Abschlussbericht Projektseminar Echtzeitsysteme Team Chipmunks WS 2014/15

Ausarbeitung zum Projektseminar Echtzeitsysteme im WS 2014/15
Proseminar eingereicht von
Team Chipmunks - Tobias Schultes, Steffen Kreis, Artur Fast und Markus Grau
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. Andy Schürr
Betreuer: Géza Kulcsár, MSc.

Inhaltsverzeichnis

1 Implementierung

1.1 Struktur

1.1.1 Kommunikation

Für die Kommunikation werden zwei Schnittstellen benötigt. Zum einen ist eine bidirektionale Verbindung zu einem einzelnen Fahrzeug sinnvoll, um den RC-Modus zu implementieren. Zum anderen müssen die Autos auch untereinander kommunizieren können. Für die Car-2-Car Aufgabenstellung müssen Informationen zwischen allen beteiligten Fahrzeugen ausgetauscht werden. Hierfür bietet sich die Kommunikation mit Broadcasts an.

1.1.2 Fahrzeugsteuerung

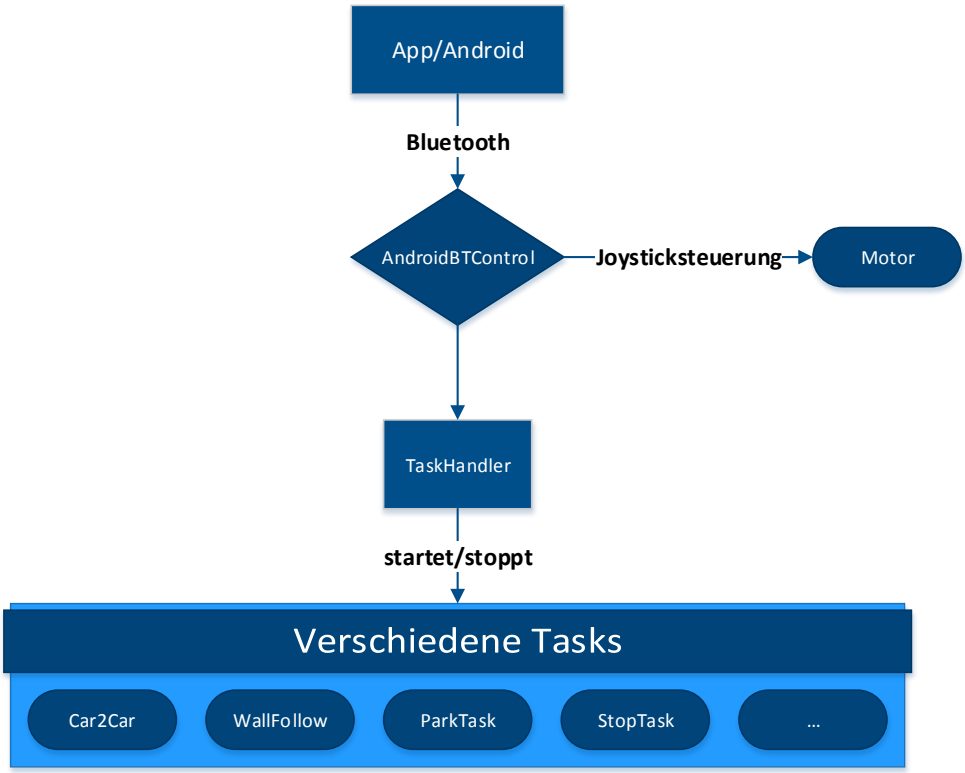

Abbildung 1.1.1.: Struktur Fahrzeugsteuerung

1.2 Fahrtasks

Die Fahrzeugsteuerung wird über die Android App Chipmote realisiert. Diese kommuniziert über Bluetooth mit der Schnittstelle „AndroidBTControl“des Autos. Einfache Steuerbefehle wie Lenkeinschlag und Geschwindigkeit werden direkt über die API an die Motoren weitergeleitet. Komplexere Fahrmanöver werden durch Tasks abgebildet. Diese werden vom Taskhandler als zentrale Stelle verwaltet. Beim Start des Autos werden hier alle Tasks initialisiert und Referenzen darauf gespeichert. Nach außen bietet er ein Interface, mit dem jeder einzelne Task gestartet und gestoppt werden kann. Nötige Überprüfungen, ob der Task bereits gestartet oder gestoppt ist, werden dabei vom Taskhandler übernommen.

1.2.1 Wallfollow & Kurven

PID-Regler

Um einen konstanten Abstand zur Wand zu halten muss ein passender Regler implementiert werden. In der Regelungstechnik sind PID-Regler üblich, die aus drei verschiedenen Anteilen bestehen: Abweichung vom Sollwert (P-Anteil), Summe der letzten Sollwertabweichungen (I-Anteil) und Änderung der Abweichung (D-Anteil). Diese Anteile werden jeweils mit einer Konstanten K_p, K_i, K_d multipliziert und ergeben addiert die Stellgröße.

Regler im Kontext Wallfollow

In der gegebenen Aufgabenstellung wird der Regler dazu verwendet, mit einem bestimmten Abstand (Sollwert) entlang einer Wand zu fahren. Dabei berechnet er aus den Sensordaten den passenden Einschlagswinkel (Regelgröße) für das Auto.
Würde man einen Regler verwenden, der nur aus einem P-Anteil besteht, so würde das Auto wellenförmig um den Sollabstand pendeln. Solange der Abstand zur Wand größer als der Sollwert ist, lenkt das Auto in Richtung Wand, bis der gewünschte Abstand erreicht ist. Allerdings steht das Auto beim Erreichen der Solllinie nicht parallel zur Wand, sondern überschreitet diese, woraufhin der Regler in die entgegengesetzte Richtung lenkt. Dadurch ergibt sich eine Fahrkurve wie in Abbildung 1.2.1). Um den Schwingungen entgegen zu wirken, kann ein D-Anteil eingesetzt werden. Dieser bewirkt, dass das Auto schon vor Erreichen der Solllinie gegenlenkt und diese im Idealfall nicht überschreitet. Mit einem I-Anteil könnte stationäre Genauigkeit erreicht werden und etwa Abweichungen der Servomotoren in Grundstellung ausgeglichen werden. Fehlerhafte Messwerte und Kurven verfälschen den I-Anteil und somit den Einschlagwinkel über längere Zeit. Dies war neben den guten Ergebnissen mit P- und D-Anteil ausschlaggebend dafür, den I-Anteil nicht zu verwenden.

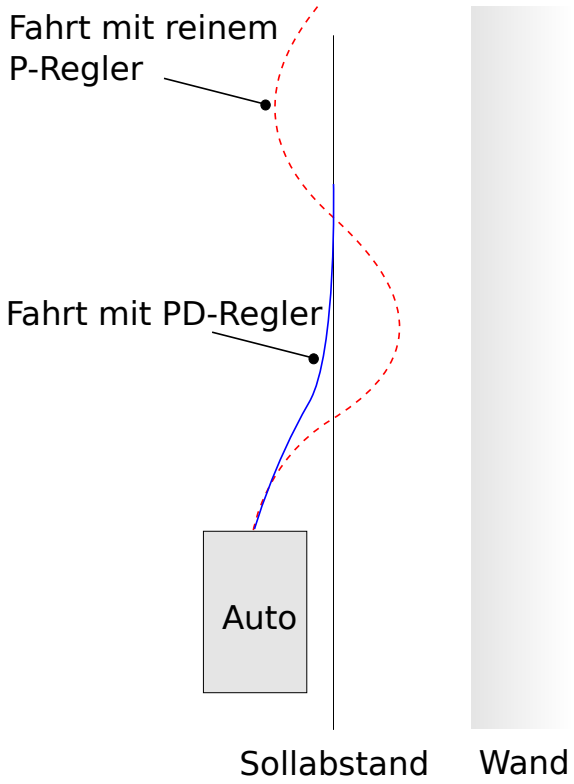

Abbildung 1.2.1.: Abstandshalter mit P- und PD-Regler

Bestimmung der Werte

Die Parameter für den PD-Regler wurden empirisch bestimmt. Dafür wurde zuerst K_p gewählt, sodass das Auto in Schwingungen um die Solllinie gefahren ist, ohne an die Wände anzustoßen. Daraufhin wurde K_d so lange erhöht, bis keine Oszillationen mehr zu sehen waren.

Rechtskurven

Neben dem Abfahren von geraden Wänden umfasste die Aufgabenstellung auch Kurven. Rechtskurven können mit dem PD-Regler gefahren werden. Erreicht das Auto eine Situation in der auf der rechten Seite keine Wand mehr erkannt wird, so erhöht sich die Abweichung vom Sollabstand und das Auto schlägt nach rechts ein. Da bei Kurven der D-Anteil des Reglers oft unerwünscht hoch wird, wurde eine Begrenzung auf einen Maximalwert vorgenommen.

Linkskurven

Wenn vor dem Auto eine Wand und links vom Auto eine freie Strecke detektiert wird (Abstand > Schwellwert), fährt das Auto eine Linkskurve. Diese wird auch mit einem PD-Regler realisiert, der sich an den Sensorwerten der linken Seite orientiert. Dadurch, dass auf der linken Seite zu Beginn der Kurve ein hoher Abstand gemessen wird, wird auch ein hoher Lenkeinschlag gewählt. Während der Linkskurve wird auch der Abstand

zur rechten Wand überwacht. Sobald dieser mehrere Male hintereinander abnimmt, wird die Linkskurve als beendet angesehen und wieder der Wallfollow-Task aktiviert. Da die Linkskurve auch durch fehlerhafte Messwerte und Messungen in Türrahmen aktiviert werden kann, wird ab der Aktivierung mehrere Zyklen lang überprüft, ob die Voraussetzungen für eine Linkskurve immer noch gegeben sind. Wenn dies nicht der Fall ist, wird das Abfahren der vermeintlichen Kurve abgebrochen.

1.2.2 Adaptive Geschwindigkeitsanpassung

Im realen Straßenverkehr wird im Normalfall auf geraden Strecken eine hohe Geschwindigkeit gefahren, wohingegen vor und in Kurven das Tempo verringert wird. Diese Vorgehensweise wurde in der Implementierung auf das automatisierte Fahren übertragen. Es wird eine hohe Geschwindigkeit gefahren, solange vor dem Auto kein Hindernis erkannt wird und der Einschlagwinkel (bedingt durch den PD-Regler) nur gering ist. Sobald mit dem Ultraschallsensor an der Front eine Wand detektiert wird, wird das Tempo gedrosselt, da ein Ausweichmanöver in Form einer Kurve gefahren werden muss. Auch während der Kurve bleibt das Tempo gedrosselt, solange der Einschlagwinkel größer als ein konfigurierbarer Wert ist. Ein noch zu lösendes Problem ist, dass fehlerhafte Messwerte dazu führen, dass auf geraden Strecken gelegentlich das Tempo gedrosselt wird.

1.2.3 Einparken

Nachdem der Parktask aufgerufen wurde, stellt das Auto mittels PD-Regler einen bestimmten Abstand zur Wand her. Wand und parkende Fahrzeuge werden mithilfe des Ultraschallsensors unterschieden. Ist ein Objekt näher als 10 cm an dem Auto, wird es als parkendes Fahrzeug interpretiert und der PD-Regler wird deaktiviert. Ist ein Objekt mehr als 15 cm entfernt wird es als Parklücke von ausreichender Breite interpretiert und der PD-Regler wieder aktiviert. Beim Entlangfahren an der Wand wird ein Distanzzähler benutzt um die Parklücke auszumessen. Erkennt das Auto eine Parklücke von ausreichender Länge, wird eine fest implementierte Einparksequenz ausgeführt.

1.2.4 Aktive Bremsung & Detektion Untergrund

Die Autos haben keine Bremsen, stattdessen schaltet man zum Stoppen des Fahrzeugs den Motor aus. Durch die Trägheit rollt das Auto noch etwas weiter, was in der Regel aber unerwünscht ist. Um dem Rollen entgegen zu wirken, wurde eine aktive Bremsung implementiert. Der Motor wird kurzzeitig entgegen seiner letzten Drehrichtung gesteuert, um schneller anzuhalten. Durch den unterschiedlichen Grip auf dem Teppichboden im Seminarraum im Vergleich zum PVC-Boden im Flur muss hier auch entsprechend unterschiedlich gebremst werden, um den gleichen Effekt zu erhalten. Ohne Fallunterscheidung würde das Auto auf dem Teppichboden ein Stück in die entgegengesetzte Richtung fahren, bzw. auf dem PVC-Boden zu weit ausrollen. Der Untergrund lässt sich

mit den Helligkeitssensoren gut erkennen, sodass diese zur Fallunterscheidung genutzt wurden.

1.2.5 Speedmode

Um eine schnelle Zeit auf dem Rundkurs zu erreichen, wurden neben der Erhöhung der Grundgeschwindigkeit weitere Anpassungen vorgenommen. Zum einen wurden die Kurven statt mit dem PD-Regler mit einem statischen Einschlagwinkel gefahren, wenn der seitliche Wandabstand einen gewissen Wert überstieg. Außerdem wurde die Adaptive Geschwindigkeitsanpassung auf Schnelligkeit statt Sicherheit ausgelegt. Die Geschwindigkeit wurde nur während Kurven gedrosselt und wenn in kurzem Abstand vor dem Auto ein Hindernis erkannt wurde.

1.3 RC-Modus

Das Fahrzeug stellt zwei drahtlose Kommunikationsschnittstellen zur Verfügung, Bluetooth oder eine Funkverbindung über das Amber-Modul. Für den RC-Modus wurde die Bluetooth-Verbindung gewählt. Zum einen ist diese verbindungsorientiert, was bei der Implementierung den Vorteil bringt, dass sich nicht um verlorene Pakete gekümmert werden muss. Zum anderen brachte Bluetooth den Vorteil, dass Smartphones als bereits vorhandene Hardware verwendet werden konnten, um die Fernsteuerung umzusetzen.

1.3.1 Struktur Chipmote

Bei der Implementierung der Android App Chipmote wurde darauf geachtet das MVC-Prinzip (Model-View-Control) nicht zu verletzen. Die View besteht aus Fragments für die verschiedenen Funktionen der App. Alle Eingaben in der View werden direkt an die Control-Klassen weitergeleitet und dort verarbeitet.

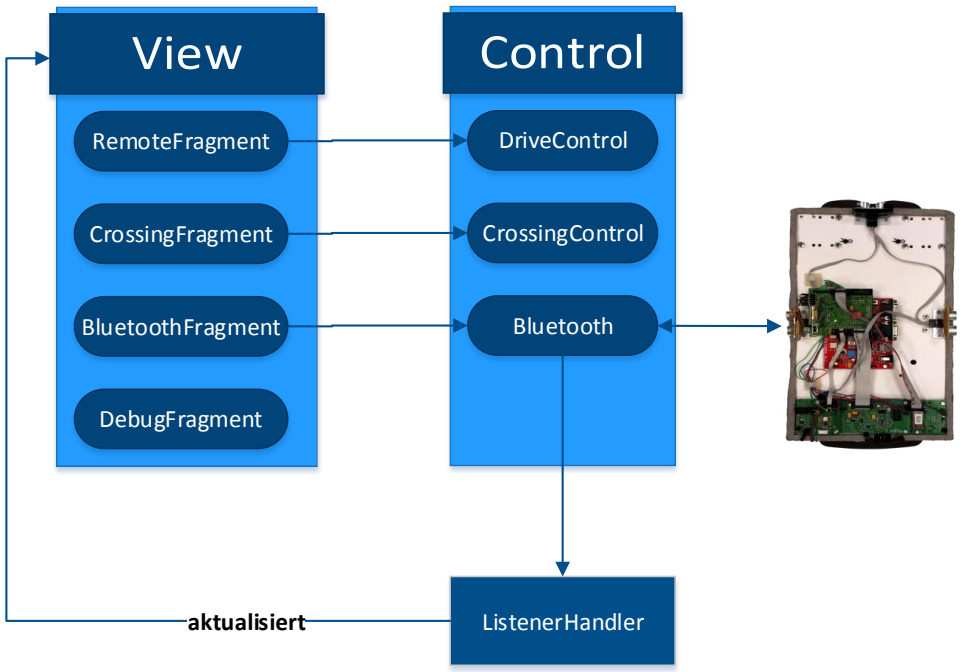

Abbildung 1.3.1.: Struktur Chipmote

1.3.2 Verbindungsmanagement

Der Aufbau der Bluetooth-Verbindung erfolgt durch den BluetoothConnectionTask und wird in der GUI ausgelöst. Nach erfolgter Verbindung werden zwei Threads gestartet. Der BluetoothDataSender zum Senden von Daten und der BlueoothReceiverthread zum Empfangen von Daten. Es werden verschiedene Nachrichtentypen verwendet. Ankommende Bluetoothnachrichten werden von dem BluetoothMessageHandler zwischengespeichert. Über den ListenerHandler werden die verschiedenen Listener informiert. Sensor- und Statusdaten werden direkt an den ListenerHandler und damit an die einzelnen Listener weitergereicht.
Der Verlust der Bluetoothverbindung führt zu einer Exception beim Lesen des Bluetoothstreams. Diese Exception wird aufgefangen, der aktuelle Verbindungsstatus wird angepasst und laufende Bluetooth-Threads beendet. Zudem wird eine Routine gestartet um dem Benutzer das einfache Wiederverbinden zu ermöglichen.

1.3.3 Fahrzeugsteuerung

Die Steuerung des Fahrzeugs kann grob in zwei Bereiche eingeteilt werden. Das manuelle Fahren und die Aktivierung von Fahrtasks auf dem Fahrzeug.

Manuelles Fahren

Die manuelle Steuerung erfolgt mit der Android-App auf dem Smartphone. Hierfür besitzt das Layout des RemoteFragment eine JoystickView, welche einen einfachen Joystick darstellt. Änderungen der Joystickposition, welche Geschwindigkeit und Einlenkwinkel definiert, werden von der Klasse DriveControl über die Bluetoothschnittstelle an das Fahrzeug weitergeleitet. Die gesamte Behandlung der Eingaben erfolgt auf dem Smartphone, es werden nur die fertig berechneten Werte für Geschwindigkeit und Lenkung an das Fahrzeug übertragen. Die Tempomatfunktion genannt Cruise Control ist eine Modifikation der manuellen Joysticksteuerung. Bei Aktivierung des Tempomats wird die aktuelle Geschwindigkeitseingabe des Joysticks gespeichert. Solange der Tempomat aktiv ist, wird nun die reale Joystickgeschwindigkeit durch die gespeicherte überschrieben. Um eine Notbremsung zu ermöglichen wird der Tempomat gestoppt, wenn über den Joystick eine negative Geschwindigkeit eingegeben wird. Das Beschleunigen bei aktivem Tempomat wurde ähnlich gelöst. Sollte die positive Geschwindigkeit die gespeicherte Tempomatgeschwindigkeit übersteigen, wird der aktuelle Joystickwert verwendet und das Auto beschleunigt. Der Tempomat wird in diesem Fall nicht deaktiviert, sondern hält nach dem Beschleunigen wieder die eingestellte Geschwindigkeit. Die Invertierung der Steuerung, erfolgt ebenfalls lokal. Hierzu werden die Joystickeingaben invertiert, wodurch bei der restlichen Eingabebehandlung keine Änderungen vorgenommen werden müssen.

Aktivierung von Fahrtasks

Im Gegensatz zu der manuellen Steuerung findet bei der Aktivierung von Fahrtasks, welche bereits in 1.2 beschrieben wurden, nur die Auslösung in der App statt. Startet ein Knopfdruck einen Fahrtask, wird ebenfalls über die DriveControl die Bluetooth-Schnittstelle angesprochen. Für die verschiedenen Tasks werden dann zwei Bytes übertragen. Das erste Byte ist die Id des Fahrtasks und das zweite Byte definiert den Zustand. Auf dem Fahrzeug kümmert sich dann der TaskHandler um die weitere Behandlung des Fahrtasks.

1.4 Kreuzungssituation

Bei der Kreuzungssituation kommunizieren mehrere Autos an einer Kreuzung, um zu ermitteln, welches von Ihnen Vorfahrt hat. Die Autos bekommen über die Android-App einzeln mitgeteilt, aus welcher Richtung sie kommen und in welche Richtung sie fahren sollen. Sie starten dann umgehend den Funkverkehr mit anderen Autos und fahren bis zur Haltelinie an der Kreuzung, um dort die weitere Vorgehensweise zu bestimmen. Sobald das erste Auto die Kreuzung passiert hat, informiert es alle anderen und fährt per Wallfollow die Wand entlang. Die verbleibenden Autos nehmen das passierte Auto aus der Kreuzungsbetrachtung heraus und beurteilen die Kreuzungssituation wieder von vorne.

1.4.1 Car2Car-Kommunikation

Amber

Die Kommunikation zwischen den Autos wird über das eingebaute Funkmodul Amber AMB8420 abgehandelt. Dies ermöglicht sehr einfache Broadcasts im Vergleich zu Bluetooth. Nachrichten werden im Intervall von 500 ms gesendet, um den sensiblen Funkverkehr nicht zu gefährden.

Protokoll

Broadcast-Nachrichten enthalten Auto-ID, Herkunft und Ziel bezogen auf die Kreuzung sowie den aktuellen Status. Der Status beschreibt, in welchem Zustand bzgl. der Kreuzung das Auto sich befindet. Hier wird unterschieden, ob das Auto noch auf die Kreuzung zu fährt, sie erreicht hat und an der Haltelinie wartet, die Kreuzung durchfährt, oder ob es die Kreuzung bereits passiert hat. Jedes Auto berechnet die eigene Vorfahrtssituation relativ zu jedem anderen Auto.

1.4.2 Rechts-vor-Links-Algorithmus

Für die Berechnung der Vorfahrt werden Rechts-vor-Links-Regeln benutzt. Bei benachbarten Autos darf immer das rechte von beiden zuerst fahren. Bei gegenüberliegenden Autos gilt zu beachten: Sollte eines geradeaus fahren wollen, während das andere links fahren möchte, hat Geradeaus Vorrang. Alle anderen Situationen können kollisionsfrei gleichzeitig bewältigt werden.

1.4.3 Kreuzung abfahren

Nachdem das Auto ermittelt hat, dass es in die Kreuzung fahren darf, werden je nach Abbiegerichtung folgende Abläufe gestartet:

Geradeaus
Da innerhalb der Kreuzung keine Wände zur Orientierung vorhanden sind, wird der Servomotor auf Grundstellung gebracht und der Motor angeschaltet, bis mit den Liniensensoren das Ende der Kreuzung erkannt wird.

Linkskurve
Für Linkskurven wird im Normalfall der Linkskurvenmodus des Wallfollow-Tasks verwendet. Da die Demo-Kreuzung über verschieden große Einfahrten verfügt, mussten jedoch Anpassungen je nach Startpunkt implementiert werden, die das Auto zuerst gerade in die Kreuzung einfahren lassen, bevor sie zur Linkskurve einschlagen.

Rechtskurve
Die Rechtskurven werden im normalen Wallfollow-Task gefahren, der jedoch auf einen geringen Wandabstand von 30 cm eingestellt wird.

1.4.4 Extras & Herausforderungen

Extras

Zusätzlich zur Aufgabenstellung wurden folgende Extras umgesetzt:

Gleichzeitig Fahren
Anhand der berechneten Kreuzungssituation können die Autos feststellen, ob eine gleichzeitige Befahrung der Kreuzung kollissionsfrei möglich ist und diese entsprechend durchführen.

Minimaler Stopp
Die Autos müssen nicht den vorgegebenen Timeout von 3 Sekunden an der Kreuzung abwarten, sondern kommen durch frühzeitige Kommunikation mit einem sehr kurzen Stopp aus. Für Autos, die keine Vorfahrt gewähren müssen, bedeutet dies eine Wartezeit von lediglich 100 ms für die Berechnung. Theoretisch könnten die Autos auf diesen Stopp auch komplett verzichten, wenn die Berechnung während der Fahrt durchgeführt wird. Dies wurde jedoch noch nicht umgesetzt.

Mehr als zwei Autos
Im Code ist vorgesehen, dass die Kreuzungssituation auch mit mehr als zwei Autos aufgelöst werden kann. Praktisch konnte dies aufgrund von Kommunikationsproblemen und fehlerhafter Hardware leider nicht validiert werden.

Herausforderungen

Funkverkehr
Aufgrund von Problemen beim Funkverkehr wurden die Sendeintervalle auf 500 ms verlängert, um eine Überlastung auszuschließen. Es ist allerdings zu beachten, dass bei zu langen Intervallen entsprechend lange Vorlaufzeiten gewährleistet sein müssen, sodass jedes Autos von allen anderen registriert werden kann.

Kreuzung
Die Einfahrten der gegebenen Demo-Kreuzung haben sehr unterschiedliche Längen und Breiten, sodass das Fahrtmanöver bei Linkskurven je nach Einfahrt entsprechend angepasst werden musste. Beim Rechts- und Geradeausfahren war dies nicht nötig.

2 Anpassungen API

2.1 Bluetooth-Protokoll

Für die Bluetoothübertragung wurde als Grundlage die Übertragung aus dem gegebenen Framework benutzt. Diese verwendet für die Übertragung Bytestuffing, um Start- und Stopbytes im Datenbereich zu escapen. Da die Nachricht bereits die Datenlänge enthält, ist es nicht nötig, die reservierten Bytes zu escapen, so lange diese im Datenbereich liegen. Außerdem liefert die verwendete Implementierung des Bytestuffing fehlerhafte Ergebnisse, wenn zwei Start- oder Stopbytes nacheinander vorkommen. Diese würden als 3x Start-/Stopbyte erkannt werden. Deshalb wurde das Bytestuffing in den Methoden startSendingNextMessage()/putStuffedByte() in der Datei bluetooth.c entfernt.

2.1.1 IDs für Funknachrichten

Die Nachrichtentypen wurden entsprechend der erweiterten Funktionalität der Android-App ergänzt. Bei der ID für Debugnachrichten musste eine Änderung vorgenommen werden: In der ursprünglichen API wurde für Debugmessages und zur Ansteuerung der Autos über Android/Bluetooth der gleiche Wert zur Identifikation verwendet. Weil Debug-Messages nur über das Amber-Modul benutzt wurden, während die Ansteuerung durch Android per Bluetooth erfolgt, hat dies keine Probleme bereitet. Weil in der Android-App auch Debugnachrichten verarbeitet werden, wurde den Debugnachrichten generell eine neue ID vergeben. Dies musste entsprechend auch im GPSconfig-Projekt geändert werden.

2.1.2 Fehler Bluetoothübertragung

Beim Senden von Nachrichten über Bluetooth trat ein Fehler auf, wenn mehrere Nachrichten in kurzer Zeit anfielen. Vor dem Versenden einer Nachricht wurde im ursprünglichen Code überprüft, ob aktuell ein Sendevorgang möglich ist. Ist dies nicht der Fall, weil bereits ein anderer läuft, so wird die Nachricht in einen Buffer gelegt. Allerdings wurde nie versucht, die Nachrichten aus dem Buffer zu senden, weshalb dieser nach und nach voll lief. Selbst wenn der Buffer voll war und keine Nachricht mehr hinzugefügt werden konnte, wurde keine Nachricht aus dem Buffer gesendet. Deshalb wurden folgende zwei Anpassungen in bluetooth.c vorgenommen:

- in Methode Bluetooth_Send: startSendingNextMessage(); nicht nur wenn Einfügen in Buffer erfolgreich (returnpd == pdTRUE), sondern immer.
- in Methode UART7_ TxISR: Wenn Senden der letzten Nachricht erfolgt, wird startSendingNextMessage(); aufgerufen und sendet Nachrichten aus dem Buffer wenn dort welche vorhanden sind.

Diese Anpassungen sollten auch übernommen werden, um das Bluetoothmodul in der Praxis verwenden zu können.

2.2 Verringerung der Stackgrößen

Da jede größere Funktionalität in einen eigenen Task ausgelagert wurde, ist der Platzbedarf auf dem Stack relativ groß. Da die gegebenen Tasks aus dem Framework bereits einen großen Teil des Stacks belegten, ohne diesen für die Funktionalität zu benötigen, wurde er reduziert. Die Stackgröße für den Task BTLayer3_Receive konnte von 1400 auf 500 Bytes verkleinert werden. Dies reichte aus, um genügend Stack für alle Tasks zur Verfügung zu stellen. Ob die Stackgröße für andere Tasks aus dem Framework auch begrenzt werden kann, muss geprüft werden.

2.3 Detektion von Stacküberläufen

Eine große Fehlerquelle während des Entwicklungsprozesses waren Stacküberläufe. Es wurde recherchiert, dass das Betriebssystem eine Möglichkeit bietet, diese zu erkennen. Dafür wurde in der Datei tasks.c die Methode extern void vApplicationStackOverflowHook(xTaskHandle xTask, signed char *pcTaskName) eingefügt. In dieser wird die Geschwindigkeit auf 0 gesetzt, um Schäden am Auto zu vermeiden und eine Zahl auf dem Display des Autos ausgegeben. Dadurch kann man Stacküberläufe schnell und sicher erkennen. Zusätzlich wurde ein Check für MallocFailed hinzugefügt, der anzeigt, dass der Stack bereits komplett reserviert ist und deshalb ein neuer Task nicht angelegt werden kann. Dies wurde analog zum Stacküberlauf mit der Methode extern void vApplicationMallocFailedHook(void) umgesetzt.

2.4 Lenk-Offset

Die Autos fahren nicht geradeaus, wenn der Servomotor in Grundstellung gebracht wird. Deshalb wurde für jedes Auto empirisch der entsprechende Wert ermittelt, mit dem es geradeaus fährt. Dieser wurde dann als Offset verwendet und beim Setzen der Servo-Werte im Code nach der Car-ID ausgewählt und jeweils aufaddiert. Durch die intensive Nutzung der Autos haben sich die Werte im Laufe des Semesters jedoch teilweise geändert, weshalb eine neue Bestimmung nötig wäre. Die Werte können in der Methode Drive_Init() in drive.c geändert werden.

3 Ausblick

Im Folgenden soll eine Hilfe für zukünftige Gruppen gegeben werden, um ihnen die Arbeit mit der eingereichten Implementierung als Grundlage zu vereinfachen. Neben Möglichkeiten zur Anpassung und Erweiterung werden auch Verbesserungsvorschläge gegeben, die im Rahmen des Projektseminars noch nicht umgesetzt wurden.

3.1 Anpassungsmöglichkeiten

3.1.1 Neue Tasks

Soll ein neuer Task eingefügt werden, der mit Chipmote angesteuert werden soll, müssen folgende Schritte durchgeführt werden:

1. Initialisierung des Tasks im Taskhandler
2. Implementierung von Methode zum Starten und Stoppen des Tasks im Taskhandler
3. Erweiterung der App um neuen Button + Senden von neuem ID-Wert bei Klick
4. Verarbeitung der ID in motorControl_bt() von AndroidBTControl.c durch Aufruf der Taskhandler-Methode

3.1.2 Änderung der Implementierung

Die einzelnen Aufgaben wurden durch unabhängige Tasks implementiert. Deshalb kann die Implementierung von einzelnen Aufgaben verändert werden, ohne andere Aufgabenteile berücksichtigen zu müssen.

3.2 Verbesserungsbedarf

3.2.1 Adaptive Geschwindigkeitsanpassung

Ein noch zu lösendes Problem ist, dass fehlerhafte Messwerte dazu führen, dass auf geraden Strecken gelegentlich das Tempo gedrosselt wird.

3.2.2 Stackgröße

Aktuell ist die Größe des reservierten Stacks für die Tasks noch nicht optimiert, einige der Tasks können auch mit einer geringeren Größe laufen. Ideale Werte müssen bei Bedarf durch weitere Tests ermittelt werden.

3.2.3 Car2Car

Um die Kreuzungssituation auch mit mehr als 2 Autos durchzuführen, sind noch weitere Tests nötig. Außerdem kann der kurze Stopp an der Kreuzung vermieden werden, indem die Berechnung der Vorfahrtsituation bereits während der Fahrt erfolgt. Dafür muss der geeignete Zeitpunkt gefunden werden, damit das Auto möglichst lange auf andere Nachrichten reagieren kann aber auch nicht erst anhält wenn es sich bereits auf der Kreuzung befindet.

A Anhang: Bedienungsanleitung Chipmote

A.1 Überblick

Die Fernsteuerungs-App Chipmote hat insgesamt vier verschiedene Ansichten. Diese können durch das Navigationsmenü gewechselt werden. Das Navigationsmenü wird durch Streichen von der linken Seite des Bildschirmrandes geöffnet. Alle Ansichten außer der Fernsteuerung sind im Portrait- und im Landscape-Modus verfügbar.

A.2 Verbindung

Die Verbindungsansicht in Abbildung A.2.1 bietet die Möglichkeit, sich mit einem Fahrzeug zu verbinden. Das Bluetooth-Pairing muss zuerst in den Android-Einstellungen im Bluetooth-Bereich erfolgen. Es werden nur Fahrzeuge des PSES Projektes zum Verbinden angezeigt. Gefiltert wird dies über den Namen, welcher „BT-Car“enthalten muss. Über den Kippknopf unten kann Bluetooth aktiviert und deaktiviert werden. Um sich mit einem Fahrzeug zu verbinden muss man auf das entsprechende Eintrag in der Liste klicken. Erneutes Klicken beendet die Verbindung. Ist man mit einem Fahrzeug verbunden und klickt auf ein anderes, so wird die aktuelle Verbindung beendet und das neue Fahrzeug verbunden. Der Verlust der Verbindung lässt einen Benutzerdialog erscheinen, der vorschlägt, sich wieder mit dem Fahrzeug zu verbinden.

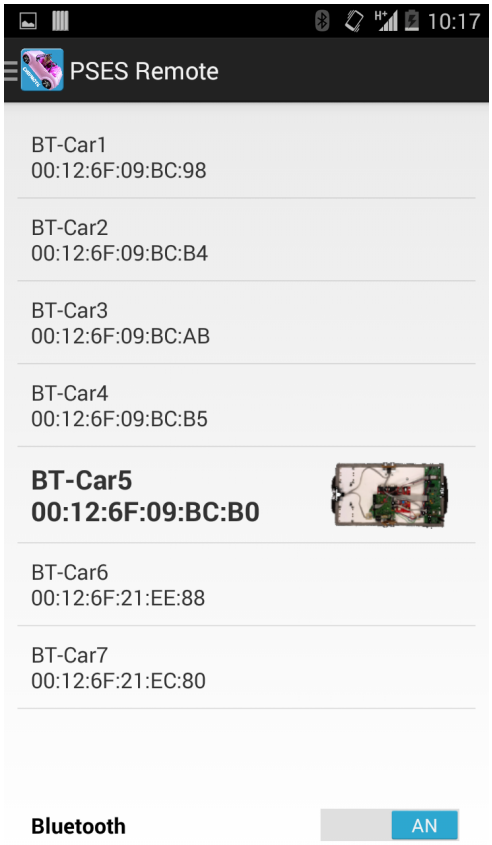

Abbildung A.2.1.: Verbindungsansicht

A.3 Fernsteuerung

Die Fernsteuerung besteht aus drei Komponenten. Auf der rechten Seite liegt ein Joystick mit dem das Auto manuell gesteuert werden kann. In der Mitte befindet sich eine Statusanzeige über das Fahrzeug. Es werden zum einen die Abstände der drei Utraschallsensoren, links, rechts und nach vorne in Zentimetern angezeigt. Zur Orientierung sind die Werte um das Bild des Fahrzeugs angeordnet. Zum anderen werden unterhalb des Bildes die aktuelle Akkuspannung und der Mittelwert der fünf Helligkeitssensoren angezeigt. Auf der linken Seite liegen mehrere Buttons für weitere Funktionen.

Invert Control

Invertiert die Steuerung des Joysticks. Die Funktionen Wall Follow und Cruise Control werden in diesem Modus deaktiviert.

Wall Follow

lässt das Fahrzeug automatisch der rechten Wand folgen und hält den aktuellen Abstand zu dieser.

Cruise Control

Tempomat für die manuelle Steuerung. Bei Aktivierung wird die aktuelle Geschwindigkeit gehalten und es kann weiterhin gelenkt werden. Während der Tempomat aktiviert ist, kann weiterhin über den Joystick beschleunigt werden. Der Tempomat bleibt aktiv und hält die ursprüngliche Geschwindigkeit bei Loslassen des Joysticks nach der Beschleunigung wieder. Um ungewollte Kollisionen zu vermeiden, führt ein starkes Abbremsen zur Deaktivierung des Tempomats.

Park

Startet den Parkvorgang, wobei das Fahrzeug weiterfährt, bis eine ausreichend große Lücke gefunden wird und parkt anschließend automatisch ein.

Stop

Button stoppt das Fahrzeug sofort mit einer aktiven Bremsung, wobei alle aktivierten Funktionen deaktiviert werden.

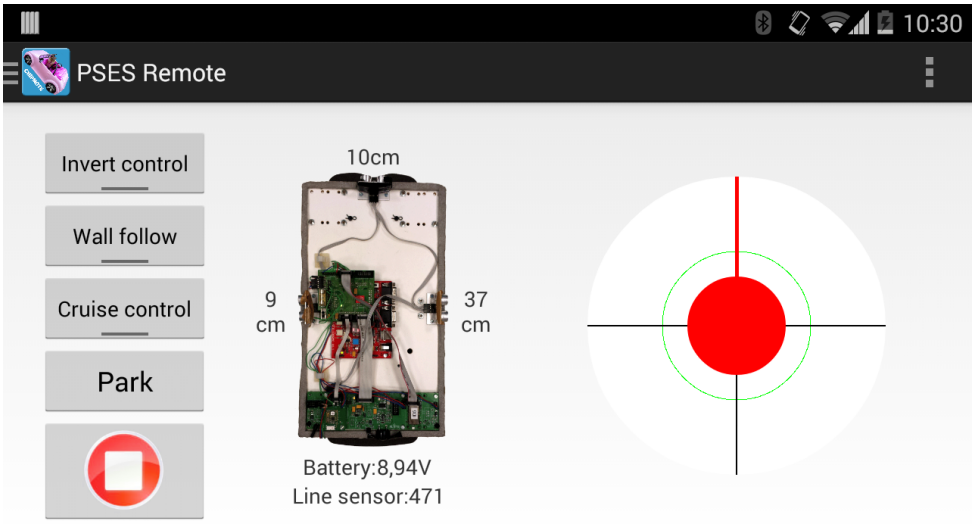

Abbildung A.3.1.: Fernsteuerung

A.4 Debugging

In der Debugging-Ansicht (Abbildung A.4.1) befindet sich im oberen Bereich die aus der Fernsteuerung bekannte Statusanzeige in verkleinerter Form. Direkt darunter gibt es ein Textfeld mit dem die Liste der Nachrichten gefiltert werden kann. Jede Nachricht hat einen Zeitstempel und einen Nachrichtentext. Über den Android-Menüknopf kann ein Auswahlmenü geöffnet werden. In diesem Menü können die Nachrichten gelöscht, das automatische Scrollen aktiviert und die gewünschten Nachrichten zur Anzeige ausgewählt werden.

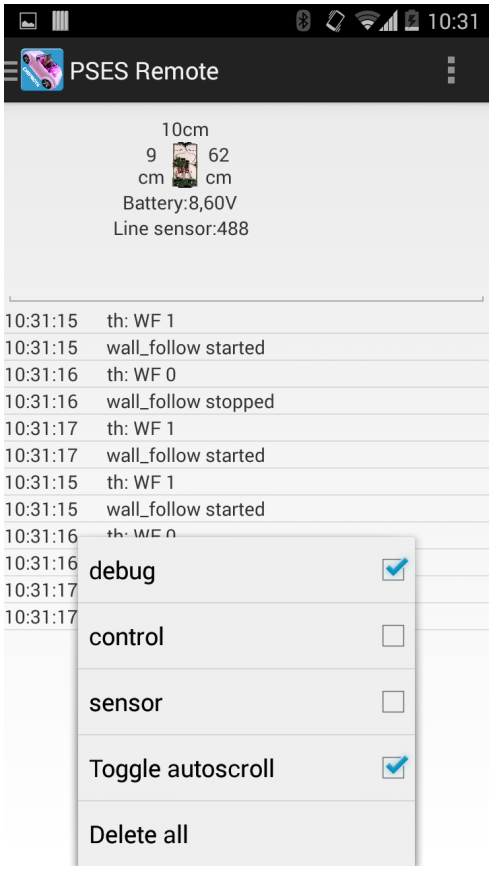

Abbildung A.4.1.: Debugging-Ansicht

A.5 Kreuzung

Die Kreuzungs-Ansicht dient der Konfiguration der Kreuzungssituation. Wie in Abbildung A.5.1 kann im oberen Bereich die gewünschte Fahrtrichtung bei Erreichen einer Kreuzung angegeben werden. Im gezeigten Bild soll nach Rechts abgebogen werden. Im unteren Bereich kann die Startposition des Fahrzeugs in der Kreuzung angegeben werden. Es wird von einer normalen Kreuzung mit vier möglichen Einfahrten ausgegangen und von oben betrachtet gegen den Uhrzeigersinn gezählt. Beim Drücken des Startbuttons fährt das Auto los. Zeitgleich beginnt die Car2Car-Kommunikation mit den anderen Fahrzeugen. Nach Erreichen der Kreuzung wird diese nach der Rechts-vor-Links-Regel abgefahren.

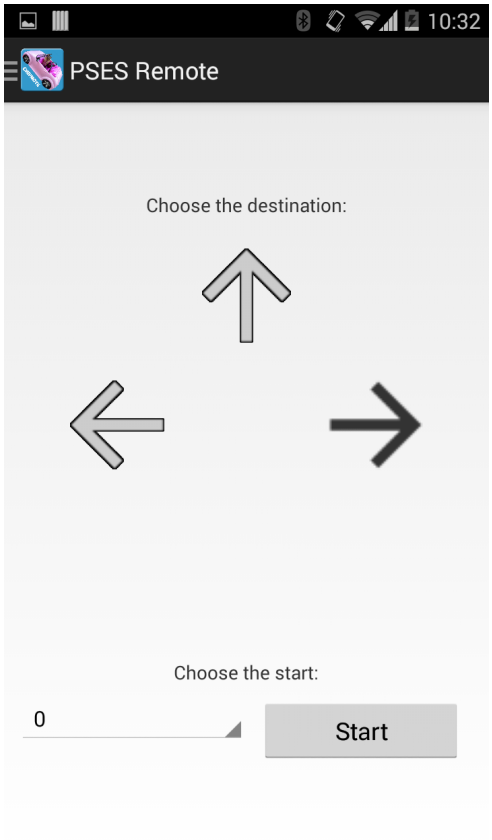

Abbildung A.5.1.: Kreuzungs-Ansicht