parkvorgang

Parken_Rückwärts:
Das Auto parkt rückwärts ein.
Transition
Terminiert nach Parkvorgang

RC_Modus:
Erlaubt dem Nutzer das Auto fernzusteuern und gewisse Zustände zu wechseln.
Transition...
...in Fahren:
Wenn der User auf „Fahren“ klickt.
...in Parken_Parallel:
Wenn der User auf „Parken_Parallel“ klickt.
...in Parken_Rückwärts:
Wenn der User auf „Parken_Rückwärts“ klickt.

4 Probleme

Ultraschallsensor

Aufgrund der Funktionsweise des Ultraschallsensors, welche darauf beruht, dass der ausgesendete Ultraschallimpuls von einem Objekt reflektiert und zurückgeworfen wird, kann so eine Messung nach dem Laufzeitprinzip errechnet werden. Eine Spiegelung bzw. starke Streuung ist zu erwarten, sobald der ausgesendete Impuls mit weniger als ungefähr 45 Grad auf ein Objekt trifft. Dieses Problem war besonders auffällig bei Abbiegevorgängen, da sich der PID-Regler nach abschließendem Abbiegen der Wand mit maximalem Einschlagwinkel annähert, da der Ultraschallsensor, aufgrund der <45 Grad Problematik, einen zu großen Abstandswert aufgenommen hat.

Abbildung 4.1: 45° Problematik

Dies war nicht nur bei dem rechten Ultraschallsensor der Fall, sondern auch bei allen anderen und wurde eingegrenzt, indem das Auto solange schwach im konstanten Winkel lenkt, bis es in einen gewissen Abstand zur Wand erreicht. Zudem besitzt der Verwendete Ultraschallsensor „400ST160“ , genau wie die meisten anderen Ultraschallsensoren eine Streuung bereits ab Abstrahlpunkt, sodass bei zu nahen Passagen zur Wand falsche Werte zurückgeworfen wurden.

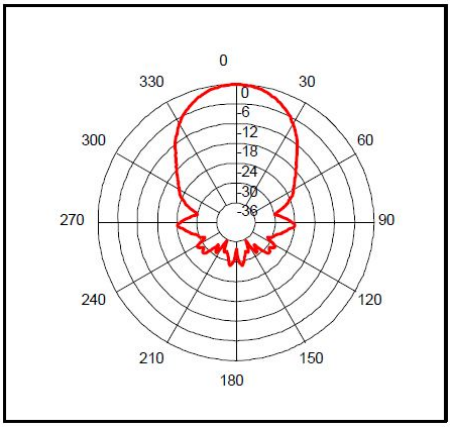

Abbildung 4.2: Ultraschallimpuls

Lösung des Problems wäre eine optische Abstandmessung, da diese mit ihrer punktgenauen Laserabtastung die Wahrscheinlichkeit des Streuens minimiert und die Genauigkeit maximiert. Jedoch kann ein solches Verfahren nicht auf allen Materialien angewendet werden, da einige den Laser absorbieren.

Kurvenerkennung

Bei der Kurvenerkennung können folgende Probleme auftreten: Wenn das Auto nicht parallel in die Kreuzung rein fährt, dann kann es sein, dass es nur eine Kurvenmöglichkeit erkennt. (Abbildung) Das Auto kann z.B. geradeaus UND links fahren, erkennt aber nur eine Linkskurve.

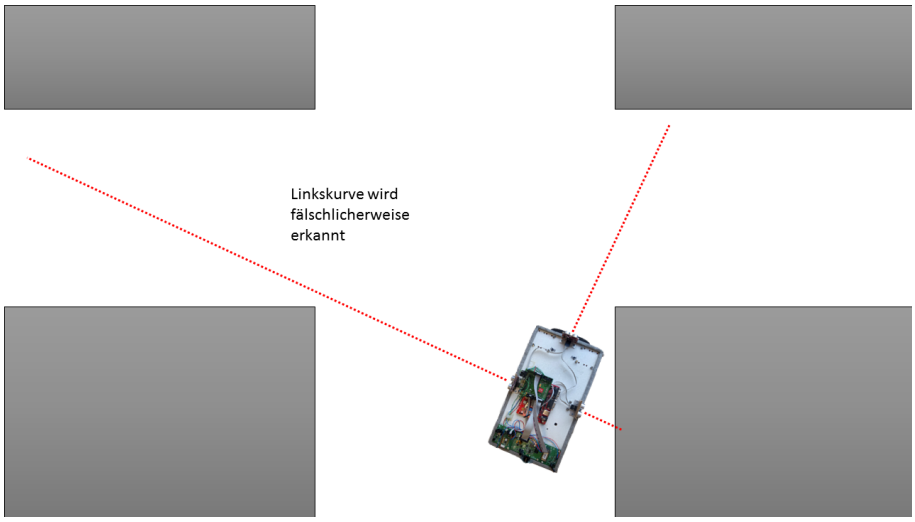

Abbildung 4.3: Linksabbiegen

GPS

Während der Benutzung des Programm GPS-Config fiel auf, dass nach dem Start der Autos kein GPS-Signal versendet wird. Erst nach einem erneuten Reset über den auf der Platine befindlichen Knopf wurde das Startsignal versendet. Dies war auch beim Testen im Car2Car Szenario zu bemerken und besonders ärgerlich, wenn mit mehreren Autos gearbeitet wurde. Dadurch konnte ohne PC und GPS-Config nicht garantiert werden, ob das Auto des versenden eines Signals fähig war. Ebenso konnten an den Kurven oftmals Signale verloren gehen, da die Wände diese zerstreuen oder absorbieren.

5 Fazit

5.1 Ergebnis

Zu Beginn des Seminars haben wir uns entschlossen gemeinsam grundlegende Implementierungen vorzunehmen, damit jedes einzelne Gruppenmitglied mit den selben Startvoraussetzungen in die jeweilige Aufgabe einsteigen kann. Dieser Weg wurde absichtlich eingeschlagen, damit im späteren Verlauf einfache Fehlerquellen selbst identifiziert und behoben werden konnten, ohne ein weiteres Gruppenmitglied fragen zu müssen. Mit fortschreitender Zeit konnte so eine gemeinschaftliche Verbesserung der Kenntnisse um das Auto errungen werden. Nach der Entscheidung einen PID-Regler zu implementieren, wurde die Gruppe erstmals in zwei Teilgruppen aufgeteilt. Eine, die sich um die Erstellung des Reglers kümmert und eine weitere, die sich mit dem Abbiegen beschäftigt. Wallfollower und Rechtsabbiegen waren somit die ersten Szenarien, die bereits recht früh in ihrer grundlegenden Form funktionsfähig waren. Diese wurden nur noch in ihren Parametern und Eintrittsbedingungen geändert. Viele kleinere Veränderungen wurden nur noch aufgrund von Abhängigkeit anderer Szenarien getroffen. Die Wegfindung (Gablung) wurde stetig von einem Teammitglied als Nebenaufgabe bearbeitet und währte durch das gesamte Projektseminar. Das parallele Einparken wurde aufbauend auf dem funktionierenden PID-Regler programmiert. Aufgrund des fehlendem Heck-Ultraschall Sensors musste hier experimentiert und mit vielen Parametern gearbeitet werden, dies ging jedoch relativ schnell von der Hand, sodass sogar Augenmerk auf das Rückwärtseinparken gelegt werden konnte. Dies war nach dem abarbeiten des Paralleleinparkens nicht mehr so schwer, da größtenteils nur noch Parameter von Servo und Ultraschallsensoren verändert werden mussten. Parallel dazu wurde bereits der RC-Mode mit Hilfe von Androidstudio als Android App programmiert, Schwierigkeiten um das Bluetooth Protokoll und des Fujitsu Autos ließen diese Aufgabe besonders schwer erscheinen, da ein Problem bei uns zu Tage kam, welches sich keiner Erklären konnte. Durch Zufall konnte dieses jedoch gelöst werden und diese Aufgabe war Mitte Januar auch gelöst. Im Szenario Car2Car konnte durch ein gut durchdachtes Konzept die Aufgabe schneller gelöst werden als gedacht. Dies motivierte uns dazu weiter zu gehen als nötig - wenige Abstimmungen reichten bereits aus um mehr als zwei Autos in das Car2Car System einzubinden. Dies zeigte uns, das vorschnelles Handeln nicht die beste Wahl in vielen Situationen in einem solch großen Projekt ist und durch überlegtes Handeln viel Zeit gespart werden kann. Nebenbei wurde ebenfalls ein Ausweichmechanismus entwickelt, welcher sich in vielen Situationen als nützlich erwies. Für die Best-Lap um die Teeküche im HBI wurden nur noch Veränderungen bezüglich Speed und PID-Regler vorgenommen und so eine persönliche Bestzeit von 25 Sekunden erreicht.

5.2 Einschätzung des Projektseminars

Da wir als Gruppe von vier Informationssystemtechnikern bereits Veranstaltungen wie GDI3 oder Praktikum C/C++ absolviert hatten, konnten wir uns von Anfang an im Code gut zurecht finden. Dies hatte auch den Vorteil, dass alle Mitglieder auf dem gleichen Wissensstand der Programmiersprache C waren. Probleme rund um die GPS-Sticks oder anderen Bugs im Compiler ließen uns jedoch oft an Problemen verzweifeln, die uns vom konstruktiven Programmieren abhielten. Auch das Eclipse-Eigene GIT lies oftmals Frustrationen aufkommen, sodass wir kurzerhand auf Tortoise GIT umgestiegen sind. Die Kommunikation zwischen den einzelnen Gruppenmitgliedern war immer gut, sodass Entscheidungen stets gemeinschaftlich getroffen werden konnten.

Insgesamt sind wir sehr zufrieden mit unserem Ergebnis, da wir nicht nur alle grundlegenden Aufgaben erfolgreich implementiert haben, sondern darüber hinaus auch alle gestellten Bonusaufgaben und Experimente zur Car2Car Implementierung erfolgreich abschließen konnten. Wir konnten neue Erkenntnisse über hardwarenahe Implementierung erlangen und lernten, wie das Erarbeiten von Aufgaben in einem großen Projekt verläuft.