Humboldt-Universität zu Berlin
Institut für Informatik
Rechnerorganisation und Kommunikation

Technische Informatik II

Leitung: Prof. Dr. Miroslaw Malek

SwARM

Swarming Autonomous Robotic Mobiles

Peter Hartig, phartig@informatik.hu-berlin.de

Jörg Pohle, pohle@informatik.hu-berlin.de

Projektbetreuung: Dr. Sommer

---

4. Juli 2006

Inhaltsverzeichnis

Kapitel 1

Problemstellung

Wir betrachten das Problem der Selbstorganisation autonomer mobiler Systeme. Konkret suchen wir eine Möglichkeit, wie sich autonome mobile Systeme zu einem Netzwerk derart zusammenschliessen bzw. ein bestehendes erweitern, dass sie um einen gegebenen Mittelpunkt eine maximale Ausdehnung bzgl. der Empfangsstärke drahtloser Netzwerke, gleichmäßig in alle Richtungen, erreichen.

1.1 Problemanalyse

In vielen Situationen besteht ein Bedarf nach einer größtmöglichen räumlichen Abdeckung durch ein Netzwerk. Während es bei der Planung von Netzwerken innerhalb von Gebäuden sinnvoll sein kann, vor allem aufgrund der höheren Datenrate kabelgebundene Netzwerktechnik zu verwenden, eignen sich drahtlose Netzwerke hervorragend für den Einsatz außerhalb von Gebäuden. Oftmals ist es jedoch notwendig, die einzelnen Access Points manuell in eine Position zu bringen, um eine größtmögliche und gleichzeitig gleichmäßige Netzabdeckung zur Verfügung zu stellen. Drei unterschiedliche Beispiele sollen dies illustrieren:

1. Auf einer großen Außenveranstaltung will der Veranstalter den Besucherinnen und Besuchern einen drahtlosen Internetzugang zur Verfügung stellen. Bisher ist es dazu notwendig, die Access Points von Hand auf dem Gelände zu verteilen.

2. Ein Sensor-Netzwerk soll zur Überwachung einer großen landwirtschaftlichen Anbaufläche aufgebaut werden und dort Daten über das Mikroklima sammeln. Auch hier müssen bisher die einzelnen Module von Hand positioniert werden.

3. Eine Verbindung mit Erschütterungssensoren wäre denkbar, wie sie am Lehrstuhl für Signalverarbeitung und Mustererkennung entwickelt wurden. Mit diesen Sensoren kann an der Erschütterung des Untergrunds die Art des Fahrzeugs bestimmt werden. Damit könnte eine demilitarisierte Zone zwischen zwei Bürgerkriegsparteien auf die Bewegung von gepanzerten Fahrzeugen überwacht werden.

In allen Fällen wäre es viel einfacher, wenn die Geräte in der Lage wären, sich selbständig auf dem Gelände zu verteilen und dabei mit möglichst wenigen

Geräten eine maximale Fläche abzudecken.

1.2 Festlegung der Basisfunktionalität

1.2.1 Autonomie

Nachdem den Geräten der Mittelpunkt ihres Einsatzgebietes in Form von geographischen Koordinaten übergeben wurden, sollen alle weiteren Schritte zum Aufbau des Netzes autonom erfolgen, also ohne menschliche Eingriffe.

1.2.2 Mobilität

Der Aufbau des Netzwerkes erfordert ein mobiles System in doppelter Hinsicht. Einerseits sollen die Geräte ihren Einsatzort selbsändig erreichen können, in unserem Fall heißt das, dass sie einen fahrbaren Chassis besitzen. Andererseits müssen sie in der Lage sein, für die Dauer des Einsatzes die notwendige Energie selbst zu gewinnen. Obwohl der Punkt der Energiegewinnung hier nicht Teil der Ausarbeitung sein soll, nehmen wir an, alle Geräte seien mit einem ausreichend großen Solarmodul ausgestattet.

1.2.3 Abdeckung

Die Standorte der einzelnen Geräte sollen jeweils so weit von einander entfernt sein, dass jedes Gerät nur diejenigen Nachbarn per WLAN-Funk erreichen kann, die in direkter Umgebung positioniert sind. Nur dadurch, dass die Abstände zwischen den einzelnen Geräten in Bezug auf die direkte funktechnische Erreichbarkeit maximal sind, lässt sich mit einer minimalen Anzahl an Geräten eine größtmögliche Fläche abdecken.

1.3 Systemanforderungen

1.3.1 Sensoren

WLAN-Modul

Das WLAN-Modul stellt für die Abstimmung und Orientierung der Roboter folgende Informationen zur Verfügung:

Kontakt zu benachbarten Agenten An dem WLAN-Modul sind für uns nur zwei Informationen interessant: die Dämpfung angegeben in Dezibel und die Richtung zu einem anderen Roboter als Winkel zur Fahrtrichtung in 1/10 Grad.

Entfernung zu benachbarten Agenten Wir gehen dabei davon aus, dass die Dämpfung an allen Orten 40 Dezibel pro 1m Entfernung in der Luft beträgt. Auf eine Entfernung von 130m ergibt sich eine Dämpfung von 85 Dezibel.[1] Aus dieser Eigenschaft und der Sendeleistung von 15 Dezibel des WLAN-Moduls ergibt sich -70 Dezibel als Richtwert für eine „optimale“ Entfernung der Roboter

[1] http://www.steckerprofi.com/wl-yagi2.htm

voneinander. Wir gehen dabei davon aus, dass damit nur die direkten Nachbarn eines bestimmten Gerätes erreicht werden können, nicht jedoch die weiter entfernten.

Winkel der Fahrtrichtung Die Richtung zu einem anderen Roboter gibt uns die Möglichkeit den Roboter in ein bestehendes Netzwerk einzuordnen und die richtige Stelle für die Einordnung zu finden.

Weitere Eigenschaften und Funktionen des WLAN sind für uns nicht von Interesse.

GPS-Modul

Für die Orientierung in der gewählten Umgebung werden die Daten eines GPS-Moduls benutzt. Für unsere Ansprüche benötigen wir den GPRMC-Datensatz (empfohlener Minimumdatensatz) und den GPVTG-Datensatz[2].

Der GPRMC-Datensatz liefert bspw. folgenden Datensatz:

$GPRMC,191410,A,4735.5634,N,00739.3538,E,0.0,0.0,181102,0.4,E,A*19

Aus diesem Datensatz benötigen wir die Informationen über die nördliche Breite (Angabe vor dem N) und die östliche Länge (Angabe vor dem E).

Beispiel für einen GPVGT-Datensatz

$GPVTG,0.0,T,359.6,M,0.0,N,0.0,K*47

Für uns interessant ist nur die erste Stelle hinter der Bezeichnung, die den wahren Kurs angibt. Wir gehen davon aus, daß es keine magnetische Deklination auszugleichen gibt, d.h. der magnetische und der wahre Kurs gleich sind.

Die GPS-Module liefern dabei auf Anfrage jedoch nicht die Datensätze selbst, sondern nur die von uns ausgewählten. Sie stellen dabei die Daten auch nicht, wie im o. g. Standard beschrieben, über eine serielle Schnittstelle bereit, sondern legen nur das angefragte Datenwort auf den zentralen Bus des Systems.

Breitengrad Der Breitengrad wird dargestellt in Grad, Gradminuten und Gradsekunden. Weiterhin ist die Unterscheidung zwischen nördlicher und südlicher Breite notwendig.

Längengrad Der Längengrad wird dargestellt in Grad, Gradminuten und Gradsekunden. Weiterhin ist die Unterscheidung zwischen östlicher und westlicher Länge notwendig.

Kurs Als Kurs wird der wahre Kurs aus dem GPVGT-Datensatz benutzt.

1.3.2 Aktuatoren

Der Prozessor steuert 2 Aktuatoren an. Der erste ist die Antriebssteuerung des Gerätes, mit dem zweiten kann das Gerät auf der Stelle nach links und nach rechts gedreht werden.

[2] http://www.kowoma.de/gps/zusatzerklaerungen/NMEA.htm

Antriebssteuerung

Die Antriebssteuerung (AS) besitzt nur zwei Zustände – das System fährt oder stoppt. Da die Fahrgeschwindigkeit nicht relevant ist und das System mit geriner Energie auskommen muss, benötigen wir nur eine einzige Geschwindigkeitsstufe (etwa 2 km/h). Außerdem ist das Gerät nur in der Lage, geradeaus zu fahren – einen Rückwärtsfahrmodus gibt es nicht.

Lenkungssteuerung

Die Lenkungssteuerung (LS) besitzt drei Zutände. Im stehenden Zustand kann das System auf der Stelle nach links bzw. nach rechts gedreht werden. Dabei dreht sich das Gerät so lange, bis die Drehung gestoppt wird.

1.4 Grenzen und Beschränkungen

1. Wir nehmen an, dass der Untergrund eben ist und es keine Hindernisse gibt, die von einem Gerät umfahren werden müssen.

2. Ausreichende Energiegewinnung gilt als über den gesamten Lebenszeitraum des Systems vorausgesetzt.

3. Für die Dauer der Installation des Netzwerkes nehmen wir an, dass es keine Hardware-bedingten Ausfälle gibt, so dass Rekonfigurationen eines bestehenden Netzwerkes nicht notwendig sein werden.

4. GPS-Daten sind zu jedem Zeitpunkt vorhanden und vom System auswertbar.

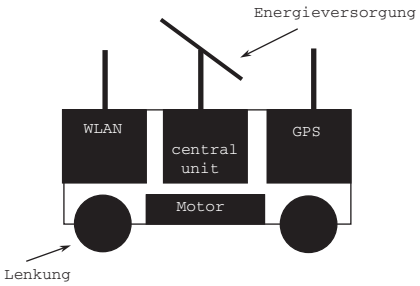
5. Alle WLAN-Module senden mit der gleichen Sendestärke und besitzen die gleiche Empfangsempfindlichkeit. Die Dämpfung der Signale ist an allen Orten gleich. Wir nehmen daher an, dass Signalstärke eindeutig als Maß für die Entfernung gebraucht werden kann.

6. Ein beliebiges Netzwerk liegt immer vollständig in einem Quadranten der Erdkugel (z. B. nördliche Breite und östliche Länge). Die Fläche, die das Netzwerk aufspannt, wird weder vom Äquator noch vom Großkreis, auf dem der Nullmeridian liegt, geschnitten. Sie überdeckt auch weder den Nord- noch den Südpol.

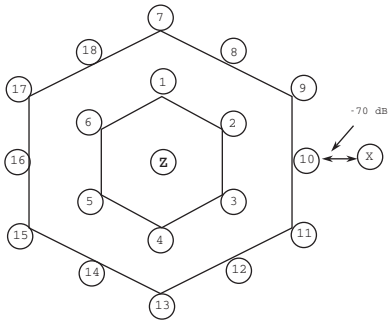
7. Alle Geräte treffen einzeln und immer von außen auf das Netzwerk, daher werden Probleme, die durch das Aufeinandertreffen von mehreren nach ihrer Position suchenden Geräten auftreten könnten, ignoriert. Ist beim Eintreffen eines Gerätes noch kein Netzwerk vorhanden, bildet dieses Gerät den zentralen Knoten des Netzwerkes.

8. Wir nehmen an, dass es keine magnetische Deklination gibt, so dass magnetischer und wahrer Kurs, wie sie vom GPS-Modul geliefert werden, gleich sind.

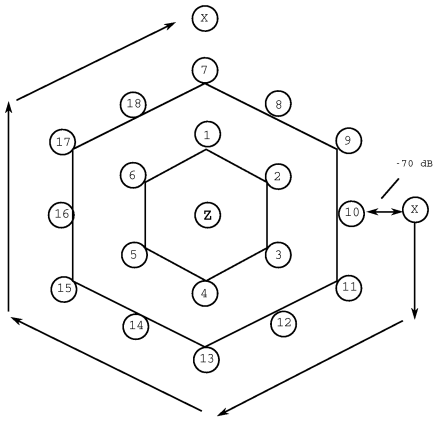
1.5 Schaubilder



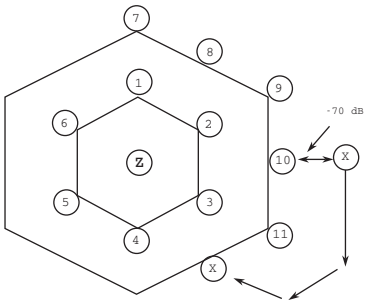
Die erste Abbildung zeigt den grundlegenden Aufbau des Gesamtsystems.



In diesem Bild kann man sehen, dass das Gerät so weit an das Netzwerk heranfährt, dass der Abstand genau -70 dB beträgt, also etwa 30m. Das Gerät trifft dabei entweder direkt von Osten (wie hier zu sehen) oder direkt von Westen (also spiegelbildlich) auf das Netzwerk.



Das Gerät umrundet das Netzwerk im Uhrzeigersinn, um eine freie Position zu finden. Wenn es keine freie Position in einer bestimmten Schale findet, dann fährt es zur nördlichsten Position und eröffnet damit eine neue Schale.



Wenn das Gerät eine frei Position im Netzwerk gefunden hat, nimmt es sie ein und tritt dem Netzwerk bei.

Kapitel 2

Architekturentwurf

2.1 Benutzerschnittstellen

In unserem Entwurf sind keine Benutzerschnittstellen vorgesehen. Alle notwendigen Basisdaten (Ziel-Signalstärke für die Entfernungsbestimmung, geographische Daten über den Mittelpunkt des aufzubauenden Netzwerkes) sind als Konstanten im Speicher vorhanden. Die Ergänzung des Systems um eine Benutzerschnittstelle für die Dateneingabe wird ausführlicher unter 4.6 diskutiert.

2.2 Sensoren

2.2.1 WLAN-Modul

Das WLAN-Modul wird mittels Signalen vom Prozessor gesteuert, kann vom Bus lesen und auf den Bus schreiben und die gestellten Anfragen verarbeiten.

Signale

`WLAN_in` Liegt dieses Signal an, so geht das WLAN-Modul in den Zustand, in dem es die auf dem Bus liegenden Daten ausliest.

`WLAN_out` Auf dieses Signal hin kann das WLAN-Modul Daten auf den Bus schreiben.

`WLAN_rdy` Dieses Signal wird von dem WLAN-Modul gesetzt, wenn es eine vollständige Antwort auf den Bus geschrieben hat, die an anderer Stelle weiterverarbeitet werden kann.

Befehle

Befehl	Antwort	Erläuterung
`0XX0`	$-850 - 150$	Signalpegel von Gerät `XX` in $\frac{1}{10}$ dB
`0XX1`	$0 - 3600$	Winkel von `XX` zur Fahrtrichtung in $\frac{1}{10}$ Grad
`1000`	keine	Modul soll sich als Zentralsystem konfigurieren
`1001`	keine	Modul soll sich als Teil des Netzwerkes konfigurieren

Das „XX“ in den Befehlen steht für die jeweils vier nächsten erreichbaren Geräte. Dabei bezeichnet „00“ das nächste und „11“ das entfernteste noch erreichbare Gerät. Es kann auch eine Situation geben, in der mehr als nur vier Geräte erreichbar sind. Nähert sich das Gerät allerdings von aussen einem bestehenden Netzwerk, sind nie mehr als vier Geräte gleichzeitig in Reichweite und nur in diesem Fall ist Information für das weitere Vorgehen relevant.

Mit dem Befehl „1000“ wird das WLAN-Modul in den Zustand versetzt, in dem es zum Mittelpunkt und damit ersten Bestandteil eines neuen Netzwerkes. Es hat dann die Möglichkeit bestimmte Dienste zur Verfügung zu stellen, wie Routing, DNS, DHCP o. ä., deren Implementation aber dem praktischen Einsatz vorbehalten bleibt.

Der Befehl „1001“ wird in dem Moment gegeben, in dem die Positionierung in einem bestehenden Netzwerk abgeschlossen, d. h. die richtige Position mit entsprechenden Abständen zu den anderen Geräten. Daraufhin integriert sich das WLAN-Modul in das bestehende WLAN und erweitert es um einen neu abgedeckten Bereich.

Auf den Befehl „0XX0“ kann es die Situation geben, dass kein anderes Gerät erreichbar ist. Für diesen Fall bedarf es eines Fehlercodes, der diese Situation eindeutig benennt. Erreicht das WLAN-Modul also kein anderes Gerät so legt es als Antwort ein 16-Bit-Wort auf den Bus, in dem alle Bits gesetzt sind. Es handelt sich also um die kleinste negative Zahl, die mit 16 Bit dargestellt werden kann. Da wir die Binärzahlen in der Darstellung des Zweierkomplements betrachten, ergibt sich als kleinste negative Zahl -2^15, also -32768. Legt also das WLAN-Modul diesen Wert als Daten auf den Bus, können wir die Antwort eindeutig interpretieren.

Ablauf

Um Daten vom WLAN-Modul zu erhalten, muss eine entsprechende Anfrage bzw. ein Befehl auf den Bus gelegt werden. Danach wird das Signal `WLAN_in` gesetzt. Das Modul liest daraufhin die Daten vom Bus und verarbeitet die jeweilige Anfrage. Danach wird das Signal `WLAN_out` gesetzt, das das WLAN-Modul veranlasst, seine Antwort auf den Bus zu schreiben. Wenn das WLAN-Modul seine Antwort vollständig auf den Bus geschrieben hat, setzt es das Signal `WLAN_rdy` und die Antwort kann vom Bus gelesen und weiter verarbeitet werden.

2.2.2 GPS-Modul

Das GPS-Modul wird durch Signale vom Prozessor gesteuert, kann vom Bus lesen und auf den Bus schreiben und die gestellten Anfragen verarbeiten.

Signale

`GPS_in` Das Signal bringt das GPS-Modul in den Zustand die auf dem Bus liegenden Daten zu lesen.

`GPS_out` Auf das Signal hin schreibt das GPS-Modul seine Antwort auf den Bus.

`GPS_rdy` Mit diesem Signal wird die Gültigkeit der Daten auf dem Bus angezeigt.

Befehle

Befehl	Antwort	Beschreibung
0000	$0 \vee 1$	nördl. oder südl. Breite
0001	$0 - 5400$	Breitenanteil in Bogenminuten
0010	$0 - 3600$	Breitenanteil in $\frac{1}{10}$ Bogensekunden
0100	$0 \vee 1$	westl. oder östl. Länge
0101	$0 - 5400$	Längenanteil in Bogenminuten
0110	$0 - 3600$	Längenanteil in $\frac{1}{10}$ Bogensekunden
1000	$0 - 3600$	eigener Kurs in $\frac{1}{10}$ Grad

Die Angaben zur nördlichen bzw. südlichen Breite und zur westlichen bzw. östlichen Länge werden in unserem System nicht ausgewertet, dienen jedoch zur späteren Erweiterbarkeit (siehe 4.6).

Ablauf

Um Daten vom GPS-Modul geliefert zu bekommen, muss eine entsprechende Anfrage auf den Bus geschrieben werden. Das Signal `GPS_in` lässt das GPS-Modul die Daten vom Bus lesen. Während das GPS-Modul die Anfrage verarbeitet wird das Signal `GPS_out` gesetzt, um es dem GPS-Modul zu ermöglichen seine Antwort auf den Bus zu schreiben. Wenn das GPS-Modul seine Antwort vollständig auf den Bus geschrieben hat, setzt es das Signal `GPS_rdy` und erlaubt damit die weitere Verarbeitung der Antwort.

2.3 Aktuatoren

2.3.1 Antriebssteuerung

Die beiden Zustände werden mittels Signalen (`AS_start` und `AS_stop`) vom Prozessor gesteuert.

Signal	Beschreibung
`AS_start`	lässt das System mit etwa 2 km/h geradeaus losfahren
`AS_stop`	das Gerät stoppt

2.3.2 Lenkungssteuerung

Die drei Zustände werden vom Prozessor über Signale (`LS_left`, `LS_right` und `LS_stop`) gesteuert.

Signal	Beschreibung
`LS_left`	Chassis nach links drehen
`LS_right`	Chassis nach rechts drehen
`LS_stop`	Drehung stoppen

2.4 Datentypen und -formate

In der nachfolgenden Darstellung sind alle verwendeten Datentypen und ihre zugehörigen Wertebereiche dargestellt. Darüberhinaus werden Festlegungen über Werte getroffen, die vor Inbetriebnahme in einen reservierten Speicherbereich

geschrieben werden müssen, damit sie für bestimmte Berechnungen zur Verfügung stehen.

Daten	Wertebereich	Datentyp	Datenbreite
Anfrage an WLAN	feste Werte	egal	4 Bit
Anfrage an GPS	feste Werte	egal	4 Bit
Antwort von WLAN	$-850 - 3600$	Integer	13 Bit
Antwort von GPS	$0 - 5400$	Integer	13 Bit

Im Hauptspeicher stehen dabei nach der Initialisierung folgende Werte: An Adresse `0` steht die Breitenangabe des Zieles in Bogenminuten und an Adresse `1` der Zehntelsekundenanteil des Breitengrades. An Adresse `2` und `3` folgenden die korrespondierenden Längenangaben. Adresse `4` beinhaltet die Signalstärke, die als Maßstab für die gewünschte Entfernung zwischen je zwei Geräten genutzt werden soll. Im vorliegenden Fall legen wir als Zielsignalstärke -70,0 dB fest, was etwa einer Entfernung von 130m entspricht und als `-700` gespeichert ist. An Adresse `5` wird die Zahl -2^{15}, also -32768, gespeichert. Diese Zahl ist zu groß, um sie mit den von uns gewählten Befehlen als Immediate-Wert übergeben zu können. Wir hätten selbstverständlich ein anderes Befehlsformat wählen können, aber bei einem Gesamtumfang dieser Arbeit von über 30 Seiten haben wir den einfacheren Weg gewählt.

Die Folge dieser Festlegungen ist eine Beschränkung der zu verarbeitenden Werte auf ganze Zahlen, die als Zweierkomplement in einer Breite von 16 Bit dargestellt werden.

2.5 Verarbeitung und Speicherung

Aus den o. g. Konventionen für Daten, Wertebereiche und Datenbreite ergeben sich folgende Konsequenzen für unsere Architektur.

2.5.1 Bus

Busbreite Für den universellen Bus folgt aus den Datentypen und -formaten eine notwendige Breite von 16 Bit.

2.5.2 Register

Alle Register besitzen eine Wortbreite von 16 Bit.

Register Der Prozessor besitzt 4 General-Purpose-Register (R1 – R4) sowie ein ALU-Hilfsregister (Ra1).

Program Counter Der Programmzähler (PC) ist (wie z. B. bei der PDP-11) auch als normales Register ansprechbar, kann also mit den normalen Register-Schreib- und Lese-Befehlen angesprochen werden. Der Programmzähler wird beim Start des Gerätes mit der ersten Adresse des Programms initialisiert, also mit `6`.

Instruction Register Das Instruktionsregister (IR) wird nie direkt angesprochen sondern nur beim Befehlholen (Fetch).

Memory Address Register Das MAR wird nier direkt sondern nur beim Abarbeiten von Befehlen genutzt.

Memory Data Register Das MDR hält die aus dem Speicher ausgelesenen Daten vor und wird nur im Rahmen der Befehlsabarbeitung angesprochen.

2.5.3 Speicher

Speichergrösse Es werden 16-Bit-Worte abgespeichert.

Speicherzugriff Auf den Speicher wird über das Memory Address Register (MAR) und das Memory Data Register (MDR) zugegriffen.

Speicherart Der Speicher ist durch seine Nähe zum Prozessor sehr schnell. Ausserdem ist er nichtflüchtig.

Speicherinhalt Am Anfang des Speichers werden vor dem Start des Gerätes die Koordinaten des Mittelpunktes des Netzwerkes, die Zielsignalstärke und der Wert -2^{15} abgelegt. Ab Adresse `6` folgt das Programm.

2.5.4 Signale

MAR_in Mit diesem Signal wird das MAR veranlasst, eine Adresse einzulesen, um dann von dieser Speicheradresse Daten holen zu können.

MEM_read Das Signal steuert das Laden eines 16-Bit-Wortes von einer Adresse im MAR in den MDR

MEM_rdy Das Signal zeigt den Abschluss des Vorgangs an, in dem auf den Speicher zugegriffen wurde.

Kapitel 3

Mikroprozessorentwurf

3.1 Spezifikation

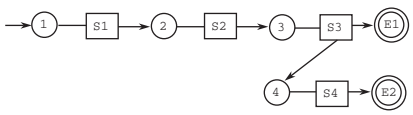
Aus unseren Anforderungen an die Verarbeitung und Speicherung von Daten ergeben sich folgende Konsequenzen für die einzusetzende Hardware.

- Es wird ein 16-Bit-Mikroprozessor benötigt.
- Der Prozessor wird durch Mikrocode gesteuert.

3.2 Verhaltensmodell

Wir haben das Verhalten des Systems unter Verwendung eines Zustandsübergangsgraphen modelliert. Dabei haben wir die einzelnen Schritte nicht nur in verschiedene Zustände aufgeteilt, sondern zur Verbesserung der Übersichtlichkeit den Graphen auch in mehrere Subgraphen aufgeteilt, die jeweils wieder spezielle komplexere Situationen in einfachere Schritte zerlegen.

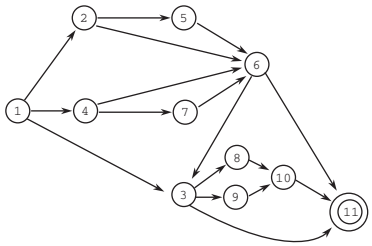
3.2.1 Hauptgraph



Zustand 1 ist der Startzustand, der bedingungslos in Subgraph I übergeht. Am Ende von Subgraph I befindet sich das Gerät auf dem gleichen Breitenkreis

wie der Zielpunkt. Zu Beginn des Subgraphen II wird überprüft, ob sich das Gerät inmitten eines aufgespannten Netzwerkes befindet. Wenn dies der Fall ist, fährt das Gerät aus dem Netzwerk heraus, andernfalls wird der Subgraph II sofort beendet und wir sind im Zustand 3 (des Hauptgraphen). Im Subgraphen III fährt das Gerät entweder bis an die Außengrenzen eines bestehenden Netzwerkes oder bis in den Zielpunkt. Im ersten Fall endet der Subgraph mit der Erkenntnis, dass ein Netzwerk existiert, im zweiten Fall „gründet“ das Gerät ein neues Netzwerk. In diesem Fall ist auch der Endzustand des Hauptgraphen erreicht. Im Subgraphen IV sucht das Gerät dann in dem bestehenden Netzwerk nach seiner Position. Dieser Subgraph (und damit auch der Hauptgraph) endet dann mit dem „Beitritt“ zum Netzwerk.

3.2.2 Subgraph I



Das Gerät befindet sich an einem beliebigen Punkt. Die einzige bekannte Information sind die Koordinaten des Mittelpunktes des Netzwerkes zu dem das Gerät gehören soll.

Zustand 1

In Zustand 1 des ersten Subgraphen wird vom GPS-Modul die Information über die Breitenminuten der Position des Gerätes abgefragt. Der Wert wird mit den Breitenminuten des Mittelpunktes verglichen und führt je nach Ergebnis des Vergleichs in einen neuen Zustand.

1 → 2 Wenn das Gerät sich nördlich vom Zielpunkt befindet.

1 → 3 Wenn das Gerät und der Zielpunkt die gleiche Breitenminute haben.

1 → 4 Wenn das Gerät sich südlich vom Zielpunkt befindet

Zustand 2

Das Gerät befindet sich nördlich des Zielpunktes und stellt seine Richtung mittels des GPS-Moduls fest.

2 → 5 Wenn das Gerät nicht in Richtung Süden steht.

2 → 6 Wenn das Gerät in Richtung Süden steht.

Zustand 5

Das Gerät dreht sich solange bis es in Richtung Süden steht.

5 → 6 Übergang ohne Bedingung

Zustand 6

Das Gerät fährt gerade aus und vergleicht dabei seine eigene Breitenminute ständig mit der Breitenminute des Zielpunktes. Sind die Breitenminuten gleich werden die Sekunden der Breite verglichen und das Ergebnis des Verleiches dient als Entscheidung für die Übergänge.

6 → 3 Wenn das Gerät sich nördlich vom Zielpunkt befindet.

6 → 3 Wenn das Gerät sich südlich vom Zielpunkt befindet

6 → 11 Wenn die Breitensekunden gleich sind.

Zustand 4

Das Gerät befindet sich südlich des Zielpunktes und prüft mittels des GPS-Moduls, ob die Fahrtrichtung tatsächlich Norden ist.

4 → 7 Wenn das Gerät nicht in Richtung Norden steht.

4 → 6 Wenn das Gerät in Richtung Norden steht.

Zustand 7

Das Gerät dreht sich solange bis es in Richtung Norden steht.

7 → 6 Übergang ohne Bedingung

Zustand 3

Das Gerät vergleicht die Breitensekunden, damit entschieden werden kann, ob sich das Gerät auf gleicher Breite mit dem Zielpunkt befindet.

3 → 8 Wenn das Gerät sich nördlich vom Zielpunkt befindet.

3 → 9 Wenn das Gerät sich südlich vom Zielpunkt befindet

3 → 11 Wenn die Breitensekunden gleich sind.

Zustand 8

Das Gerät dreht sich solange bis es in Richtung Süden steht.

8 → 10 Übergang ohne Bedingung

Zustand 9

Das Gerät dreht sich solange bis es in Richtung Norden steht.

9 → 10 Übergang ohne Bedingung

Zustand 10

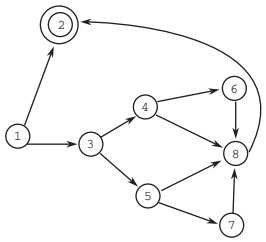
Das Gerät fährt, solange ein Unterschied zwischen den Breitensekunden des Zielpunktes und des Gerätes besteht.

10 → 11 Übergang ohne Bedingung

Zustand 11

Zustand 11 ist ein Endzustand von Subgraph I. Das Gerät befindet sich jetzt auf dem gleichen Breitenkreis wie der Zielpunkt.

3.2.3 Subgraph II



Zustand 1

Das Gerät prüft mittels des WLAN-Moduls ob es Kontakt zu einem Netzwerk aufnehmen kann.

1 → 2 Wenn das Gerät keinen Kontakt zu einem Netzwerk herstellen kann.

1 → 3 Wenn das Gerät Kontakt zu einem Netzwerk herstellen kann.

Zustand 3

Das Gerät vergleicht mittels der Werte aus dem GPS-Modul die Längenminuten und gegebenenfalls die Längensekunden. des Zielpunktes und des Gerätes selber.

3 → 4 Wenn das Gerät sich östlich des Zielpunktes befindet.

3 → 5 Wenn das Gerät sich westlich des Zielpunktes oder genau im Zielpunkt befindet.

Zustand 4

Das Gerät prüft mittels des GPS-Moduls, ob die Fahrtrichtung 90,0 Grad beträgt

4 → 6 Wenn die Fahrtrichtung ungleich 90,0 Grad ist.

4 → 8 Wenn die Fahrtrichtung gleich 90,0 Grad ist.

Zustand 5

Das Gerät prüft mittels des GPS-Moduls, ob die Fahrtrichtung 270,0 Grad beträgt.

5 → 7 Wenn die Fahrtrichtung ungleich 270,0 Grad ist.

5 → 8 Wenn die Fahrtrichtung gleich 270,0 Grad ist.

Zustand 6

Das Gerät dreht sich solange bis die Fahrtrichtung 90,0 Grad beträgt.

6 → 8 Übergang ohne Bedingung

Zustand 7

Das Gerät dreht sich solange bist die Fahrtrichtung 270,0 Grad beträgt.

7 → 8 Übergang ohne Bedingung

Zustand 8

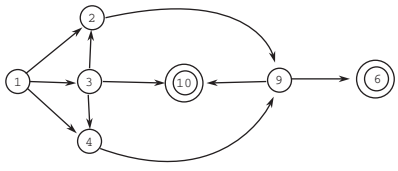
Das Gerät bewegt sich vorwärts, solange es Kontakt zu einem Netzwerk hat. Verliert es diesen Kontakt geht es in den Endzustand des Subgraphen über.

8 → 2 Übergang ohne Bedingung

Zustand 2

Zustand 2 ist ein Endzustand des Subgraphen II.

3.2.4 Subgraph III



Das Gerät befindet sich entweder westlich oder östlich des Netzwerkes auf dem gleichen Breitenkreis wie der Zielort oder es steht direkt im Zielpunkt, wobei im letzten Fall noch kein Netzwerk aufgespannt ist.

Zustand 1

Das Gerät überprüft anhand des Minutenanteils des Längengrads, ob sich das Gerät östlich oder westlich des Zielpunktes befindet oder ob sein Standort die gleiche Bogenminute wie der Zielpunkt aufweist.

1 → 2 Wenn das Gerät östlich des Zielpunktes liegt.

1 → 3 Wenn Gerät und Zielpunkt die gleiche Längenminute besitzen.

1 → 4 Wenn das Gerät westlich des Zielpunktes liegt.

Zustand 2

Das Gerät liegt östlich des Zielpunktes. Wenn das Gerät noch nicht nach Westen ausgerichtet ist (Kurs = 270,0 Grad), dann muss es sich erst in diese Richtung drehen.

2 → 9 Wenn der richtige Kurs anliegt.

Zustand 3

Die Bogenminute des Gerätestandortes entspricht derjenigen der Zielkoordinaten, daher muss geprüft werden, ob sie der Zehntelbogensekundenanteil des Längengrads von Standort und Zielpunkt unterscheidet. Wenn sich das Gerät direkt am Zielort befindet, existiert kein Netzwerk.

3 → 2 Wenn das Gerät östlich des Zielpunktes liegt.

3 → 4 Wenn das Gerät westlich des Zielpunktes liegt.

3 → 10 Wenn der Zielort erreicht ist.

Zustand 4

Da sich das Gerät westlich des Zielpunktes befindet, muss es sich nach Osten ausrichten, wenn dieser Kurs noch nicht anliegt.

4 → 9 Wenn der richtige Kurs anliegt.

Zustand 6

Das Gerät hält an, weil es die Außengrenzen eines bestehenden Netzwerkes erreicht hat. Zustand 6 ist ein Endzustand des Subgraphen III.

Zustand 9

Solange das Gerät weder ein bestehendes Netzwerk findet noch sich an den Koordinaten des Zielpunktes befindet, fährt es geradeaus weiter.

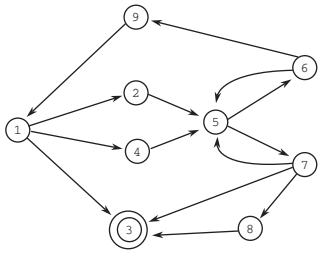
9 → 6 Wenn ein bestehendes Netzwerk detektiert wird.

9 → 10 Wenn der Zielpunkt erreicht wurde.

Zustand 10

Das Gerät hat den Zielpunkt erreicht, ohne ein Netzwerk zu finden. Das Gerät „gründet“ ein neues Netzwerk, indem es als zentraler Knoten konfiguriert wird. Zustand 10 ist ein Endzustand des Subgraphen III.

3.2.5 Subgraph IV



Das Gerät befindet sich entweder am östlichen oder am westlichen Rand eines bestehenden Netzwerkes.

Zustand 1

Nachdem das Gerät so nahe an das Netzwerk herangefahren ist, dass die Empfangsstärke -70 dB beträgt, wird die Position der Nachbargeräte überprüft. Wenn das Gerät weder einen Nachbarn direkt voraus noch zwei Nachbarn in je 30,0 Grad Abweichung vom eigenen Kurs besitzt, steht es somit direkt neben einer freien Position. Daher fährt es weiter geradeaus, bis der nächstliegende Nachbar sich bei genau 90,0 Grad befindet, um sich dann selbst um 90,0 Grad nach links zu drehen. Danach fährt es direkt zu der freien Position in dieser Schale des Netzwerkes, die es erreicht, wenn die Entfernung zu einem beliebigen Nachbarn genau 130m (d. h. -70 dB Empfangsstärke) beträgt.

1 → 2 Wenn sich ein Nachbar auf direktem Kurs des Gerätes befindet.

1 → 3 Übergang ohne Bedingung

1 → 4 Wenn das Gerät genau zwei Nachbarn hat.

Zustand 2

Das Gerät dreht sich um genau 90,0 Grad nach links.

2 → 5 Übergang ohne Bedingung

Zustand 3

Das Gerät befindet sich an der individuellen Zielposition und muss daher jetzt dem Netzwerk „beitreten“. Zustand 3 ist der Endzustand von Subgraph IV.

Zustand 4

Das Gerät muss sich um genau 90,0 Grad nach links drehen und dann so lange geradeaus fahren, bis sich der nächstliegende Nachbar in einer 90-Grad-Position zum eigenen Kurs befindet.

4 → 5 Übergang ohne Bedingung

Zustand 5

Das Gerät fährt so lange geradeaus, bis sich der nächstliegende Nachbar in einer 115-Grad-Position zum eigenen Kurs befindet, also schräg rechts hinter dem Gerät. Dann testen wir, ob ein zweites Gerät in Empfangsreichweite liegt.

5 → 6 Wenn (mindestens) zwei Geräte in Empfangsreichweite sind.

5 → 7 Übergang ohne Bedingung

Zustand 6

Nachdem wir jetzt wissen, dass das Gerät mindestens zwei Nachbarn hat, testen wir auf die Existenz eines dritten. Wenn es nur genau zwei Nachbarn gibt, muss das Gerät wieder geradeaus fahren, bis es eine 90-Grad-Position zum nähesten Nachbarn erreicht hat.

6 → 9 Wenn genau drei Geräte in Empfangsreichweite sind.

6 → 5 Übergang ohne Bedingung

Zustand 7

Es befindet sich nur genau ein Nachbar in Reichweite des Gerätes – schräg rechts hinter ihm. Das Gerät fährt geradeaus, bis der Winkel 120,0 Grad beträgt. Es steht nun auf einer potentiellen Zielposition, daher wird getestet, ob sich das Gerät genau nördlich des Mittelpunktes befindet. In diesem Fall muss eine neue Schale „eröffnet“ werden.

Andernfalls dreht sich das Gerät so lange nach rechts, bis es zum Nachbarn eine 90-Grad-Position innehat. Dann fährt das Gerät erst so lange geradeaus, bis die Entfernung zum nächsten Nachbarn so weit gesunken, dass die Signalstärke echt größer als -70 dB ist, danach weiter bis zum Punkt mit einer Signalstärke von -70 dB und testet dort auf weitere Geräte in der Nachbarschaft. Wenn es genau drei Nachbarn gibt, hat das Gerät eine gültige freie Position für den Netzwerkbeitritt gefunden. Wenn es genau zwei Nachbarn hat, befindet es sich jetzt auf einer neuen Seite des Netzwerkes und fährt daher weiter, bis zum „90-Grad-Punkt“, an dem dann wieder der „Seitenabfahralgorithmus“ aufgerufen wird.

7 → 3 Wenn das Gerät seine Zielposition im Netzwerk erreicht hat.

7 → 5 Aufruf des „Seitenabfahralgorithmus“

7 → 8 Wenn sich die Zielposition des Gerätes im Netzwerk genau hinter einer Ecke befindet.

Zustand 8

Das Gerät hat gerade eine der sechs Ecken des Netzwerkes umfahren. Da die Zielposition direkt hinter der Ecke liegt, dreht sich das Gerät in Richtung der Zielposition und fährt dann dort hin.

8 → 3 Übergang ohne Bedingung

Zustand 9

Das Gerät hat sich der nördlichsten Position des Netzwerkes genähert und dabei festgestellt, das die neue Schale bereits mindestens ein anderes Gerät enthält, d. h. es muss seinen Abstand vom Mittelpunkt so weit vergrößern, dass es den Weg um das Netzwerk wieder aufnehmen kann.

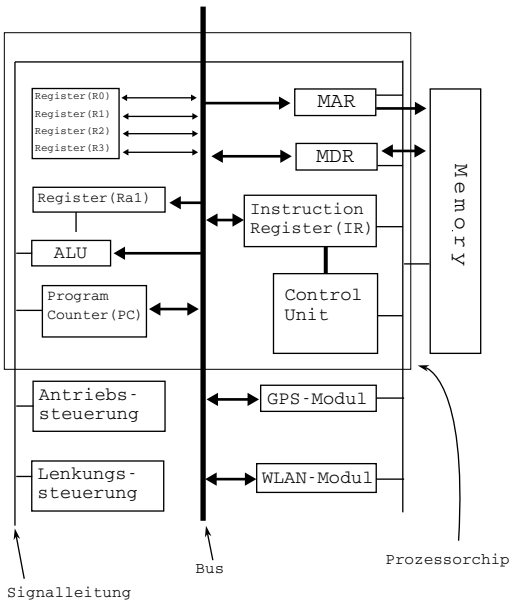
Dabei fährt das Gerät erst so lange geradeaus, bis der nächste Nachbar direkt voraus liegt – dies ist das Gerät an der nördlichsten Position des Netzwerkes. Dann fährt es einfach am diesem vorbei, bis der Abstand zwischen bei wieder auf 130m gestiegen ist. Danach dreht sich das Gerät so lange, bis der eigene Kurs senkrecht zum Rand des Netzwerkes steht. Dies ist dann für das Gerät die gleiche relative Position wie beim erstmaligen Auftreffen auf das Netzwerk.

9 → 1 Übergang ohne Bedingung

3.2.6 Graphen und Programm

Obwohl wir anhand des Graphen die Korrektheit des Modells zeigen können, haben wir das Modell nicht 1:1 in den Programmcode übernommen. Stattdessen haben wir vor allem an den Stellen, bei denen es im Zustandsübergangsgraphen zu unbedingten Sprüngen kommt, den Programmablauf so gestaltet, dass die Anzahl der Sprünge minimiert wird. Dadurch ist jedoch die Reihenfolge der Abfragen und Auswertungen im Programm teilweise anders als im Modell. Dennoch sollte sich das Programm (auch durch den Versuch, die einzelnen Sprungmarken nach den jeweiligen Zuständen aus dem Modell zu benennen) leicht mit Hilfe des Modells überprüfen lassen.

3.3 Schaltungsentwurf



3.4 Befehlsformate und Befehlssatz

Nachfolgend sind alle verwendeten Befehle und ihre Funktionsweise dargestellt.

Die OP-Code werden in der Art einer Huffman-Codierung festgelegt. Die Gründe hierfür liegen in der notwendigen Datenbreite bei der Benutzung von Immediate-Werten bei den Move- und Jump-Befehlen.

`Rx` und `Ry` stehen dabei für beliebige Register, `Sy` kann das GPS-Modul oder das WLAN-Modul sein.

3.4.1 Move-Befehle

Alle Move-Befehle sind 2-Adress-Befehle. Dabei wird ein Wert in ein Register oder einen Sensor geschrieben, der selber aus einem Register, einem Sensor, aus einer Speicheradresse oder als immediate-Wert aus dem Befehl kommen kann.

`movD #n, Rx`

Der Befehl `movD` hat mit `1` den kürzesten OP-Code, weil es nur dann möglich ist, mit diesem Befehl 12-Bit-Zahlen zu kopieren. Es handelt sich hierbei also um eine Befehlskodierung, die sich an den Huffman-Algorithmus anlehnt, aber statt die Befehlswortverteilung nach Häufigkeit vorzunehmen, wählen wir die Verteilung nach der notwendigen Wortbreite bei Immediate-Zahlen. Die größte Zahl, die wir als Immediate-Wert kopieren müssen, hat einen Wert von 2700 und braucht daher einen Speicherbereich von 12 Bit.

OP-Code	Quelle	Ziel
`1`	`#n`	`Rx`
1 Bit	12 Bit	3 Bit

`movD` hat dabei folgende Bedeutung:

`movD #n, Rx`	`Rx` ← `n`	n wird in Rx geschrieben

weitere Move-Befehle

Alle anderen Move-Befehle – die „normalen“, nicht die Sprungbefehle – haben einen OP-Code von 8 Bit und jeweils 4 Bit für Quelle und Ziel und beginnen jeweils mit `0001`.

OP-Code	Quelle	Ziel
`0001XXXX`		
8 Bit	4 Bit	4 Bit

Dabei steht `XXXX` im OP-Code jeweils für:

`0000`	`movR Rx, Ry`	`Ry` ← `Rx`	Rx wird in Ry geschrieben
`0001`	`movM addr, Rx`	`Rx` ← (`addr`)	lade aus dem Speicher
`0010`	`movPR Rx, Sy`	`Sy` ← `Rx`	Anfrage in 4-Bit-Code
`0011`	`movPD #n, Sy`	`Sy` ← `#n`	Anfrage in 4-Bit-Code
`0100`	`movG Sy, Rx`	`Rx` ← `Sy`	lade Daten aus Sensor y

Unbedingte Sprünge

Die Jump-Befehle sind ihrer Funktion nach Move-Befehle, bei denen ein Wert in den Program Counter (PC) geschrieben wird. Im Gegensatz zu den anderen Move-Befehlen handelt es sich dabei um 1-Adress-Befehle, weil die zweite Adresse (der PC) immer gleich ist. Alle Jump-Befehle haben eine Länge von 6 Bit und beginnen jeweils mit `001`.

OP-Code	Ziel
001XXX	
6 Bit	10 Bit

Dabei steht XXX im OP-Code jeweils für:

000	jmp n	PC ← n	Sprungbefehl an eine Adresse n im Speicher
001	jmp Rx	PC ← Rx	Sprungadresse liegt in Rx

Bedingte Sprünge

If-Befehle sind Move-Befehle, bei denen die Sprungausführung von einem bestimmten Argument-Bit abhängig ist. Dabei wird der Befehl nur ausgeführt, wenn das jeweilige Argument-Bit gesetzt ist.

Dabei steht XXX im OP-Code jeweils für:

010	ifeq n	jmp n if eq	Sprung wenn Flag eq gesetzt ist
011	ifne n	jmp n if not eq	Sprung wenn Flag eq nicht gesetzt ist
100	iflt n	jmp n if lt	Sprung wenn Flag lt gesetzt ist
101	ifgt n	jmp n if gt	Sprung wenn Flag gt gesetzt ist

ALU-Befehle

Die folgenden Befehle weisen die ALU an, eine Operation mit den übergebenen Argumenten auszuführen.

`sub #n, Rx`

Der OP-Code von sub ist mit 01 nur unwesentlich länger als der von movD und wurde von uns deshalb so gewählt, weil die größte Zahl, die wir als Subtrahenden benutzten, 900 ist und damit einen Speicherbereich von 10 Bit braucht.

OP-Code	Quelle	Ziel
01	#n	Rx
2 Bit	11 Bit	3 Bit

sub hat dabei folgende Bedeutung:

sub #n, Rx	Rx ← Rx − n	Rx = Rx − n

weitere ALU-Befehle

Die OP-Codes der anderen ALU-Befehle beginnen mit 0001, wobei die Länge des OP-Codes immer 8 Bit ist.

OP-Code		
0001XXXX		
8 Bit	4 Bit	4Bit

XXXX steht dabei für folgende Befehle:

0101	`inc Rx`	Rx ← Rx + 1	Inkrementierung von Rx
0110	`cmp Rx, Ry`	Rx ? Ry	Vergleichsergebnis induziert Flag
0111	`add #n, Rx`	Rx ← Rx + n	Rx = Rx + n

Da es sich beim Inc-Befehl nur um einen 1-Adress-Befehl handelt, werden dabei die letzten 4 Bit ignoriert.

sonstige Befehle

Diese Befehle werden ohne Parameter aufgerufen. Die CU setzt auf Grund dieser Befehle ein bestimmtes Signal. Die OP-Codes aller Befehle starten mit `00011`.

OP-Code	
`00011XXX`	
8 Bit	8 Bit

`XXX` steht dabei jeweils für:

000	`AS_start`	Das Gerät fährt mit ca. 2 km/h.
001	`AS_stop`	Der Motor wird ausgeschaltet.
010	`LS_left`	Das Chassis dreht nach links.
011	`LS_right`	Das Chassis dreht nach rechts.
100	`LS_stop`	Die Drehung wird gestoppt.
101	`GPS_on`	Das GPS-Modul wird eingeschaltet.
110	`GPS_off`	Das GPS-Modul wird ausgeschaltet.
111	`WLAN_on`	Das WLAN-Modul wird eingeschaltet.

3.5 Steuereinheit – Mikrocode

Die Steuereinheit (CU) ist Mikroprogramm-gesteuert. Der Mikrocode-Zähler beginnt bei 0, mit RUN springt er zum nächsten Mikrobefehl und mit END springt der Zähler zurück auf 0. Der Zähler springt auch auf 0, wenn der nächste Befehl „Fetch“ lautet. Nach dem Befehlsholen (Fetch) dekodiert die CU den OP-Code des Befehls und springt jeweils zur richtigen Stelle im Mikroprogramm. Da es 35 Mikrobefehle gibt und jeder Befehl die Adresse des danach auszuführenden Befehls enthält, brauchen wir jeweils 6 Bit zur Darstellung der Adresse, insgesamt also 12 Bit für einen Befehl. Der Mikrobefehlsspeicher muss also (bei 35 Befehlen und je 12 Bit) 420 Bit umfassen.

Die folgende Tabelle zeigt, welche Befehle in welchem Schritt welche Signale setzen bzw. überprüfen, welche Signale (z. B. von der ALU) gesetzt wurden.

	Signale	MAR_in	MEM_read	MEM_rdy		Rx_in	Rx_out	Ry_in	Ry_out	IR_in	IR_out	MDR_out	Ra1_in	ALU_in	ALU_out	Sy_rdy	Sy_in	Sy_out	WLAN_on	GPS_on	GPS_off	AS_start	AS_stop	LS_left	LS_right	LS_stop	PC_in	PC_out	eq	not eq	gt	lt	cmp	inc	add	sub	RUN	nächster Befehl		
Adresse	Befehl																																							
0	Fetch	x	x											x														x					x			x	1		
1															x												x									x	2		
2				x						x		x																								x	3	RUN if MEM_rdy	
3	Decode																																				siehe OP-Code		
4	movR Rx, Ry					x	x																													x	Fetch		auch PC
5	movD #n, Rx				x						x																									x	Fetch		IR[14..3]
6	movM addr, Rx	x	x								x																									x	7		IR[7..4]
7				x	x							x																								x	Fetch	RUN if MEM_rdy	
8	movPR Rx, Sy						x										x																			x	Fetch		
9	movPD #n, Sy										x						x																			x	Fetch		IR[7..4]
10	movG Sx, Ry							x								x		x																		x	Fetch	RUN if Sy_rdy	
11	inc Rx						x							x																				x		x	12		
12					x										x																					x	Fetch		
13	add #n, Rx						x						x																							x	14		
14											x			x																					x	x	15		IR[7..4]
15						x									x																					x	Fetch		
16	sub #n, Rx						x						x																							x	17		
17											x			x																					x	x	18		IR[13..3]
18					x										x																					x	Fetch		
19	cmp Rx, Ry						x						x																							x	20		
20									x																								x			x	Fetch		
21	ifeq n										x																x		x							x	Fetch	if Flag, else END	IR[9..0]
22	ifne n										x																x			x						x	Fetch	if Flag, else END	IR[9..0]
23	ifgt n										x																x				x					x	Fetch	if Flag, else END	IR[9..0]
24	iflt n										x																x					x				x	Fetch	if Flag, else END	IR[9..0]
25	jmp n										x																x									x	Fetch		IR[9..0]
26	jmp Rx						x																				x									x	Fetch		
27	AS_start																					x															x	Fetch		
28	AS_stop																						x														x	Fetch		
29	LS_left																							x													x	Fetch		
30	LS_right																								x												x	Fetch		
31	LS_stop																									x											x	Fetch		
32	GPS_on																			x																	x	Fetch		
33	GPS_off																				x																x	Fetch		
34	WLAN_on																		x																		x	Fetch		

3.6 Applikation

```
# Feste Werte im Speicher
# MEM[0]: (ziel_breite_min)
# MEM[1]: (ziel_breite_sec)
# MEM[2]: (ziel_laenge_min)
# MEM[3]: (ziel_laenge_sec)
# MEM[4]: (daempfung: hier immer -700)
# MEM[5]: -32768

# Initialisierung
        # Schalte GPS und WLAN an
        GPS_on
        WLAN_on

# Beginn Subgraph I
        # lade die aktuelle Breite (min) in Ra1
        movPD #1, GPS     # GPS <- 0001 (befehl "0001" an gps-modul)
        movG GPS, Ra1     # Ra1 <- GPS (aktuelle breite in min)
        movM 0, R0        # lade Zielbreite in R0
        cmp Ra1, R0       # vergleiche die aktuelle mit der Zielbreite
        ifgt aZwei        # wenn Ra1 > R0 goto aZwei
        iflt aVier        # wenn Ra1 < R0 goto aVier
```

```
        # lade die aktuelle Breite (sec) in Ra1
        movPD #2, GPS    # GPS <- 0010 (befehl "0010" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktuelle breite in sec)
        movM 1, R0       # lade Zielbreite in R0
        cmp Ra1, R0      # vergleiche die aktuelle mit der Zielbreite
        ifgt aAcht       # wenn Ra1 > R1 goto aAcht
        iflt aNeun       # wenn Ra1 < R1 goto aNeun
        jmp aZehn        # das System befindet sich auf dem richtigen Breitengrad

:aZwei  movPD #8, GPS    # GPS <- 1000 (befehl "1000" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktueller kurs)
        movD #1800, R1   # R1 <- 1800
        cmp Ra1, R1      # steht das System in Richtung Sueden?
        ifeq aSechs      # wenn es in Richtung Sueden steht goto aSechs
        # Subroutine: gps_tl(R1) (linksdrehen)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp gps_tl       # springe zur Subroutine

:aSechs movM 0, R1       # lade Zielbreite in R1
        movD #1, R2      # R2 <- 0001
        # Subroutine: gps_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp gps_go       # springe zur Subroutine
        # lade die aktuelle Breite (sec) in Ra1
        movPD #2, GPS    # GPS <- 0010 (befehl "0010" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktuelle breite in sec)
        movM 1, R0       # lade Zielbreite in R0
        cmp Ra1, R0      # vergleiche die aktuelle mit der Zielbreite
        ifgt aAcht       # wenn Ra1 > R1 goto aAcht
        iflt aNeun       # wenn Ra1 < R1 goto aNeun
        jmp aElf         # das System befindet sich auf dem richtigen Breitengrad

:aVier  movPD #8, GPS    # GPS <- 1000 (befehl "1000" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktueller kurs)
        movD #0, R1      # R1 <- 0
        cmp Ra1, R1      # steht das System in Richtung Norden?
        ifeq aSechs      # wenn es in Richtung Norden steht goto aSechs
        # Subroutine: gps_tl(R1) (linksdrehen)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp gps_tl       # springe zur Subroutine
        jmp aSechs

:aAcht  movD #1800, R1   # R1 <- 1800
        # Subroutine: gps_tl(R1) (linksdrehen)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp gps_tl]      # springe zur Subroutine
```

```
        jmp aZehn

:aNeun  movD #0, R1      # R1 <- 0
        # Subroutine: gps_tl(R1) (linksdrehen)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp gps_tl       # springe zur Subroutine

:aZehn  movM 1, R1       # lade Zielbreite in R1
        movD #2, R2      # R2 <- 0010
        # Subroutine: gps_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp gps_go       # springe zur Subroutine

# Ende Subgraph I
# Begin subgraph II

:aElf   movPD #0, WLAN   # WLAN <- 0000 (befehl "0000" an wlan-modul)
        movG WLAN, Ra1   # Ra1 <- WLAN (entfernung zum naechsten peer)
        movM 5, R0       # lade -32768 in R0
        cmp Ra1, R0      # gibt es einen peer
        ifeq bZwei       # falls es keinen peer gibt goto bZwei

        # lade die aktuelle Laenge (min) in Ra1
        movPD #5, GPS    # GPS <- 0101 (befehl "0101" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktuelle laenge in min)
        movM 2, R0       # lade Ziellaenge in R0
        cmp Ra1, R0      # vergleiche die aktuelle mit der Ziellaenge (min)
        ifgt bVier       # wenn Ra1 > R0 goto bVier
        iflt bFuenf      # wenn Ra1 < R0 goto bFuenf
        # lade die aktuelle Laenge (sec) in Ra1
        movPD #6, GPS    # GPS <- 0110 (befehl "0110" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktuelle laenge in sec)
        movM 3, R0       # lade Ziellaenge in R0
        cmp Ra1, R0      # vergleiche die aktuelle mit der Ziellaenge (sec)
        iflt bFuenf      # wenn Ra1 < R0 goto bFuenf

:bVier  movPD #8, GPS    # GPS <- 1000 (befehl "1000" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktueller kurs)
        movD #900, R1    # R1 <- 900
        cmp Ra1, R1      # steht das System in Richtung Osten?
        ifeq bZehn       # wenn es in Richtung Osten steht goto bZehn
        # Subroutine: gps_tl(R1)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp gps_tl       # springe zur Subroutine
        jmp bZehn

:bFuenf movPD #8, GPS    # GPS <- 1000 (befehl "1000" an gps-modul)
```

```
        movG GPS, Ra1    # Ra1 <- GPS (aktueller kurs)
        movD #2700, R1   # R1 <- 2700
        cmp Ra1, R1      # steht das System in Richtung Westen?
        ifeq bZehn       # wenn es in Richtung Westen steht goto bZehn
        # Subroutine: gps_tl(R1)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp gps_tl       # springe zur Subroutine

:bZehn  movM 5, R1       # R1 <- -32768
        movD #0, R2      # R2 <- 0000
        # Subroutine: w_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_go         # springe zur Subroutine

# Ende Subgraph II
# Beginn Subgraph III

:bZwei  movPD #5, GPS    # GPS <- 0101 (befehl "0101" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktuelle laenge in min)
        movM 2, R0       # lade Ziellaenge in R0
        cmp Ra1, R0      # vergleiche die aktuelle mit der Ziellaenge (min)
        ifgt cZwei       # wenn Ra1 > R0 goto cZwei
        iflt cVier       # wenn Ra1 < R0 goto cVier
        # lade die aktuelle Laenge (sec) in Ra1
        movPD #6, GPS    # GPS <- 0110 (befehl "0110" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktuelle laenge in sec)
        movM 3, R0       # lade Ziellaenge in R0
        cmp Ra1, R0      # vergleiche die aktuelle mit der Ziellaenge (sec)
        ifgt cZwei       # wenn Ra1 > R0 goto cZwei
        iflt cVier       # wenn Ra1 < R0 goto cVier
        jmp cZehn        # System ist am Zielpunkt

:cZwei  movPD #8, GPS    # GPS <- 1000 (befehl "1000" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktueller kurs)
        movD #2700, R1   # R1 <- 2700
        cmp Ra1, R1      # steht das System in Richtung Westen?
        ifeq cNeun       # wenn es in Richtung Westen steht goto cNeun
        # Subroutine: gps_tl(R1)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp gps_tl       # springe zur Subroutine
        jmp cNeun

:cVier  movPD #8, GPS    # GPS <- 1000 (befehl "1000" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktueller kurs)
        movD #900, R1    # R1 <- 900
        cmp Ra1, R1      # steht das System in Richtung Osten?
        ifeq cNeun       # wenn es in Richtung Osten steht goto cNeun
```

```
        # Subroutine: gps_tl(R1)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp gps_tl       # springe zur Subroutine

:cNeun  AS_start         # motor.an
:go     movPD #0, WLAN   # WLAN <- 0000 (befehl "0000" an wlan-modul)
        movG WLAN, Ra1   # Ra1 <- WLAN (entfernung zum naechsten peer)
        movM 5, R1       # lade R1 mit -32768
        cmp Ra1, R1      # gibt es einen peer
        ifne cSechs      # falls es einen peer gibt goto cSechs
        movPD #5, GPS    # GPS <- 0101 (befehl "0101" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktuelle laenge in min)
        movM 2, R0       # lade Ziellaenge in R0
        cmp Ra1, R0      # vergleiche die aktuelle mit der Ziellaenge (min)
        ifne go
        movPD #6, GPS    # GPS <- 0110 (befehl "0110" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktuelle laenge in sec)
        movM 3, R0       # lade Ziellaenge in R0
        cmp Ra1, R0      # vergleiche die aktuelle mit der Ziellaenge (sec)
        ifne go
        AS_stop          # motor.aus

:cZehn  movPD #8, WLAN   # set(wlan(1000)); netzwerk gruenden
        GPS_off          # GPS wird nicht mehr gebraucht
        HALT
        # Ende Subgraph III und Hauptgraph

:cSechs AS_stop          # motor.aus
                         # Netzwerk gefunden
        # Ende Subgraph III

:dEins  movPD #1, WLAN   # WLAN <- 0001 (befehl "0001" an wlan-modul)
        movG WLAN, Ra1   # Ra1 <- WLAN (winkel zum naechsten peer)
        movD #0, R0
        cmp Ra1, R0      # ist peer direkt voraus?
        ifeq dZwei
        # der peer liegt nicht direkt voraus
        movPD #2, WLAN   # WLAN <- 0010 (befehl "0010" an wlan-modul)
        movG WLAN, Ra1   # Ra1 <- WLAN (entfernung zum zweiten peer)
        movM 5, R0       # lade -32768 in R0
        cmp Ra1, R0      # gibt es einen zweiten peer?
        ifne dVier       # falls es einen zweiten peer gibt goto dVier
        # schraeg links vor dem geraet ist eine freie position
        movD #900, R1    # R1 <- 900
        movD #1, R2      # R2 <- 0001
        # Subroutine: w_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_go         # springe zur Subroutine
```

```
        movD #1800, R1   # R1 <- 1800
        movD #1, R2      # R2 <- 0001
        # Subroutine: w_tl(R1, R2) (linksdrehen)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_tl         # springe zur Subroutine
        movM 4, R1       # lade Zielsignalstaerke in R1
        movD #0, R2      # R2 <- 0000
        # Subroutine: w_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_go         # springe zur Subroutine
        # das funktioniert aufgrund der triangulation, bei der die
        # entfernung zu allen nachbarn gleich (hier -700) ist.
        # welches geraet als 00 erkannt wird, spielt dabei keine rolle.

:dDrei  movPD #9, WLAN   # set(wlan(1001)); netzwerk beitreten
        GPS_off          # GPS wird nicht mehr gebraucht
        HALT
        # Ende Subgraph IV und Hauptgraph

:dVier  movPD #8, GPS    # GPS <- 1000 (befehl "1000" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktueller kurs)
        sub #900, Ra1    # 90 Grad nach links
        movR Ra1, R1     # R1 <- Ra1
        # Subroutine: gps_tl(R1)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp gps_tl       # springe zur Subroutine
        movD #900, R1    # R1 <- 900
        movD #1, R2      # R2 <- 0001
        # Subroutine: w_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_go         # springe zur Subroutine
        jmp dFuenf

:dZwei  movPD #8, GPS    # GPS <- 1000 (befehl "1000" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktueller kurs)
        sub #900, Ra1    # 90 Grad nach links
        movR Ra1, R1     # R1 <- Ra1
        # Subroutine: gps_tl(R1)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp gps_tl       # springe zur Subroutine

:dFuenf movD #1150, R1   # R1 <- 1150
        movD #1, R2      # R2 <- 0001
        # Subroutine: w_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
```

```
        add #2, R3       # erhoehe R3 um 2
        jmp w_go         # springe zur Subroutine
        movPD #2, WLAN   # WLAN <- 0010 (befehl "0010" an wlan-modul)
        movG WLAN, Ra1   # Ra1 <- WLAN (entfernung zum zweiten peer)
        movM 5, R1       # lade -32768 in R1
        cmp Ra1, R1      # gibt es einen zweiten peer?
        ifeq dSieb       # falls es keinen zweiten peer gibt goto dSieb
        # es gibt also mindestens zwei nachbarn
        # auch einen dritten?
        movPD #4, WLAN   # WLAN <- 0100 (befehl "0100" an wlan-modul)
        movG WLAN, Ra1   # Ra1 <- WLAN (entfernung zum dritten peer)
        cmp Ra1, R1      # gibt es einen dritten peer?
        ifne dNeun       # falls es einen dritten peer gibt goto dNeun
        # es gibt also genau zwei nachbarn
        movD #900, R1    # R1 <- 900
        movD #1, R2      # R2 <- 0001
        # Subroutine: w_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_go         # springe zur Subroutine
        jmp dFuenf

:dSieb  movD #1200, R1   # R1 <- 1200
        movD #1, R2      # R2 <- 0001
        # Subroutine: w_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_go         # springe zur Subroutine
        # testen, ob das geraet genau im norden des zielpunktes steht
        # lade die aktuelle Richtung in Ra1
        movPD #8, GPS    # GPS <- 1000 (befehl "1000" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktueller kurs)
        movD #600, R0    # eigener Kurs nordoestlich => noerdlichster Punkt
        cmp Ra1, R0
        ifne next        # wenn anderer Kurs
        # lade die aktuelle Laenge (min) in Ra1
        movPD #5, GPS    # GPS <- 0101 (befehl "0101" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktuelle laenge in min)
        movM 2, R0       # lade Ziellaenge in R0
        cmp Ra1, R0      # vergleiche die aktuelle mit der Ziellaenge (min)
        ifne next        # wenn Kurs richtig, aber Position falsch
        # lade die aktuelle Laenge (sec) in Ra1
        movPD #6, GPS    # GPS <- 0110 (befehl "0110" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktuelle laenge in sec)
        movM 3, R0       # lade Ziellaenge in R0
        cmp Ra1, R0      # vergleiche die aktuelle mit der Ziellaenge (sec)
        ifne next        # wenn Kurs und Minute richtig, aber Rest falsch
        jmp dDrei

:next   movD #900, R1    # R1 <- 900
```

```
        movD #1, R2      # R2 <- 0001
        # Subroutine: w_tr(R1, R2) (rechtsdrehen)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_tr         # springe zur Subroutine

        movM 4, R1       # lade Zielsignalstaerke in R1
        inc R1           # da steht dann -699 (zumindest in unserem beipiel)
        # das Geraet faehrt also hier nur ein kleines Stueck, gerade weit
        # genug, um im naechsten Schritt wieder bis zur Zielsignalstaerke
        # zu fahren
        movD #0, R2      # R2 <- 0000
        # Subroutine: w_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_go         # springe zur Subroutine

        movM 4, R1       # lade Zielsignalstaerke in R1
        movD #0, R2      # R2 <- 0000
        # Subroutine: w_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_go         # springe zur Subroutine

        movPD #4, WLAN   # WLAN <- 0100 (befehl "0100" an wlan-modul)
        movG WLAN, Ra1   # Ra1 <- WLAN (entfernung zum dritten peer)
        movM 5, R0       # R0 <- -32768
        cmp Ra1, R0      # gibt es einen dritten peer?
        ifne dDrei       # falls es einen dritten peer gibt goto dDrei
        movPD #2, WLAN   # WLAN <- 0010 (befehl "0010" an wlan-modul)
        movG WLAN, Ra1   # Ra1 <- WLAN (entfernung zum zweiten peer)
        cmp Ra1, R0      # gibt es einen zweiten peer?
        ifeq dAcht       # falls es keinen zweiten peer gibt goto dAcht

        movD #900, R1    # R1 <- 900
        movD #1, R2      # R2 <- 0001
        # Subroutine: w_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_go         # springe zur Subroutine
        jmp dFuenf

:dAcht  movD #600, R1    # R1 <- 600
        movD #1, R2      # R2 <- 0001
        # Subroutine: w_tr(R1, R2) (rechtsdrehen)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_tr         # springe zur Subroutine

        movM 4, R1       # lade Zielsignalstaerke in R1
```

```
        movD #4, R2      # R2 <- 0100
        # Subroutine: w_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_go         # springe zur Subroutine
        jmp dDrei

:dNeun  movD #0, R1      # R1 <- 0
        movD #1, R2      # R2 <- 0001
        # Subroutine: w_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_go         # springe zur Subroutine
        movM 4, R1       # lade Zielsignalstaerke in R1
        movD #4, R2      # R2 <- 0100
        # Subroutine: w_go(R1, R2)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp w_go         # springe zur Subroutine
        movD #2100, R1   # R1 <- 2100
        # Subroutine: gps_tl(R1) (linksdrehen)
        movR PC, R3      # speichere den PC in R3
        add #2, R3       # erhoehe R3 um 2
        jmp gps_tl       # springe zur Subroutine
        # das Geraet steht jetzt senkrecht auf dem Rand des Feldes
        # kurz hinter der noerdlichen Ecke und geht in Zustand dEins
        jmp dEins

# Subroutine: gps_tl(R1) (linksdrehen in Abhaengigkeit der Daten vom GPS)
:gps_tl LS_left          # starte die Linksdrehung
:gps_l  movPD #8, GPS    # GPS <- 1000 (befehl "1000" an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktueller kurs)
        cmp Ra1, R1      # vergleiche den aktuellen mit dem Zielkurs
        ifne gps_l       # wenn aktueller Kurs != Zielkurs => drehe weiter
        LS_stop          # stoppe die Linksdrehung
        movR R3, PC      # verlasse die Subroutine und springe zurueck


# Subroutine: gps_go(R1, R2) (fahren nach GPS)
:gps_go AS_start         # das Geraet faehrt los
:gps_g  movPR R2, GPS    # GPS <- R2 (befehl aus R2 an gps-modul)
        movG GPS, Ra1    # Ra1 <- GPS (aktueller wert)
        cmp Ra1, R1      # vergleiche den aktuellen mit dem Zielwert
        ifne gps_g       # wenn der Zielwert noch nicht erreicht wurde
        AS_stop          # Geraet anhalten
        movR R3, PC      # verlasse die Subroutine und springe zurueck


# Subroutine: w_go(R1, R2) (fahren in Abhaengigkeit der Daten vom WLAN)
:w_go   AS_start         # das Geraet faehrt los
```

```
:w_g    movPR R2, WLAN   # WLAN <- R2 (befehl aus R2 an wlan-modul)
        movG WLAN, Ra1   # Ra1 <- WLAN (aktueller wert)
        cmp Ra1, R1      # vergleiche den aktuellen mit dem Zielwert
        ifne w_g         # wenn der Zielwert noch nicht erreicht wurde
        AS_stop          # Geraet anhalten
        movR R3, PC      # verlasse die Subroutine und springe zurueck


# Subroutine: w_tl(R1, R2) (linksdrehen nach WLAN-Empfang)
:w_tl   LS_left          # starte die Linksdrehung
:w_l    movPR R2, WLAN   # WLAN <- R2 (befehl aus R2 an wlan-modul)
        movG WLAN, Ra1   # Ra1 <- WLAN (aktueller wert)
        cmp Ra1, R1      # vergleiche den aktuellen mit dem Zielkurs
        ifne w_l         # wenn aktueller Kurs != Zielkurs => drehe weiter
        LS_stop          # stoppe die Linksdrehung
        movR R3, PC      # Subroutine verlassen und zurueck


# Subroutine: w_tr(R1, R2) (rechtsdrehen nach WLAN-Empfang)
:w_tr   LS_right         # starte die Rechtsdrehung
:w_r    movPR R2, WLAN   # WLAN <- R2 (befehl aus R2 an wlan-modul)
        movG WLAN, Ra1   # Ra1 <- WLAN (aktueller wert)
        cmp Ra1, R1      # vergleiche den aktuellen mit dem Zielkurs
        ifne w_r         # wenn aktueller Kurs != Zielkurs => drehe weiter
        LS_stop          # stoppe die Rechtsdrehung
        movR R3, PC      # Subroutine verlassen und zurueck
```

Kapitel 4

Schlussbemerkungen

4.1 Bewertung der Einsatzmöglichkeiten

Für den Einsatz sich autonom organisierender Roboter lassen sich viele Situationen finden. Die Geräte stellen lediglich eine Plattform dar, an die sich weitere Module anschliessen lassen. Die von uns in der Problemanalyse (siehe 1.1) dargestellten Möglichkeiten wären mit diesem Konzept durchaus zu realisieren, auch wenn bei der Überwachung eines Gebietes weniger eine Kreisform als eher ein langgestreckter Gürtel abgedeckt werden soll. Es steht zu vermuten, dass auch das Militär Verwendungszwecke für eine solche Plattform finden kann – und damit meinen wir nicht die Möglichkeit, entmilitarisierte Zonen überwachen zu können. Wahrscheinlich würden daraus dann fahrende Bomben oder Minen entstehen.

4.2 Bewertung der Einschränkungen

Die getroffenen Einschränkungen sollen lediglich bestimmte Bedingungen soweit vereinfachen, dass sie für uns handhabbar werden. Die Entfernung zwischen den Geräten lässt sich auch anders als über die Dämpfung des WLAN ermitteln. Dies würde aber weitere ALU-Funktionen wie Multiplikation und Winkelfunktionen (*sin*, *cos*, etc.) erfordern, um die GPS-Koordinaten dafür sinnvoll nutzen zu können. Die Wabenförmigkeit des Netzwerks kann sich bei bestimmten Umgebungsbedingungen verändern. In dem Moment funktioniert unsere Abstandsmessung per WLAN nicht mehr und man müsste zu anderen Methoden greifen.

4.3 Sicherheits- bzw. Fehlerbetrachtungen

Abgesehen von einem möglichen Diebstahl ist das WLAN das einzig mögliche Sicherheitsrisiko. Dies müsste in den Modulen sicher implementiert sein. Dies zu überwachen kann aber nicht unsere Aufgabe sein.

Einen Fehler stellt die fehlende Kollisionserkennung dar. In einem Praxiseinsatz dürfte den Geräten also erstmal nichts im Weg stehen.

4.4 Zeitkritische Abläufe

Unsere Geräte bewegen sich mit einer Geschwindigkeit von ca. 50 cm/s. Die Abweichung des GPS in Mitteleuropa beträgt ca. 3m bei der von uns verwendeten Genauigkeit von einer Zehntelbogensekunde. Daher ist eine genügend hohe Taktfrequenz für Bus und Prozessor notwendig, um ständig mit aktuellen Messwerten arbeiten zu können. Diese Taktfrequenz kann von handelsüblichen Embedded-Prozessoren zum jetzigen Zeitpunkt ohne Probleme erreicht werden. Schwierigkeiten könnten sich nur dann ergeben, wenn das GPS-Modul des Kontakt zu den Satelliten verliert und dann mit einer Antwort auf die von uns gestellten Daten-Anfragen so lange warten will, bis der Kontakt wieder hergestellt ist.

Die Geschwindigkeit von Paketen im WLAN (z. B. die Round-Trip-Time) ist für uns nicht interessant, da wir nur die Empfangsstärke messen. Genügend WLAN-Verkehr vorausgesetzt, lässt sich die Signalstärke in sehr kurzen Abständen aktualisieren.

Zeitkritische Abläufe spielen in dem Konzept sonst keine Rolle.

4.5 Darstellung von Testmöglichkeiten

Eine naheliegende Testmöglichkeit wäre z. B. das Aufspannen eines WLAN auf der Wiese vor dem Motorenprüfstand.

4.6 Mögliche Erweiterungen

Für einen Praxiseinsatz ist es auf jeden Fall erforderlich, mit Empfangsschwankungen im WLAN umgehen zu können. Dies schließt das Aufbauen eines Netzwerkes in unebenem Gelände mit ein, da hier durch Hindernisse die Verbindung zwischen den Geräten unterbrochen werden kann.

Weiterhin müsste ein Weg gefunden werden, wie der Ausfall eines Gerätes im Netzwerk kompensiert werden kann.

Für die Etablierung eines Netzwerks müsste eine Benutzerschnittstelle entworfen werden, die eine einfache Übergabe von GPS-Koordinaten an alle Geräte ermöglicht. Das existierende WLAN-Modul würde sich dafür anbieten, diese Daten an alle Geräte zu verteilen.

Gerade für den Ausseneinsatz wäre auch zu überlegen, eine Diebstahlsicherung zu implementieren, da die Geräte mit ihrer Ausstattung sicher ein begehrtes Sammlerstück darstellen.

Kapitel 5

Literatur

Vorlesungsskript „Technische Informatik 2“

GPS-Datenstruktur NMEA

http://www.kowoma.de/gps/zusatzerklaerungen/NMEA.htm

Signaldämpfung bei WLAN

http://www.steckerprofi.com/wl-yagi2.htm