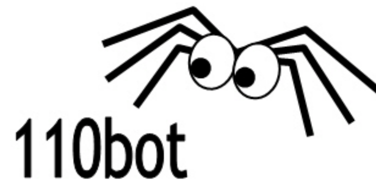
Fakultät II - Department für Informatik



Projektgruppe Smart Systems

Insektengruppe

Endbericht

Anton Abaschin, Carsten Kellermann, Henning Kleen, Martin Kummer,
Gerold Mauson, Dimitri Nijasow, Jann Poppinga, Andreas Prüllage,
Dietrich Schuckmann, Michael Taphorn

7. April 2005

Inhaltsverzeichnis

1 Einleitung **4**
1.1 Allgemeine Beschreibung einer Projektgruppe 4
1.2 Die Projektgruppe Smart Systems 4

2 Aufgabenstellung **5**

3 Arbeitsumgebung **6**
3.1 Universitätswerkstätten 6
3.2 Verwendetes Werkzeug im OFFIS 6

4 Dokumentation **7**
4.1 Mechanische Plattform 7
4.1.1 Allgemein 7
4.1.2 Entwurf 7
4.1.3 Prototyp 8
4.1.4 Hinweise zur Montage, Reparatur und Erweiterung 8
4.2 Platine 9
4.3 Roboter-Software 9
4.3.1 FPGA-Implementierung 9
4.3.2 Kommunikation 13
4.3.3 Bildkompression 17
4.3.4 Aktoren 22
4.3.5 Sensoren 34
4.4 Host 38
4.4.1 Karte und Navigation 38
4.4.2 Wegefindung 40
4.4.3 Sensorwerte 51
4.4.4 Fotofenster 51
4.4.5 Kommunikation 52

5 Projektablauf **54**
5.1 Regelmäßiges Treffen 54
5.2 Projektplan 55
5.2.1 Anforderungsdefinition 55
5.2.2 Entwurf 57
5.2.3 Implementierung 58
5.2.4 Zwischenbericht 58
5.2.5 Integration und Test 58
5.2.6 Test mit 4WD 59
5.2.7 Zusammenfassung 59
5.3 Meilensteinentwicklung 61
5.3.1 Ausgewählte Meilensteine 61

5.3.2 Plattform und Platine . . . 62
5.3.3 Sensoren . . . 64
5.3.4 Aktoren . . . 66
5.3.5 Kommunikation . . . 67
5.3.6 Host . . . 67

6 Bewertung des Systems **68**
6.1 Roboter . . . 68
6.1.1 Erfolge . . . 68
6.1.2 Bekannte Probleme . . . 68
6.2 Host . . . 69
6.2.1 Erfolge . . . 69
6.2.2 Bekannte Probleme . . . 69

7 Erfahrungssammlung **70**
7.1 Anton Abaschin . . . 70
7.2 Carsten Kellermann . . . 70
7.3 Henning Kleen . . . 71
7.4 Martin Kummer . . . 72
7.5 Gerold Mauson . . . 73
7.6 Dimitri Nijasow . . . 74
7.7 Jann Poppinga . . . 75
7.8 Andreas Prüllage . . . 76
7.9 Dietrich Schuckmann . . . 76
7.10 Michael Taphorn . . . 77

8 Fazit **79**

1 Einleitung

Bei dem vorliegenden Dokument handelt es sich um den Endbericht der Projektgruppe Smart Systems - Insektenroboter. Zunächst soll eine kurze Beschreibung einer Projektgruppe (im Folgenden PG) im Allgemeinen und der PG Smart Systems im Speziellen erfolgen.

1.1 Allgemeine Beschreibung einer Projektgruppe

Im Studiengang Informatik an der Carl von Ossietzky Universität Oldenburg ist es im Rahmen des Hauptstudiums verbindlich vorgesehen, dass jeder Student an einer Projektgruppe teilnimmt. Eine Projektgruppe besteht in der Regel aus sechs bis zwölf Teilnehmern, die über einen Zeitraum von zwei Semestern eine größere Aufgabe aus dem Bereich der Informatik bearbeiten. Im Rahmen der Projektgruppe finden wöchentlich moderierte Treffen statt, in denen Fortschritte und Probleme diskutiert werden. Ziel der PG ist es, die Teamarbeit bei der gemeinsamen Lösung einer nicht trivialen Aufgabe zu üben und somit zusätzliche Kompetenzen auf dem Gebiet der Teamfähigkeit zu erlangen. Für die genaue Ausgestaltung des Ablaufs einer PG existieren viele Freiheitsgrade, so dass der Ablauf durch die Betreuer individuell an die Bedürfnisse der Projektgruppe angepasst werden kann.

1.2 Die Projektgruppe Smart Systems

Nach der Vorstellung der PG Smart Systems im Rahmen des PG-Tages im Wintersemester 2003 / 2004 meldeten sich 22 Studenten für die Teilnahme an der Projektgruppe an. Die große Zahl der Anmeldungen machte eine Aufteilung in zwei einzelne Projektgruppen nötig. Beide Gruppen sollten jedoch eng zusammenarbeiten. Die Ursprüngliche Idee einen Roboter zur Fernerkundung ähnlich den zum Beispiel für die Erkundung des Mars eingesetzten Roboters, wurde zu Beginn der Projektgruppe abgewandelt um ein Szenario zu schaffen in dem die beiden Gruppen eine Gemeinsame Aufgabe bearbeiten können. Eine Gruppe sollte einen Roboter bauen, der sich konventionell auf Rädern fortbewegt und den zweiten Roboter transportiert. Die andere Gruppe sollte einen kleineren Roboter entwerfen, der zur Fortbewegung sechs Beine besitzt, wie ein Insekt.
Zehn der insgesamt 22 Teilnehmer bildeten die Insektengruppe, die sich im Verlauf der Projektgruppe mit der Entwicklung des Insektenroboters befassten.

In den weiteren Teilen dieses Dokuments soll ein Überblick über das erstellte System, sowie unsere Erfahrungen gegeben werden.

2 Aufgabenstellung

Die Aufgabe dieser Projektgruppe war es einen insektenartigen Roboter zu entwerfen und zu implementieren. Der Roboter sollte so entworfen werden, dass er möglichst universell einsetzbar ist und einfach um andere Funktionen erweitert werden kann. Als Beispielaufgabe wurde ein Szenario vorgegeben, an dem wir uns hauptsächlich orientiert haben:

Der Insektenroboter soll auf einem ungepflasterten Parkplatz herumlaufen und dabei ferngesteuert Fotos von den Kennzeichen bestimmter Autos schießen. Der Benutzer sitzt dabei irgendwo an einem Rechner, auf dem ein Programm läuft, mit dessen GUI er den Roboter steuert. Der Insektenroboter arbeitet dabei mit einem anderen, vierradgetriebenen Roboter zusammen. Vor allem erstellen die beiden Roboter zusammen eine Karte der Umgebung, an Hand derer sich der Insektenroboter orientiert und Ziele ansteuert. Der vierradgetriebene Roboter wurde von einen zweiten Untergruppe konstruiert.

Während dieses Projektes wurde nur ein Insektenroboter gebaut, aber das System sollte so ausgelegt werden, dass später ohne großen Aufwand mehrere Insektenroboter betrieben können.

3 Arbeitsumgebung

3.1 Universitätswerkstätten

Für den Bau des Prototyps der mechanischen Plattform haben wir nach Absprache mit dem Meister der Metallwerkstatt der Uni Herrn Helms, seine Werkstatt benutzt. Die Werkstatt befindet sich im AVZ der Universität. Wir durften dort alles an Werkzeug benutzen, außer etwas komplexeren Maschinen wie Drehbänken und Fräsmaschinen, die wir auch nicht benötigten. Herr Helms gab uns Einweisung für die Bohrmaschine und hat uns auch sonst mit Rat und Tat unterstützt. Die Werkstatt konnten wir in der vorlesungsfreien Zeit nutzen, später kamen wir noch um die Einsatzplattform zusammenzubauen während der Zeit der 'offenen Werkstatt'. In der Metallwerkstatt konnten wir alles an Werkzeug finden, was wir für den Bau des Prototyps, sowie für die Montage der Einsatzplattform benötigten. Für das Material muss man allerdings selbst sorgen.

Die Einsatzplattform ließen wir in der Uni-Werkstatt in Wechloy fertigen. Diese Werkstatt verfügt über CNC-Maschinen und macht Auftragsfertigungen für die Universität. Wir überließen ihnen die Skizzen und CAD-Dateien für die Einzelteile und in ein Paar Monaten erhielten wir die fertigen Teile der Plattform. Mehrere Wochen Wartezeit sollte man schon einplanen, falls man in dieser Werkstatt etwas fertigen lässt.

Das meiste Material für die Plattform haben wir uns über die Werkstattlager in Wechloy besorgt. Es gibt dort Metall- und Elektroniklager. Falls man im dortigen Assortiment nicht fündig wird, kann man auch viele Sachen über sie nachbestellen.

3.2 Verwendetes Werkzeug im OFFIS

- Oszilloskop
- SMD-Lötstation
- Digitalmultimeter
- Altera-Development Kit mit Software
- CADSTAR 6.0 (Platinenlayout)
- Linear Technology Switcher CAD III (Simulation)

4 Dokumentation

4.1 Mechanische Plattform

4.1.1 Allgemein

Die mechanische Plattform dient als Träger für den gesamten Roboteraufbau: elektronische Platine, Aktoren, Sensoren, Fotokamera, sowie die Spannungsversorgung durch Batterien. Als Material für die Fertigung der Plattform haben wir 2mm-starkes Lexan gewählt. Diese Wahl hatte sich als eine recht gute Entscheidung erwiesen: dieses Material ist sehr einfach zu verarbeiten, leicht, robust, transparent und preisgünstig. Insgesamt setzt sich der 110-Bot aus 66 Lexan-Einzelteilen zusammen. Für die Montage setzten wir noch Schrauben und Gewindestangen (meist M2), Abstandbolzen, sowie Kunststoffkleber ein. Es war uns wichtig, dass die Plattform mit möglichst wenig Aufwand herzustellen war und möglichst flexibel für spätere Änderungen bzw. Erweiterungen sein sollte.

Insgesamt lässt sich sagen, dass wir unsere Vorgaben bei der Plattform relativ preisgünstig erreicht haben. Lediglich die Wahl der Aktoren hat sich als ziemlich unglücklich herausgestellt.

4.1.2 Entwurf

Bei dem Entwurf der mechanischen Plattform haben wir uns nach den ähnlichen, schon vorhandenen Projekten[1]orientiert und viele Ideen daraus übernommen. Die gesamte Mechanik wurde mit Hilfe eines CAD-Tools entworfen. Das hatte sich als sehr hilfreich erwiesen, da man bereits sehr früh auf mögliche Probleme aufmerksam wurde. Außerdem konnten die Entwurfsdateien direkt für die Fertigung auf CNC-Maschinen in der Werkstatt genutzt werden. Man muss allerdings auch festhalten, dass der Aufwand für die Einarbeitung in solche CAD-Programme nicht zu unterschätzen ist. Als Anhaltspunkt für die Abmessungen der fertigen Einzelteilen, insbesondere der Beinteile, dienten die Servos. Leider wendete sich später dieser Umstand zu einem Nachteil (siehe auch Kap. 6.1.2): die Plattform lässt sich schwer auf Servos, die andere Abmessungen haben umstellen. Es müssten vor allem die Einzelteile der Beine neu entworfen und gefertigt werden. Eventuell lassen sich die Teile auch einfach auf die benötigten Abmessungen skalieren.

Im „Torso“ des Insekts ist ein Batteriefach untergebracht. In der Entwurfsphase waren maximal 6 Batterien eingeplant und deswegen auch war dort eine andere Art der Befestigung der Batterien vorgesehen. Später kam es zu einer Entscheidung 8 Batterien einzusetzen mit dem entsprechend anderen Batterienhalter. Bei dieser Umstellung gab es keine größeren Probleme.

[1]Z.B.: http://www.lynxmotion.com

4.1.3 Prototyp

Bevor die Einzelteile in die Fertigung auf CNC-Maschinen gingen, hat sich unsere PG entschlossen parallel dazu einen Prototypen in der Lehrwerkstatt der Universität „von Hand“ zu fertigen. Dies sollte wiederum der Möglichkeit dienen den Entwurf möglichst früh auf seine Tauglichkeit zu testen, eventuell auftretende Probleme erkennen, beseitigen und Informationen über die Änderungen an die Werkstatt weiterleiten, um Minimum an Zeit bei der Fertigung der Einsatzplattform zu verlieren. Diese Entscheidung hat sich ebenfalls als nützlich erwiesen. Auch wenn wir keine gravierenden Probleme bei den in Auftrag gegebenen Entwurfsskizzen feststellen konnten, so haben wir bereits einige Erfahrung über die Montagemöglichkeiten der Einzelteile zu einer fertigen Plattform gesammelt. Außerdem haben wir die Eigenschaften und Möglichkeiten der Plattform sehen und testen können. Nachdem der Prototyp komplett fertig war, konnte er schon für die Weiterentwicklung des Roboters mit nur einigen wenigen Einschränkungen genutzt werden, bis die Einzelteile aus der CNC-Fertigung eintrafen. Dieser Umstand ersparte der Gruppe mehrere Tage Zeit. Ein weiterer, nicht zu verachtender Vorteil dieser Prototypenerstellung war ein relativ großer Motivationsschub für die gesamte Projektgruppe: endlich sah man, dass sich irgendetwas bewegt.

Nach dem die CNC-gefertigten Teile verfügbar waren, stellten wir die Plattform auf diese um, da deren Qualität doch um einiges besser als der von Hand ausgesägten Prototyp-Teilen war. Einige Einzelteile aus dem Prototypen können weiterhin als Ersatzteile für den Roboter dienen.

4.1.4 Hinweise zur Montage, Reparatur und Erweiterung

Im Folgenden möchten wir einige Hinweise und Tipps für eventuelle Reparatur- und Erweiterungsarbeiten an den mechanischen Teilen des Roboters geben.

- Die Servos **XS-6G** von Jamara, die wir einsetzen, haben sich leider als nicht besonders zuverlässig erwiesen. Für späteres Arbeiten mit dem 110-Bot empfehlen wir andere Servomotoren einzusetzen, wenn möglich, mit einem Getriebe aus Metall.
- Die Servos, die für die Bewegung nach vorne/hinten verantwortlich sind, können zwischen den beiden Beinplatten geklemmt werden. Alternativ kann man sie zur besseren Stabilität auch mit ein wenig Kunststoffkleber festkleben. Allerdings wird es dann schwieriger solche Servos auszutauschen.
- Das Schmieren der Achsen und Gelenke kann die Lebensdauer der Servos etwas verlängern.
- Die Muttern an den beweglichen Gelenken sollten mit etwas Schraubenlack festgemacht werden, damit sie sich nicht während des Laufens lösen. Alternativ kann man auch etwas Nagellack nehmen.
- Platine, Kamera und Ultraschallsensor werden mit Hilfe metallischer Abstandbolzen auf der Plattform befestigt. Es sollte darauf geachtet werden, dass kein Kontakt mit der Elektronik auf der Platine bzw. Sensoren entsteht.

- Der Kompass sollte mit etwas Abstand zur Platine befestigt werden, damit er zuverlässige Werte liefern kann.
- Lexan ist flexibel und sehr Stabil. Man kann dieses Material um weit mehr als 90° biegen ohne es erwärmen zu müssen und die Form bleibt danach erhalten. Auf diese Weise lassen sich ohne großen Aufwand verschiedene Teile herstellen. Der Winkel der Kamerabefestigung kann problemlos auch durch biegen verändert werden.
- Um die Beine des Roboters rutschfester zu machen, haben wir ihm die Füße aus Silicon gefertigt. Füße aus Radiergummis waren im Vergleich zum Silicon weniger verschleißfest.
- Falls eine Ersatzteilherstellung oder weitere Roboter benötigt werden, können die Skizzen aus dem Entwurfsdokument, sowie die zur Verfügung stehende CAD-Dateien genutzt werden. Die Endversion der mechanischen Plattform unterscheidet sich im wesentlichen nicht von der im Entwurf.

4.2 Platine

Eine Dokumentation der Platine befindet sich in einem gesonderten Dokument. Aus Gründen der Übersichtlichkeit wurde auf eine Wiedergabe des kompletten Dokuments an dieser Stelle verzichtet.

4.3 Roboter-Software

4.3.1 FPGA-Implementierung

I2C-Bus Dieser Bus kann sowohl für die Ansteuerung des Ultraschallsensors und Kompassensors (in diesem Projekt nicht eingesetzt) als auch für die Konfiguration der Kamera zuständig. Bei der Kamera wird zwar der SCCB-Bus von Philibs benutzt der bei eingeschlateter Steuerleitung wie der I2C-Bus funktioniert.
Er basiert auf einer 5 Volt Logik und ist bidirektional. Für eine Nichtbelegegung/logische „1“ schaltet die jeweilige Seiten auf Hochohmig (Z). Über die Belegung der Leitung entscheidet ein Master (Bei uns der I2C-Controller). Er spricht die einzelnen Module (Slaves) über eine auf dem I2C-Buss gesendetet Adresse an. Genaueres zum I2C-Busprotokoll befindet sich im Anhang.

c-Ansteuerung des i2c-Busses In der insekt.c werden die pios, die mit dem i2c-Modul verbunden sind, wie folgt angesprochen. Hierbei wird der cmd-methode ein Integer mit dem gewünschten Befehl (start, stop, read, write ack und data(8-bit)) übergeben. Die dafür notwendigen Konstanten sind unter ihrem jeweiligen Namen in der insekt.c vorhanden. Das new_cmd_bit wird automatisch von der cmd-Methode richtig gesetzt. Für die Anstossung einen write-Befehls über den i2c-Bus müsste man beispielsweise cmd(start + write + data(200)) setzten (Dies entspricht z.b. 2(2te Bit) +16

(4te Bit) + 64*200 (bit 5-12)) . Die data-methode kontrolliert, ob sich der Wert von data innerhalb des zulässigen Wertebereichs (8-bit) befindet und multipiziert diesen mit 64 (für die Bits 5-12). In der cmd-methode werden die werte dann als 13 Bit wert (Das 0te Bit ist das new_cmd_bit das von der cmd-methode automatisch gesetzt wird). Die Ansteuerung der pios erfolgt über na_data_out_i2c (Adresse der i2c-pios) die über np_piodata beschrieben werden. Ein solcher Befehlt sieht in C folgendermassen aus : **na_data_out_i2c->np_piodata = x + 1;** .

I2C-Modul Dieses Modul ist in unserer Quartussoftware für die Ansteuerung der I2C-Busses zuständig. Es besitzt die Eingänge : I2C_clock (daran wird die Systemclock angeschlossen), reset (aktive high) und Nreset (aktive low).
new_cmd_i2c (falls man einen neuen Befehl an den i2c-Bus übergeben will) und der Eingang data_i2c enthalten die Ansteuerung des i2c-Controllers und werden direkt aus C angesteuert :

- Bit 0 : start (ob die i2c Übertragung beginnen soll)
- Bit 1 : stop (ob die i2c Übertragung beendet ist)
- Bit 2 : read (ob vom i2c gelesen werden soll)
- Bit 3 : write (ob auf den i2c geschrieben werden soll)
- Bit 4 : ack (ob ein acknoledge angefodert werden soll)
- Bit 12-5 : data_i2c (Die daten die (beim write) auf den i2c geschrieben werden sollen)

Innerhalb dieses Moduls ergibt sich folgendes Bild :

zentrale Elemente hierbei sind die in Quartus erstellten fifos und der i2c-Controller von open corse . Die lpm_fifo_0 ist vor und lpm_fifo_1 hinter den i2c_contoller gesschaltet.Beim lpm_fifo_0 werden durch das modul cbw_interface das einlesen gesteuert. Die Kommunikation mit dem i2c_contoller übernimmt das modul cbr_interface. Bei lpm_fifo_1 ist das modul dbw_interface für das ein- und dbr_interface für das auslesen zuständig.

cbr_interface Speichert einen neuen Befehl in lpm_fifo_0, wenn sich das new_cmd Bit ändert.

cbw_interface Die Eingänge des moduls setzten sich wie folgt zusammen:

- clk die niosclock für die synchronisation des moduls
- reset aktive high reset

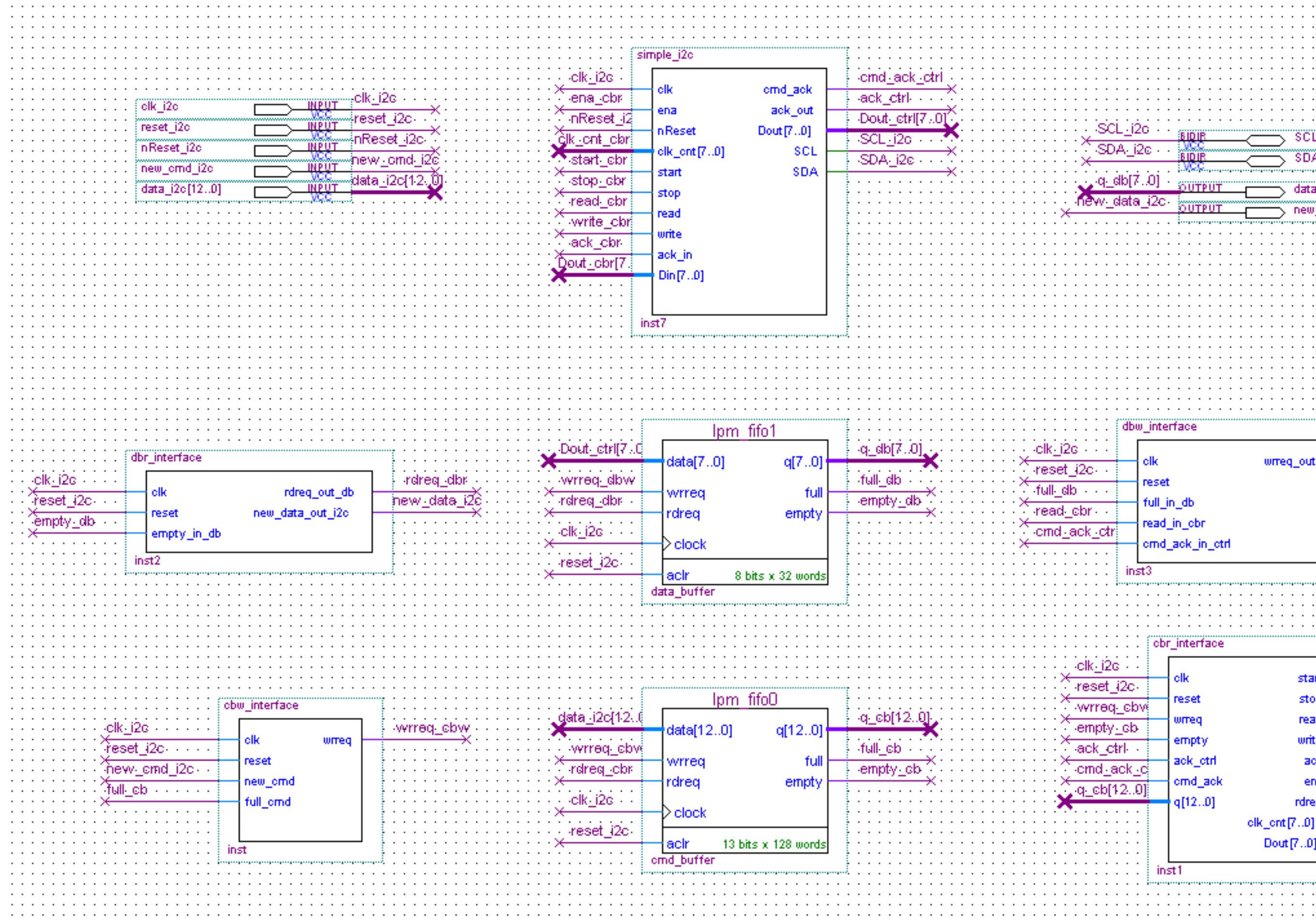


Abbildung 1: Interner Aufbau des angeschlossenen I2C-Moduls

- wrreq nicht mehr benutzt
- empty zeigt an ob der fifo leer ist
- ack_ctrl nicht mehr benutzt
- cmd_ack wird vom i2c_controller gesetzt, falls ein neuer Befehlabgearbeitet werden kann
- q Datenausgang vom Fifo der in die Signale start, stop, read, write, ack und data für i2c_controller übersetzt wird

Das Modul enthält einen Zustandsautomaten der wie folgt aussieht :

bei einem gesetzten Reset wird sofort in init gesprungen und alles abgebrochen. Die Zustände bewirken folgendes :

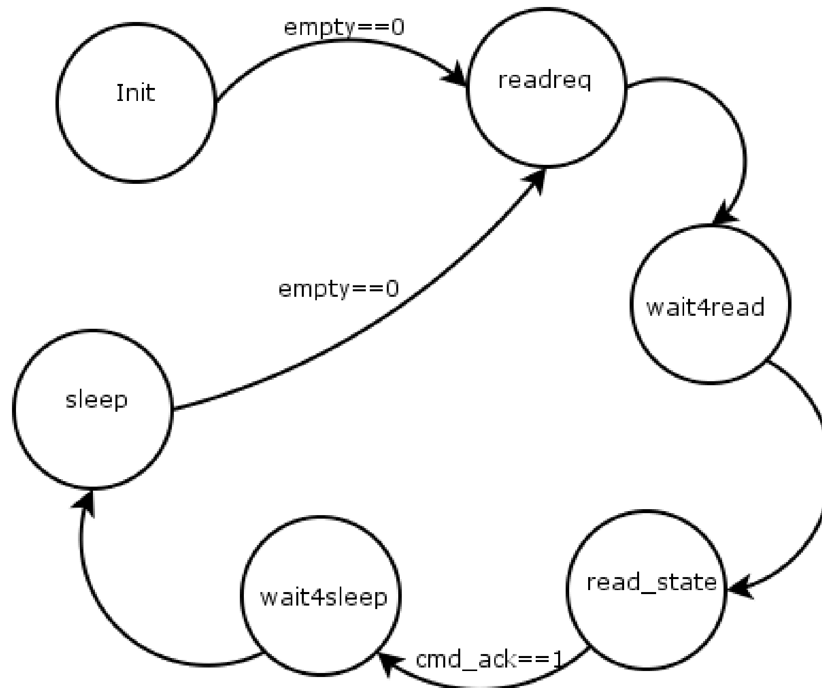


Abbildung 2: Zustandsautomat für die Ansteuerung des i2c-moduls

1. init setzt alle eingänge des i2c-controllers auf 0 bis auf enable (schaltet den i2c_contoller aus)
2. readreq setzt den Ausgang readreq auf 1 um neue werte zu erhalten
3. wait4read setzt den Ausgang readreq wieder auf 0 damit keine weiteren Befehle gelesen werden
4. read_state setzt die Eingänge des i2c-controllers auf die eingelesenen Werte. Es wird solange in read_state verweilt bis der i2c_controller mit der letzten aktion fertig ist . Danach werden alle Datenausgänge zum i2c-contoller auf 0 gesetzt und nach wait_for_sleep gewechselt.
5. wait_for_sleep wechselt automatisch nach einem takt in sleep
6. sleep überprüft ständig, ob sich ein Befehl im fifo befindet, falls ja wird nach readreq gewechselt.

Der clk_cnt Ausgang bestimmt die Verlangsamung der nios-clock. Er ist in unserem Fall auf 199 eingestellt, damit die i2c_clock eine Geschwindigkeit von 100 kHz erreicht. Der readrq Augang legt fest wann aus dem Fifo ausgelesen wird.

I2C-Controller Es wird für den I2C-Bus ein Hardwarecontroller von Open Corse benutzt, der in VHDL beschrieben ist. Es verlangsamt die Geschwindigkeit der niosclock

auf die Geschwindigkeit des I2C-busses. Die Einstellung der Geschwindigkeit der I2C Clock erfolgt über den clock_cnt Eingang. Alle Kommunikation mit dem i2c_bus ist hier realisiert.

Der i2c-controller wurde aufgrund der Pegelwandler verändert, so dass er am Anfang jedes I2C-Taktes für einen nios-takt statt auf Z auf 1 gezogen wird, um die Leitung die ansonsten auf 0 bleiben würde wieder auf high zu ziehen und somit eine Kommunikation zu ermöglichen.

dbw-interface Dieses Modul wartet auf den Abschluss eines Lesezugriffs des i2c-controllers und liesst speichert nach dessen Abschluss die Daten am Ausgang des i2c-controllers im fifo.

dbr-interface Falls Daten in das fifo geschrieben wurden (Das fifo nicht mehr leer ist) liesst das modul die Werte aus dem fifo aus und löst bei am nios-prozessor einen Interrupt aus, woraufhin der Prozessor die Werte einliesst.

funktionierende Simulation des i2c-Busses Hier wurde ein Lesezugriff simuliert. Der i2c-cotroller bekommt 4 befehle (er startet die Übertragung, schreibt die Adresse des gewünschten Geräts auf den Bus, danach schreibt er mit wiederholtem Start die Adresse des gewünschten Registers auf den Bus und liesst beim Stoppen der Übertragung die Antwort des Sensors. Die zwischenzeitlichen undefinierten Zustände von SDA-result durch die nicht gut darzustellende Bidirektionalität der Leitung und zeitweise Abschlatung des i^2c-Busses.

4.3.2 Kommunikation

Allgemein
Der Roboter und der Host-Computer kommunizieren via Bluetooth. Auf dem Roboter wird das Bluetooth-Modul über einen UART angesteuert, was den Vorteil hat, dass keine Bluetooth-Software auf dem FPGA implementiert werden musste. Dies erleichterte uns auch die Arbeit, da wir im Vorfeld mit einem normalen 0-Modem-Kabel testen konnten.

Bluetooth
Wir haben das Bluetooth-Modul (BTM) „WRAP THOR 2022-1-B2B“ der Firma BlueGiga verwendet. Es eignet sich sehr gut für unseren Einsatzfall, da es relativ geringe Maße (40x20mm) hat und keine externe Antenne benötigt. Die Reichweite beträgt 100m, Tests haben allerdings ergeben, dass dies nicht unbedingt immer der Fall ist, wenn z.B. Autos zwischen Host und Roboter stehen.
Das BTM hat zwei Modi. Zum einen den Kommandomodus in dem es Konfiguriert werden kann und zum anderen den Datenmodus in dem es sich dann wie ein 0-Modem-Kabel verhält. Die Verbindung wird vom Host initiiert. Erkennt das BTM eine Verbindung, so wechselt es automatisch in den Datenmodus. Die maximale Geschwindigkeit

Abbildung 3: Die Übertragung eines i^{2}c-Lesezugriffs

von Bluetooth konnten wir bedingt durch die Grenzen des UARTs leider nicht erreichen.

Implementierung

Die gesamte Kommunikation wurde in einer C-Datei implementiert: `kommunikation.c` bzw. `kommunikation.h` Alle anderen Teile mussten lediglich die Header-Datei einbinden und konnten somit den Methoden der Kommunikation benutzen. Der Roboter reagiert nur auf Kommandos vom Host, er selbst beginnt keine Aktivität ausgenommen eines Notstopps, wenn er einem Hindernis zu nahe gekommen ist, oder wenn kein Bodenkontakt festgestellt wird.

Zum Empfangen von Kommandos wird vom Hauptprogramm aus zyklisch die Methode `kommunikation_thread` aufgerufen. Diese Methode enthält eine Schleife, die mittels der Methode `receive_cmd` 10ms lang testet, ob ein Kommando empfangen wurde. Ist dies der Fall, so wird die Methode `cmd_handler` aufgerufen, die dann die entsprechenden Aktionen ausführt, wie z.B. Servo-Methoden ausführen, oder Meßwerte an den Host zurücksenden.

Das Senden und Empfangen von Zeichen ist gepuffert. Es wird ein Interrupt ausgelöst, sobald ein Zeichen über den UART empfangen wurde. Die Interrupt-Service-Routine

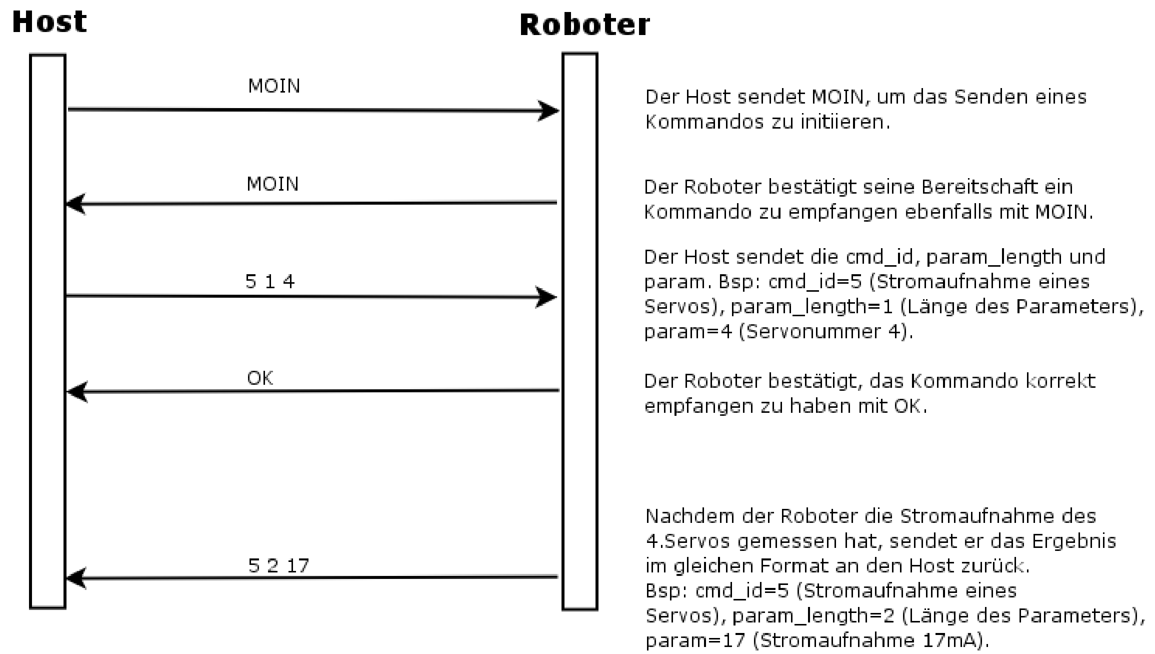
kopiert das empfangene Zeichen in einen Ringpuffer. Die Methode `receive_cmd` wertet diesen dann aus, wenn sie das nächste mal aufgerufen wird. Der Vorteil des Ringpuffers besteht darin, dass keine Zeichen verloren gehen können, auch wenn der Roboter gerade anderweitig beschäftigt ist. Gleiches gilt für das Senden von Zeichen.

Protokoll
Zur Kommunikation reicht bekanntlich nicht alleine das Senden von Zeichen, es muss zusätzlich ein Protokoll definiert werden. Unser primäres Ziel war es das Protokoll so einfach wie möglich zu halten und trotzdem den Anforderungen gerecht zu werden.
Ein Kommando besteht aus einem Kommandocode, zugehörigen Parametern und der Länge dieser Parameter. Dadurch ist kein Terminierungszeichen notwendig, welches dann in den Parametern nicht vorkommen dürfte. Ein Kommando sieht also folgendermaßen aus:

```
<cmd_id> <param_length> <param1> <param2> ... <paramN>
```

Um dieses Kommando herum haben wir einen einfachen Handshake implementiert. Der Verbindungsaufbau wird vom Host mit einem `MOIN` begonnen. Der Roboter antwortet ebenfalls mit einem `MOIN`, wodurch er die Bereitschaft zum Empfangen bekannt gibt. Dann wird das oben beschriebene Kommando gesendet. Hat der Roboter das Kommando inkl. der Parameter empfangen, beendet er die Verbindung mit `OK`.
Die Rückantwort, also z.B. das Senden von Meßwerten, oder die Bestätigung, dass die angeforderten Schritte ausgeführt wurden, hat dasselbe Format wie das empfangene Kommando selbst.
Sehr wichtig ist bei einem Handshake auch die Verwendung von Timeouts. Wir haben einen Timeout von 2s verwendet. Dadurch ist gewehrleistet, dass der Roboter nicht in einer Endlosschleife hängen bleibt, was das fatalste wäre das passieren könnte.

Beispiel: Handshake zwischen Host und Roboter.


Host
Roboter
MOIN
Der Host sendet MOIN, um das Senden eines Kommandos zu initiieren.
MOIN
Der Roboter bestätigt seine Bereitschaft ein Kommando zu empfangen ebenfalls mit MOIN.
5 1 4
Der Host sendet die cmd_id, param_length und param. Bsp: cmd_id=5 (Stromaufnahme eines Servos), param_length=1 (Länge des Parameters), param=4 (Servonummer 4).
OK
Der Roboter bestätigt, das Kommando korrekt empfangen zu haben mit OK.
5 2 17
Nachdem der Roboter die Stromaufnahme des 4.Servos gemessen hat, sendet er das Ergebnis im gleichen Format an den Host zurück.
Bsp: cmd_id=5 (Stromaufnahme eines Servos), param_length=2 (Länge des Parameters), param=17 (Stromaufnahme 17mA).

4.3.3 Bildkompression

Wie im vorigen Kapitel beschrieben steht uns durch den Einsatz eines UARTs nur eine sehr begrenzte Bandbreite zur Übertragung von Bildern zur Verfügung. Das Bild hat eine Auflösung von 352x288 Pixel, wobei ein Pixel einem Byte entspricht (256 Graustufen). Die Größe des Bildes beträgt also etwa 100kb. Wird der UART mit 115000bps betrieben, so dauert die Übertragung eines Bildes knapp 10 Sekunden. Dieser Wert war unserer Meinung nach viel zu groß und somit nicht akzeptabel.
Zwei Faktoren beeinflussen also die Übertragungsdauer. Zum einen die Geschwindigkeit und zum anderen die Größe des Bildes selbst. Da die Geschwindigkeit aus technischen Gründen nicht erhöht werden kann, haben wir uns entschlossen die Bilder zu komprimieren. Folgendes Verfahren ist zum Einsatz gekommen.

Run Length Encoding (RLE)
Die RLE ist schnell und vom Umfang her einfach zu implementieren. Sie beruht darauf, dass gleiche aufeinanderfolgende Grauwerte zu einem zusammengefasst werden. Gesendet wird der Grauwert dann nur einmal und dazu ein Faktor.

```
Original      : 3 3 3 3 4 4 4 9 9 9 9 9 8
RLE           : 4 3 3 4 5 9 1 8
Rückkodierung : 3 3 3 3 4 4 4 9 9 9 9 9 8
```

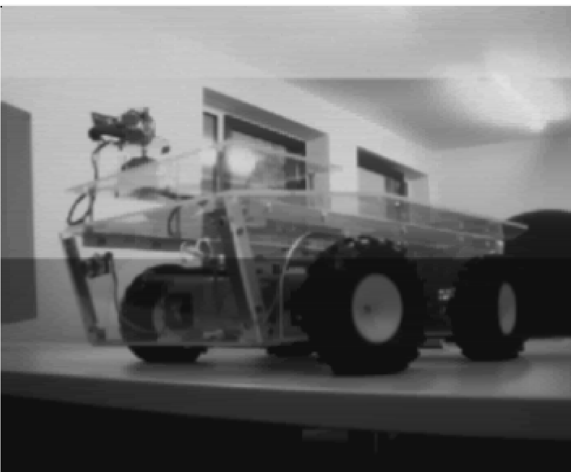
Der erste Wert ist als Faktor zu interpretieren und der zweite als Grauwert. Im Beispiel wurde eine Kompression um 38% erreicht. Das Rückkodierung, also das Bild welches auf dem Host angezeigt wird, entspricht exakt dem Original. Es kommt also zu keinem Qualitätsverlust.
Wie leicht ersichtlich ist, kann die Kodierung durch RLE aber auch zu einer Vergößerung der Datenmenge führen. Man stelle sich ein Bild vor, in dem keine gleichen benachbarten Pixel existieren. Jeder Grauwert würde dann mit dem Faktor 1 bewertet, was zu einer Verdopplung der Datenmenge führen würde. Bekanntermaßen ist dies bei Fotos aber nicht der Fall. In der Natur haben benachbarte Pixel auch immer ähnliche Grauwerte. Die Praxis hat gezeigt, dass diese Kodierung für unseren Einsatzfall sehr effektiv ist.

Erweiterung der RLE
Da das menschliche Auge sehr nahe beieinander liegende ähnliche Grauwerte nicht unterscheiden kann, haben wir den Algorithmus mit einer Unschärfe-Toleranz erweitert. Bei der oben beschriebenen Kodierung müssen benachbarte Pixel den exakt gleichen Grauwert haben, um zusammengefaßt zu werden. Die Idee war jetzt ähnliche Grauwerte, innerhalb eines Toleranzbereiches, zu einem Grauwert zusammenzufassen. Eine Toleranz von 2 Pixeln bedeutet also, dass ein Pixel welches maximal 2 Pixel vom Bezugswert abweicht als gleich betrachtet wird. Als Grauwert wird der arithmetische Mittelwert der zusammengefaßten Gruppe übertragen. Obiges Beispiel sieht dann folgendermaßen aus:

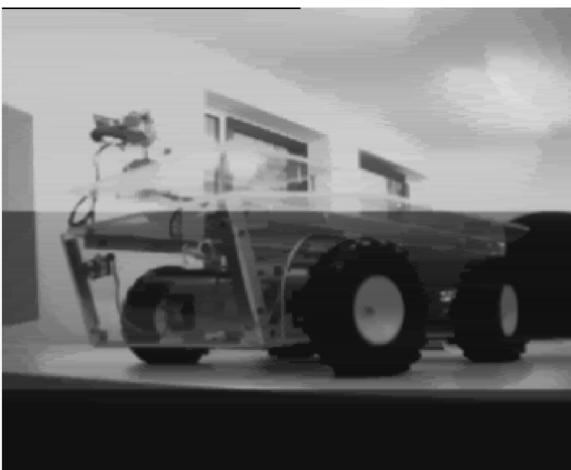
```
Original       : 3 3 3 3 4 4 4 9 9 9 9 9 8
RLE            : 7 3 6 9
Rückkodierung : 3 3 3 3 3 3 3 9 9 9 9 9 9
```

Hier wird eine Kompression um 69% erreicht. Die Rückkodierung unterscheidet sich vom Original. Die Toleranz bei der Kodierung ist also ein Maß für die Qualität des Bildes. Je größer die Toleranz, desto kleiner die Datenmenge und schlechter die Qualität.
Wie bereits erwähnt sind benachbarte Pixel in der Natur oft ähnlich und zusätzlich unterscheidet das Auge nahe beieinander liegende ähnliche Grauwerte nicht. Genau diese beiden Tatsachen haben wir uns in diesem erweiterten RLE-Algorithmus zunutze gemacht.
Der Toleranzwert ist also ein Kompromiss zwischen Datenmenge und Bildqualität. Bei der Auflösung von 256 Grauwerten erzielt eine Toleranz von 10 Pixeln im allgemeinen sehr gute Ergebnisse. Die Kompression liegt häufig zwischen 70 und 90%. Die Übertragung des Bildes dauert dann nur 2 oder 3 Sekunden.
Die Implementierung ist in `kommunikation.c` zu finden, Methode `pictureRLE()`.
Hier ein paar Beispiele, um einen visuellen Eindruck zu bekommen.
Anmerkung: Die großflächigen Helligkeitsunterschiede die zu sehen sind haben nichts mit der Kompression zu tun, sondern es handelt sich dabei um ein generelles Problem mit der Kamera.

Toleranz=0 Pixel, Kompression=0%

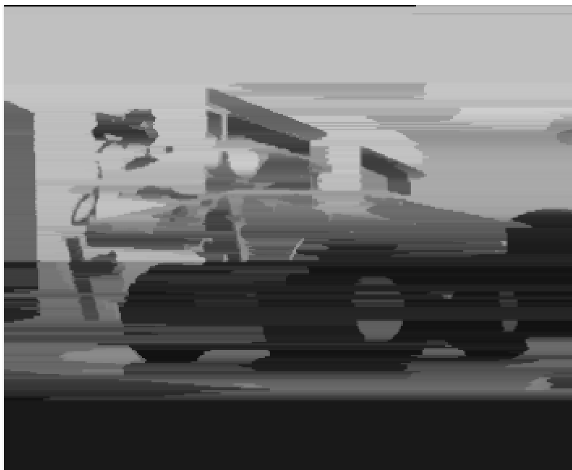


Toleranz=5 Pixel, Kompression=65%



Toleranz=10 Pixel, Kompression=81%

Toleranz=20 Pixel, Kompression=89%


Toleranz=50 Pixel, Kompression=95%

Zukunft
Eine weitere Idee war Bilder durch Auslassen von Pixeln zu komprimieren. Die Rückkodierung interpoliert die ausgelassenen Pixel dann z.B. linear oder eventuell auch mit einer anderen Interpolationsmethode. Die Schrittweite, also die Anzahl der auszulassenden Pixel, wäre dann ein Maß für die Qualität des Bildes. Eine Schrittweite von 2 bedeutet also, dass nur jedes zweite Pixel und jede zweite Zeile gesendet wird. Die Hostsoftware ergänzt dann die fehlenden Pixel und Zeilen.
Beispiel mit linearer Interpolation:

```
Original      : 3 3 3 4 7
                4 3 3 5 9
                5 3 5 5 8

Kodierung     : 3 3 7
                5 5 8

Rückkodierung : 3 3 3 5 7
                4 4 4 6 8
                5 5 5 7 8
```

Auch diese Idee basiert darauf, dass nebeinander liegende Pixel ähnlich sind. Hat ein Pixel den Wert 3 und das übernächste den Wert 5, so erscheint es mir wahrscheinlicher, dass das dazwischenliegende Pixel eher den Wert 4 hat und nicht z.B. 137. Mit dieser Annahme wäre eine lineare Interpolation zwischen den Pixeln vielleicht sinnvoll. Der Algorithmus ist bisher nur auf dem Roboter implementiert `kommunikation.c`, Methode `pictureInterpol()`.
Da wir mit der erweiterten RLE überraschend gute Ergebnisse erzielten, haben wir dieses Verfahren nicht weiter betrachtet. Für die Zukunft wäre es aber sicherlich interessant zu testen welche Bild-Qualitäten man damit erreichen kann.

4.3.4 Aktoren

Die gesamte Bewegung des Roboters wird von 13 Servos ausgeführt. Durch gezielte Ansteuerung der Servos wird folgende Funktionalität erreicht:

- Winkeleinstellung von Servos zwischen -90° und 90°
- Beinpositionierung horizontal und vertikal
- Vorwährts- und Rückwärtsgehen
- Drehen auf der Stelle
- Einstellen der Normalhöhe der Plattform
- Neigen der Plattform nach vorne, zurück und zur Seite

Winkeleinstellung von Servos Ein FPGA-Modul zur Ansteuerung von Servos ist im Nios integriert. Dieses Modul ermöglicht, über den Aufruf der Prozedur *servocontroller_set_servo_angle()* einen Wert für jedes Servo an die Pulsweitenmodulation zu übergeben. Dieser Wert ist 16-Bit breit und stellt eine Signalbreite zwischen 0 und 2.62 ms dar. Die eingesetzten Servos haben einen Stellbereich zwischen 0° und 180°. Dafür benötigen sie eine Impulsbreite zwischen 0.5 und 2.5 ms, dies entspricht den Werten 12500 und 62500.

Um eine Positionsveränderung beim Servo um 1 Grad zu bewirken, benötigt man eine Wertänderung von etwa 278. Die Prozedur *setServoAngleTenthDegree()* übernimmt die Umwandlung des übergebenen Winkels im Bereich von -90° bis +90° in einen Wert für die Pulsweitenmodulation und steuert das Servo an. Diese Prozedur erwartet einen Winkel in Zehntel Grad, also von -900 bis +900. 0 entspricht der Mittelstellung von Servos. Der Wert für die Pulsweitenmodulation beträgt dabei 37500.

Da innerhalb der Prozedur mit Integer-Werten gerechnet wird, wird für eine Winkeländerung von 1 Grad nicht mit 27.8, sondern mit 28 multipliziert. Das verursacht einen Fehler von 0.8%. Die Abweichung des erforderlichen Signals bei 90° würde damit einen Stellwinkel von 90.72° bewirken. Da die Stellgenauigkeit der Servos selbst mehrere Grad beträgt, ist diese Abweichung hinnehmbar.

Die Prozedur *setServoAngleTenthDegree()* wird nur intern verwendet. Für die Ansteuerung außerhalb der Datei *servo.c* steht die Prozedur *setServoAngle()* zur Verfügung. Die Entscheidung für interne Verwendung der Methode *setServoAngleTenthDegree()* wurde mit dem Zweck getroffen, die Stellgeschwindigkeit von Servos beeinflussen zu können. Vom Hersteller wird eine Stellgeschwindigkeit von 0.13 s pro 60° angegeben. Bei den ersten Gehversuchen hat es sich herausgestellt, dass dies zu schnell war, so dass Beine auf dem Boden durchgerutscht waren. Außerdem war es sinnvoll, die Plattform langsamer zu bewegen, um große Krafteinwirkungen auf Servos zu vermeiden. Eine Schrittweise Ansteuerung mit 1-Grad-Schritten hat nicht zum gewünschten Ergebnis geführt. Kleinere Zwischenschritte sollten möglich sein. Durch Einführen der Prozedur *setServoAngleTenthDegree()* ist die Bewegung beim Gehen flüssiger geworden.

Beinpositionierung Zur Beipositionierung stehen zwei Prozeduren zur Verfügung: *setLegHeight()* und *setLegDistance()*. Sie erwarten jeweils ein Parameter als Eingabe in mm. *setLegHeight()* positioniert das Bein in der Höhe, *setLegDistance()* in horizontaler Ablenkung. Die beiden Methoden sind im Gegensatz zu *setServoAngle()* keine direkten Methoden, d.h. die Position der Beine wird nicht sofort verändert. Diese Methoden stoßen die Bewegung der Beine nur an, die Kontrolle dieser Bewegung geschieht in anderen Prozeduren. Alle anderen Prozeduren zur Ansteuerung von Servos sind ebenfalls keine direkten Methoden.

Ein Aufruf der Prozedur *setLegHeight()* bzw. *setLegDistance()* ruft intern die Methode *setServoTargetAngleTenthDegree()* auf. Die letzte speichert in einer Variable den Zielwinkel. Das Ansteuern des zugehörigen Servos geschieht später in der Funktion *moving()*. Diese Funktion berechnet die Zeitspanne zwischen diesem und dem letzten Aufruf von *moving()*. Daraus berechnet sie Winkelaktualisierungen für Höhen- und Positionsservos. Entspricht der aktuelle Winkel eines Servos nicht dem gespeicherten Zielwinkel, wird das Servo um neu berechnete Winkelaktualisierung verstellt. Wird eines der 12 Beinservos in *moving()* verstellt, liefert die Funktion 1 zurück, sonst 0. Damit kann festgestellt werden, ob eine Teilbewegung abgeschlossen ist.

Die Funktion *moving()* wird von folgenden Methoden aufgerufen: *servo_thread()*, *stepping()*, *turning()* und *leaning()*. Die Hauptmethode ist *servo_thread()*. Hier wird nach Bewegungszuständen unterschieden und, falls eine Bewegung durchzuführen ist, eine der Methoden *moving()*, *stepping()*, *turning()* oder *leaning()* aufgerufen.

Die Umwandlung von Parameterwerten in mm in Winkel für Servos erfolgt in *setLegHeight()* und *setLegDistance()* durch Multiplikation mit einem Faktor. Das ist zwar nicht korrekt, für Winkel unter 10° jedoch ausreichend. Die Faktoren zum Berechnen des Winkels aus dem Abstand in mm wurden experimentell ermittelt und Betragen 4.2 für Höhenservos und 1 für Positionsservos. Auf genauere Berechnung der Winkel wurde verzichtet, da sie eine rechenintensive arcsin-Funktion benötigen, welche auf dem Nios, wie es sich später herausgestellt hat, nicht korrekt implementiert war. Zudem wurden diese Methoden im Laufe der Integration und des Systemtests kaum aufgerufen, da sie keinen praktischen Nutzen aufwiesen.

Gehen Zum Vor- und Rückwärtsgehen steht die Prozedur *step()* zur Verfügung. Als Parameter erwartet sie einen positiven Wert für Vorwärtsgehen oder eine negative Zahl für Rückwärtsgehen. Der Wert selbst gibt die Anzahl zurückzulegender Schritte an. Mit jedem Schritt wird eine Strecke von 2 cm zurückgelegt.
Die Geschwindigkeit der Bewegung kann über Methoden *setHeightDelayPerDegree()* und *setDistanceDelayPerDegree()* eingestellt werden. Als Parameter erwarten beide Prozeduren eine Zeitspanne in Millisekunden, innerhalb welcher eine Winkeländerung von 1° erfolgen soll. Die erste Methode stellt die Geschwindigkeit für Höhenservos ein, die zweite für Positionsservos. Default-Werte betragen 4 für Höhenservos und 15 für Positionsservos. Der Grund für unterschiedliche Geschwindigkeiten der Höhen- und Positionsservos liegt an unterschiedlicher Auswirkung gleicher Winkeländerung. Während schnelle Bewegung der Höhenservos keine Nachteile hat, rutschen bei großer

Geschwindigkeit der Positionsservos Beine auf dem Boden durch.

Im Entwurf wurde nur eine Geschwindigkeitseinstellung für alle Servos vorgesehen. Bei den Tests hat sich schnellere Ansteuerung von Höhenservos als vorteilhaft erwiesen. Denn so ließen sich lange Bewegungspausen beim Anheben und Absenken der Beine verkürzen.

Man sollte beachten, dass die Einstellzeit für 1° nicht unter 2 ms liegen sollte, denn das entspricht in etwa der Stellzeit von Servos.

Wie auch bei der Beinansteuerung löst *step()* die Bewegung nur aus. Die Kontrolle erfolgt in der Prozedur *stepping()*. Darin wird zunächst *moving()* aufgerufen. Liefert sie 0 (d.h. letzte Teilbewegung abgeschlossen), wird mit dem nächsten Teilschritt fortgesetzt. In den Teilschritten wird neue Position der Servos berechnet und mit Hilfe der Methode *setServoTargetAngleTenthDegree()* aktualisiert.

Bodenkontakt Nach jedem Absetzen der Beine wird überprüft, ob die gerade abgesenkten Beine einen Bodenkontakt aufweisen. Ist das nicht der Fall, werden sie weiter abgesenkt, bis jedes der drei Beine auf dem Boden steht, oder der maximale Winkel zur Bodensuche erreicht ist. Dieser ist durch die Konstante *MIN_HEIGHT_ANGLE-_WITHOUT_GROUND_CONTACT* gegeben und hat einen Wert in Zehntel Grad von -300 (Absenken bedeutet negativen Wert). Der Winkel für Zwischenschritte bei der Bodenkontaktsuche definiert die Konstante *SEARCH_GROUND_ANGLE*. Sie enthält ebenfalls einen Wert in Zehntel Grad und ist 40 gleichgesetzt.

muss explizit aktiviert werden

Wird beim Absetzen der Beine kein Bodenkontakt festgestellt, bleibt der Roboter stehen und teilt dem Host Notstop mit. Die Suche nach dem Bodenkontakt erfordert eine

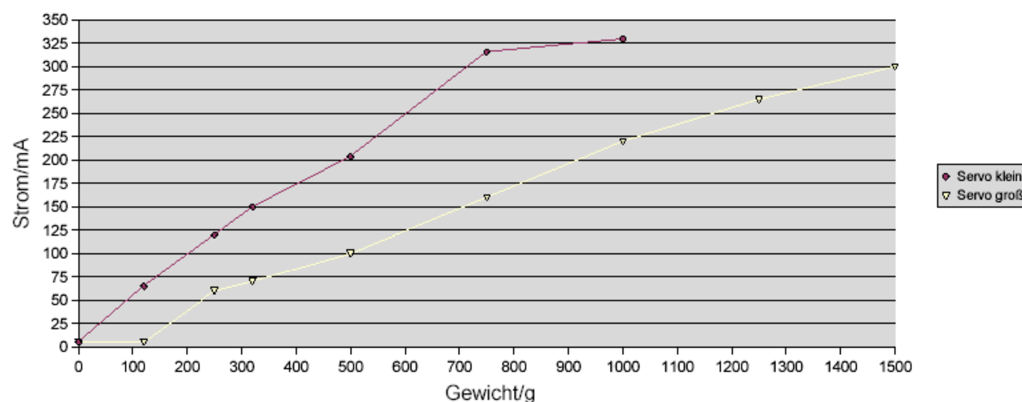


Abbildung 4: Stromaufnahme von Servos bei Belastung

gewisse Zeit. Das liegt daran, dass er von der Stromaufnahme der Servos abgeleitet wird. Nimmt ein Servo mehr Strom auf, als es unbelastet benötigt, wird auf den Bodenkontakt geschlossen. Das Diagramm 4 zeigt die Stromaufnahme abhängig von der Belastung. Dabei wurde das angegebene Gewicht in einem Abstand von 1 cm von der Achse des Servos angehängt. Die Kennlinie mit der Bezeichnung „Servo klein“ charakterisiert die eingesetzten Servos.

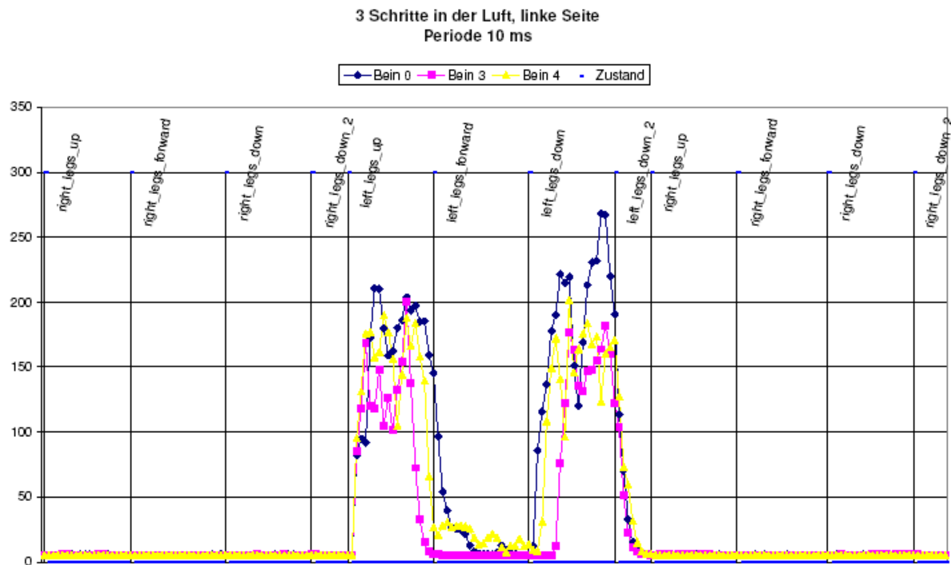


Abbildung 5: Stromaufnahme von Servos in der Luft, linke Seite

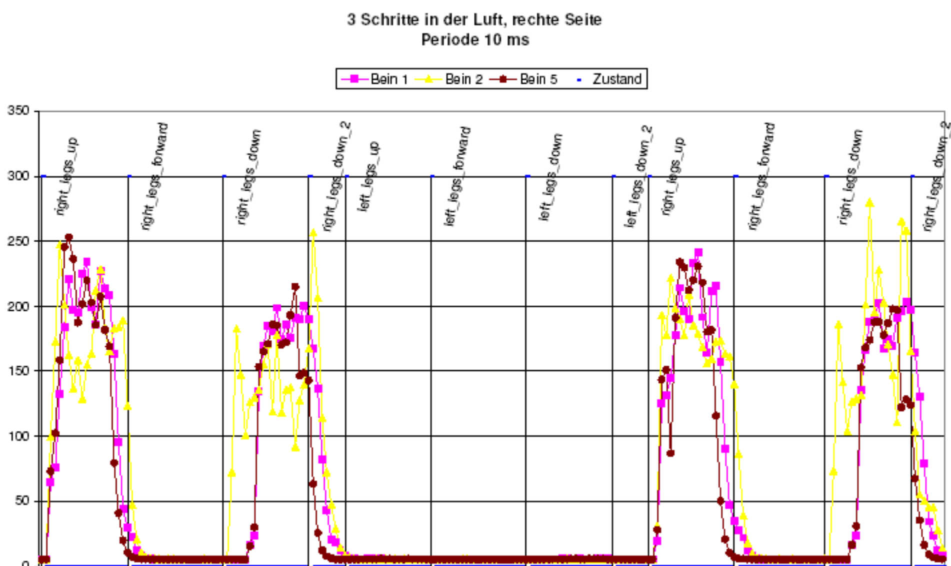


Abbildung 6: Stromaufnahme von Servos in der Luft, rechte Seite

Diagramme 5 bis 10 zeigen Stromaufnahme der Höhenservos beim Gehen. Die Abbildungen 5 und 6 zeigen Verlauf der Stromaufnahme-Kennlinien beim Ausführen des Laufalgorithmus ohne Bodenkontakt. Anhand dieser Kennlinien kann man für jedes einzelne Bein am besten einen Schwellwert feststellen, ab welchem das Bein nicht mehr unbelastet ist. Ohne Belastung nehmen Servos etwa 5 mA auf. Um auf eine Belastung

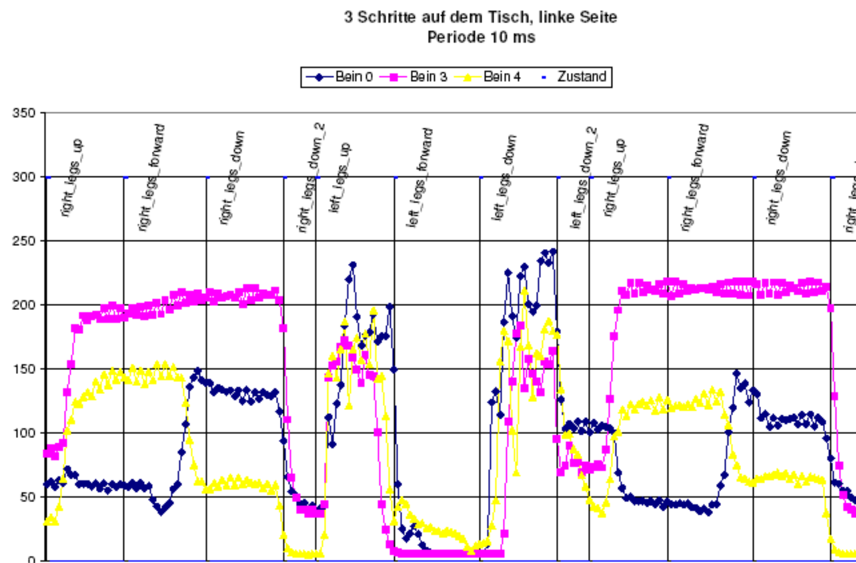


Abbildung 7: Stromaufnahme von Servos auf dem Tisch, linke Seite

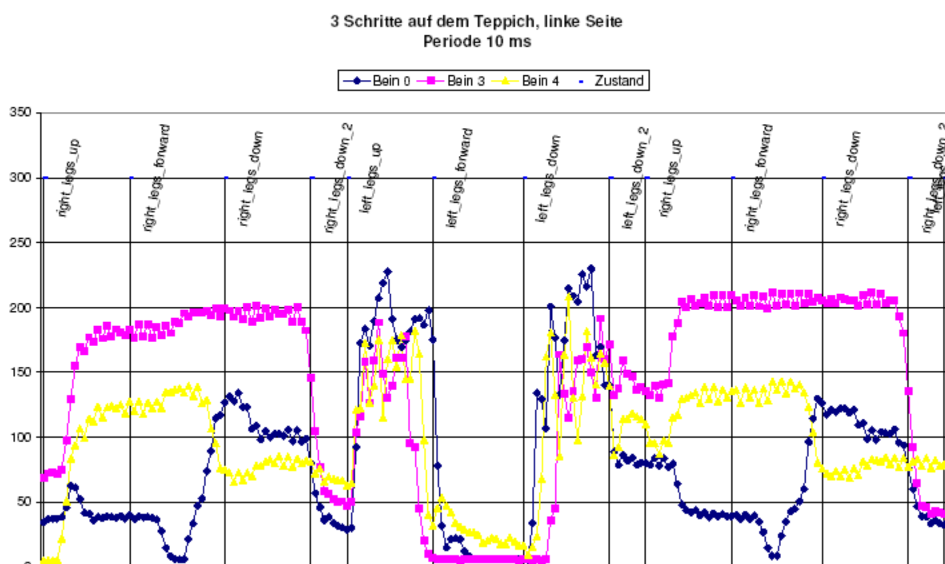


Abbildung 8: Stromaufnahme von Servos auf dem Teppich, linke Seite

zu schließen, sollte der Schwellwert im Bereich zwischen 25 und 50 mA liegen. Je höher dieser Wert ist, um so schneller kann auf Fehlen des Bodenkontakts geschlossen werden. Damit steigt aber auch die Wahrscheinlichkeit einer Fehlannahme.

Diese sechs Diagramme sind in Bereiche aufgeteilt. Diese Bereiche gehören zu den Teilschritten. Zum Beispiel bedeutet der Teilschritt *left_legs_up*, dass hier linkes vorderes, linkes hinteres und rechtes mittleres Bein angehoben werden (Im Diagramm wird

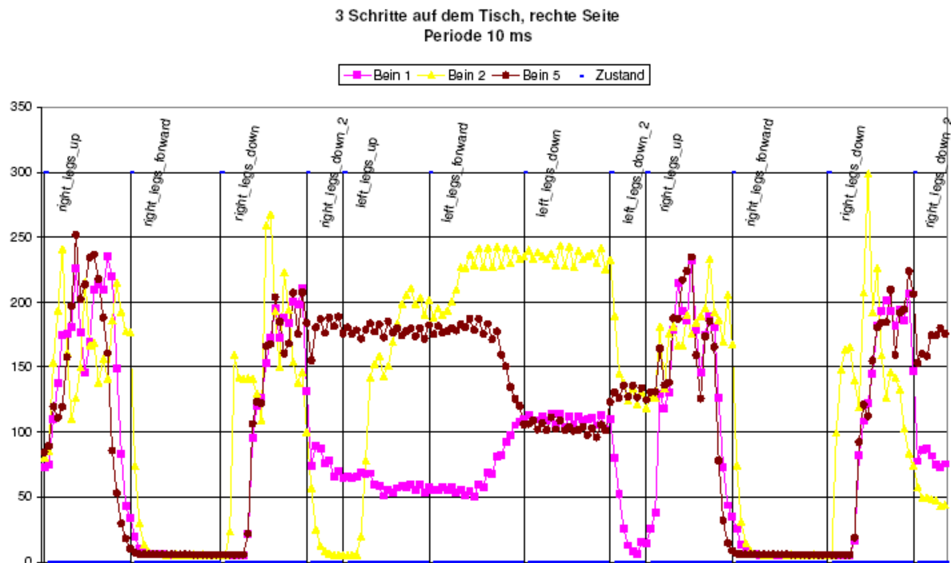


Abbildung 9: Stromaufnahme von Servos auf dem Tisch, rechte Seite

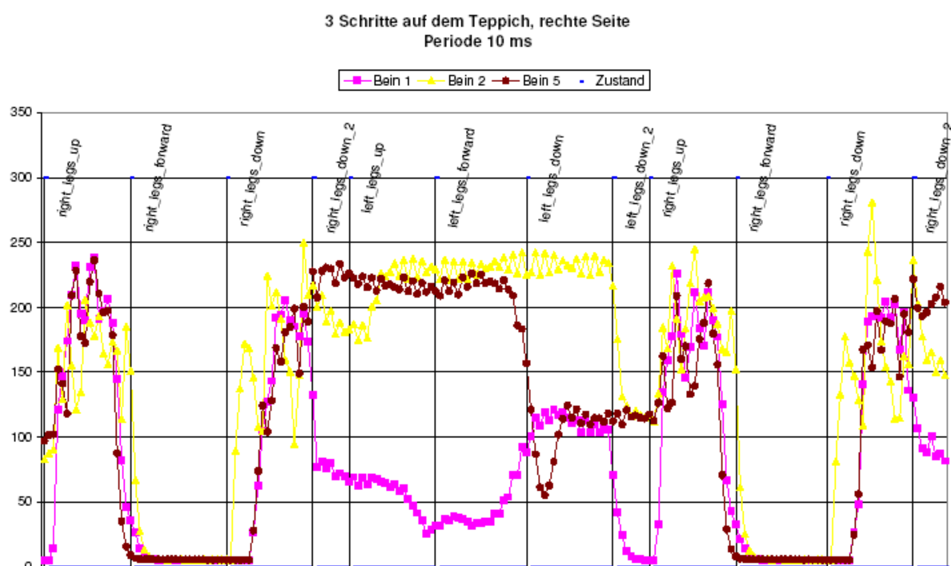


Abbildung 10: Stromaufnahme von Servos auf dem Teppich, rechte Seite

diese Kombination mit „linke Seite“ bezeichnet). Für das Feststellen des Bodenkontakts sind Zustände mit den Endungen „up“ und „down“ von Interesse. In diesen Zuständen werden Höhenservos bewegt. Die Stromaufnahme steigt auf das Mehrfache an. Man beachte, dass die Stromaufnahme in den nachfolgenden Zuständen noch erhöht ist. Es dauert einige Zeit, bis sie wieder auf den tiefsten Wert fällt. Aus diesem Grund wurde

der Zustand mit der Endung „down_2“ eingeführt, wo 100 ms abgewartet wird, bevor der Bodenkontakt gecheckt wird. Innerhalb dieser Zeitspanne sollte sich die Stromaufnahme stabilisieren und auf den tiefsten Punkt fallen, falls kein Bodenkontakt besteht. Vor Ablauf dieser Zeit ist es nicht sinnvoll, die Stromaufnahme zu messen.

Diese Vorgehensweise führt zu einer Verzögerung beim Gehen. Auch im günstigsten Fall, wenn alle Beine auf dem Grund stehen, muss diese Zeit abgewartet werden. Findet ein Bein nicht den Boden, wird das Bein weiter nach unten gestellt. Dabei muss wieder eine Pause eingehalten werden. Bei unebenem Boden entstehen mehrere solche Pausen. Das Gehen ist deswegen nicht besonders schnell.

Ist der Boden eben, kann man die Suche nach dem Boden mit der Methode *setDelayToCheckGroundContact()* ausschalten, indem die Verzögerung auf Null gesetzt wird. Beim Programmstart ist Überprüfen des Bodenkontakts ausgeschaltet. Bei den Tests war aktivierte Bodensuche hinderlich, weil oft eines der Höhenservos defekt war. So musste man nach jedem Programmstart zunächst diese Option deaktivieren.

Diese Methode kann auch benutzt werden, um den Gangablauf zu tunen. Bei Verzögerungen unter 80 ms konnte der Roboter im Test jedoch keinen Abgrund mehr erkennen. Ein zuverlässiger Wert liegt bei 150.

Diagramme 7 bis 10 zeigen Stromaufnahmen beim Gehen auf unterschiedlichen Oberflächen: Tisch und Teppich. Hier sind keine besonderen Unterschiede zu vermerken, da beide Oberflächen eben sind.

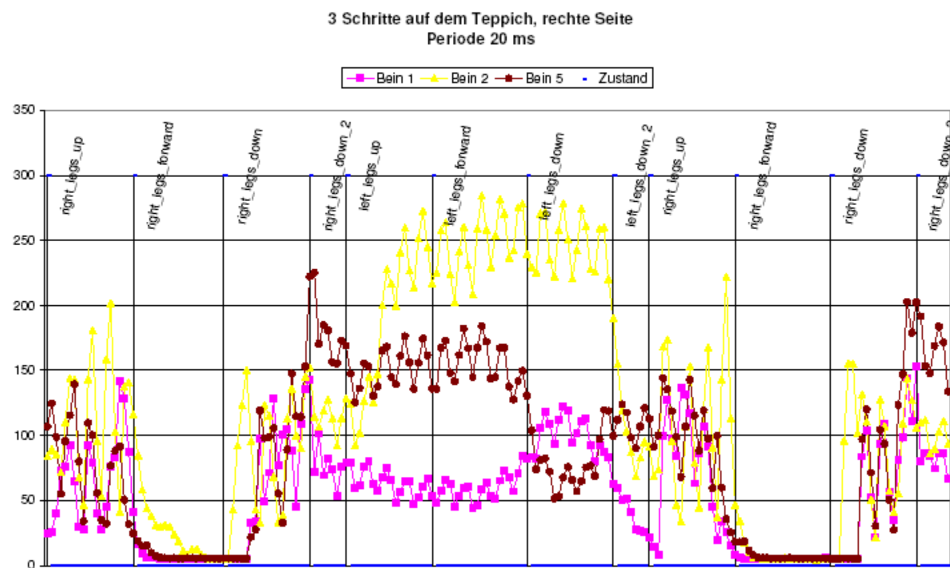


Abbildung 11: Stromaufnahme von Servos auf dem Teppich bei seltener Servo-Ansteuerung, rechte Seite

Das Diagramm in der Abbildung 11 zeigt Stromaufnahme von Servos bei gleichen Bedingungen wie in Abbildung 10 mit dem Unterschied, dass dabei Servos nur halb so oft angesteuert worden sind. Servos erwarten einen Steuersignal jede 10 bis 25 ms. In den Diagrammen 5 bis 10 wurde das Steuersignal jede 10 ms wiederholt, im Diagramm

11 jede 20 ms. Die Stromaufnahme in der Abbildung 11 wurde vor den anderen gemessen. Man sieht, dass hier große Schwankungen der Werte auftreten. Das liegt daran, dass Servos nur zum Zeitpunkt des Steuersignals nachregeln. Aus diesem Grund wurde die Ansteuerfrequenz erhöht. Abbildungen 5 bis 10 zeigen, dass es zum positiven Ergebnis bei der Strommessung geführt hat.

Hindernisabstand Der Roboter bleibt vor Hindernissen stehen, welche in weniger als 10 cm Entfernung erkannt werden. Das Erkennen von Hindernissen geschieht mit dem Ultraschall-Sensor. Der Sensor liefert den Abstand centimetergenau. Der Roboter kann bis zu 8 cm an das Hindernis herantreten, bevor er stehen bleibt, da er sich in einem Schritt um 2 cm fortbewegt und eine Entfernung von 10 cm noch zugelassen wird. Der Roboter sollte vor Hindernissen also in 8 bis 9 cm Abstand stehen bleiben.

Zur Hindernis-Erkennung könnte auch der Infrarot-Sensor verwendet werden. Er hat aber zwei Nachteile. Erstens, der Infrarot-Sensor misst punktgenau. Damit müsste öfter der Turm gedreht werden, um den vorderen Bereich zu checken. Dabei könnte er auch nur Hindernisse erkennen, welche sich in der Höhe des Sensors befinden. Beim Ultraschall-Sensor tritt dieses Problem nicht auf, da er großen Öffnungswinkel hat. Der zweite Nachteil des Infrarotsensors ist sein Messbereich. Er liefert Abstände erst ab 15 cm. Bei geringeren Abständen liefert er einen Wert, welcher auf über 15 cm schließen lässt. Der Ultraschall-Sensor liefert dagegen einen Abstand ab 2-3 cm.

Drehen Das Drehen auf der Stelle wird mit dem Befehl *turn()* ausgelöst. Die Prozedur erwartet einen Wert in Grad. Positiver Wert bedeutet Drehrichtung nach Uhrzeigersinn. Die Kontrolle beim Drehen erfolgt in der Prozedur *turning()*. Hier werden neue Winkel für Servos berechnet und mit der Methode *setServoTargetAngleTenthDegree()* gesetzt. Die Geschwindigkeit kann wie beim Gehen über Methoden *setHeightDelayPerDegree()* und *setDistanceDelayPerDegree()* eingestellt werden.

In einem Drehschritt wird der Roboter um 10° gedreht. Ist der einzustellende Winkel geringer als 10°, macht er kleinere Schritte. Soll der Roboter um 25° drehen, so macht er 2 Schritte mit 10-Grad-Winkel und einen Schritt mit 5°. Soll er noch um 10° drehen, dreht er in zwei Schritten mit 5°. Bleibt der Roboter in einem nicht abgeschlossenen 10-Grad-Schritt, so versucht er zunächst eine Endposition zu finden, um mit vollen 10-Grad-Schritten zu drehen.

Nach jedem Drehschritt wird der Host über den aktuellen Winkel relativ zum Anfangswinkel benachrichtigt. Der Benutzer hat die Möglichkeit einzustellen, ob Kompaß zur Korrektur beim Drehen einbezogen werden soll. Zum Start des Programms ist die Korrektur abgeschaltet. Sie wird mit der Prozedur *setCheckCompass()* beeinflusst. 1 aktiviert sie, bei 0 wird sie wieder deaktiviert. Da das Kompassmodul 3-4 Grad genau ist, wird im Korrekturmodus nur nach Kompass korrigiert, wenn die Abweichung mehr als 3 Grad beträgt. Bei größeren Winkeln ist aktivierte Kompasskontrolle sinnvoll. Dazu sollte sichergestellt werden, dass das Kompassmodul korrekte Winkelwerte liefert. Anderenfalls sollte es zunächst kalibriert werden.

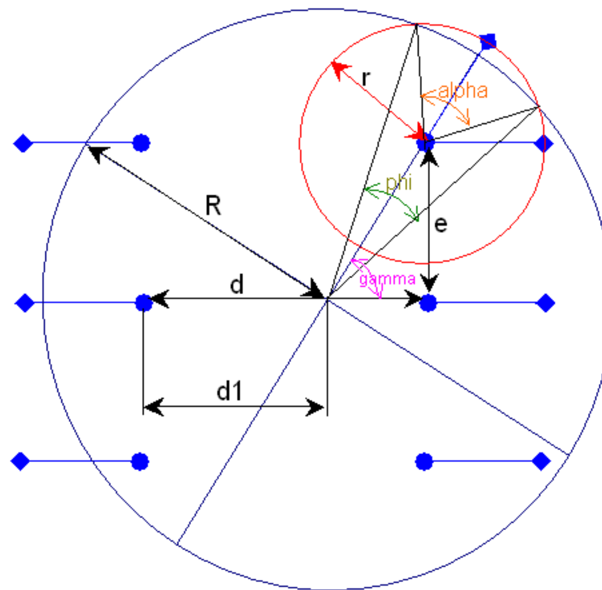


Abbildung 12: Berechnung der Winkel für Beine beim Drehen

Berechnung der Winkel Um die Plattform zu drehen, müssen Beine passend angesteuert werden. Zunächst wurde versucht, um Mittelbeine herum zu drehen. Dazu stand ein Mittelbein einer Seite starr auf dem Boden, während das vordere und hintere Bein der anderen Seite nach vorne oder hinten bewegt wurde. So drehte sich der Roboter um das mittlere Bein. Nachteilig war nicht nur die Tatsache, dass das Zentrum der Drehung an der Seite des Roboters lag, sondern auch, dass solche Drehschritte nur geringe Drehwinkel ermöglichten. Später wurde eine bessere Möglichkeit entdeckt, beide Nachteile zu umgehen.

In neuer Lösung sollten Beine sternförmig angeordnet sein, so dass sie vom Zentrum der Drehung gleichen Abstand haben sollten. Damit war es möglich, alle drei Beine zu bewegen, die in einem Schritt auf dem Boden standen. So waren größere Drehwinkel in einem Schritt möglich und das Zentrum versetzte sich weiter zur Mitte des Roboters.

Anhand der Abbildung 12 wird die Berechnung der Beinpositionen erklärt. Blaue Linien mit einem Kreis und einer Raute am Ende stellen Beine des Roboters dar. Der große Kreis mit dem Radius R befindet sich im Zentrum der Drehung. Auf der Linie dieses Kreises setzen sich Beine ab, welche den folgenden Schritt durchführen: rechtes vorderes, rechtes hinteres und linkes mittleres Bein. Die Startposition und die Endposition dieser Beine befindet sich im nächsten Schritt auf dieser Linie. Schnittpunkte des kleinen Kreises mit dem großen deuten Start- und Endposition für das rechte vordere Bein an. Dabei beschreibt der kleine Kreis die Bewegung des Beins. Der Winkel *alpha* ist der erforderliche Winkel für dieses Bein, um die Plattform um den Winkel *phi* zu drehen. Da Abstände der drei Beine zum Zentrum der Drehung gleich sind, gilt der

Winkel *alpha* auch für das rechte hintere und linke mittlere Bein.

Was noch fehlt, ist der Winkel *gamma*, der angibt, wie man Eckbeine ausrichten muss, bevor drei stützende Beine um *alpha* gedreht werden. Der Radius des Beins *r* beträgt 55 mm, Achsen zwei benachbarten Beine einer Roboterseite *e* sind 60 mm voneinander entfernt und der Abstand zwischen den Achsen der Beine beider Seiten *d* beträgt 97 mm. Daraus wird mit der Formel $d1 = \frac{e^2+d^2}{2d}$ der Abstand des Mittelpunkts der Drehung von der Achse des linken mittleren Beins berechnet. Das Ergebnis ist 67 mm. Mit $arcsin(\frac{e}{d1}$ wird der Winkel *gamma* berechnet. In unserem Fall: *gamma* = 63.5°.

Nun kann mit der Formel $R = d1 \cdot cos(\frac{phi}{2}) + r \cdot cos(\frac{phi}{2})$ der große Radius für verschiedene Winkel für einen Schritt berechnet werden. Und die Formel $alpha = 2 \cdot arcsin(\frac{R \cdot sin(\frac{phi}{2})}{r})$ berechnet den entsprechenden Winkel zum Ansteuern der Beine.

phi	R	alpha	delta r
1	122,1	2,2	0,01
2	122,0	4,4	0,04
3	122,0	6,7	0,09
4	122,0	8,9	0,17
5	121,9	11,1	0,26
6	121,9	13,3	0,37
7	121,8	15,5	0,51
8	121,8	17,8	0,66
9	121,7	20,0	0,83
10	121,6	22,2	1,03
11	121,5	24,4	1,25
12	121,4	26,7	1,48
13	121,3	28,9	1,74
14	121,1	31,1	2,02
15	121,0	33,4	2,32

Tabelle 1: Berechnung der Winkel für Drehen

Die Tabelle 1 zeigt zu den Winkeln von 1 bis 15 Grad für *phi* die zugehörigen Werte für den großen Radius *R* und den Winkel für Beine *alpha*. Außerdem wird hier die Weite angegeben, um welche der kleine Kreis aus dem großen hinausgeht. Dieser Wert muss von Beinen elastisch vertragen werden. Je kleiner dieser Wert ist, umso besser ist es für Beine, weil sie weniger beansprucht werden. Ist dieser Wert zu groß, kann Beinkonstruktion beim Drehen beschädigt werden, oder Beine werden auf dem Boden durchrutschen. Bei dem benutzen Winkel *phi* = 10°, müssen Beine beim Drehen etwa 1 mm elastisch aufnehmen. Dieser Wert sollte keinen negativen Einfluss auf die Beinkonstruktion aufweisen. Während des Tests lies es sich nicht auf Beeinträchtigungen schießen.

Normalhöhe Auf dem Roboter kann die Höhe der Plattform über dem Boden eingestellt werden. Diese Höhe wird hier als Normalhöhe oder Nullhöhe bezeichnet. Soll

der Roboter auf einem stark unebenen Untergrund laufen, kann die Normalhöhe höher eingestellt werden. Dazu stehen zwei Prozeduren zur Verfügung. Die Methode *setNullHeight()* stellt Normalhöhe für alle Beine, die Methode *setLegNullHeight()* für ein einzelnes Bein ein. Der Wert für die Normalhöhe ist kein absoluter Wert. Nach dem Start des Roboterprogramms hat die Normalhöhe für jedes Bein den Wert Null. Soll die Plattform höher gestellt werden, übergibt man positive Zahl, bei Tieferstellung eine negative. Der Wert selbst hat Millimeter als Einheit. Wie bei der Einstellung der Beinhöhe wird auch hier keine korrekte Berechnung der Winkel für Servos mit der arcsin-Funktion benutzt, sondern der gleiche Höhenfaktor angewendet.

Das Einstellen der Normalhöhe bei einzelnen Beinen kann dazu verwendet werden, ein Bein in Höhe zu Kalibrieren. Eine Kalibrierung der Beine ist in der Software programmiert. Will man Beine neu einstellen, kann man durch einen Aufruf dieser Methode Beine auch vom Host aus kalibrieren. Mit dem Aufruf der Methode *setNullHeight()* werden Einstellungen der Normalhöhe für einzelne Beine überschrieben. Damit geht eventuelle Kalibrierung der Beine über die Prozedur *setLegNullHeight()* verloren. Sie geht auch verloren, wenn das Programm auf dem Roboter neu gestartet wird. Eine Verbesserung in diesem Fall würde das Speichern der Nullhöhen von Beinen im Flash bringen.

Setzen der Normalhöhe sind keine direkten Methoden. Das bedeutet, dass nach dem Aufruf dieser Methoden eine Veränderung der Höhe mit der Geschwindigkeit geschieht, welche durch die Prozedur *setHeightDelayPerDegree()* beeinflusst wird.

Neigen Die Roboterplattform kann nach vorne, nach hinten und zur Seite geneigt werden. Zum Neigen nach vorne und zurück steht die Prozedur *lean()* bereit. Sie erwartet einen Parameter als Winkel in Grad. Beim positiven Wert, wird der Roboter nach hinten geneigt. Der zulässige Bereich liegt von -10 bis +10.

Zum Neigen der Plattform zur Seite dient die Prozedur *tilt()*. Sie erwartet ebenfalls einen Parameter in Grad. Positive Werte neigen den Roboter zur rechten Seite. Zulässige Werte liegen im Bereich von -5 bis +5.

Die beiden Methoden hängen voneinander ab. Wurde mit einer Methode der Maximalwert eingestellt, kann mit der anderen Methode keine Neigung mehr eingestellt werden. Wird es dennnoch versucht, wird der Aufruf ignoriert und der Host empfängt eine Fehlermeldung.

Wird nach dem Neigen eine andere Servo-Funktion auf Beinen angewandt, gehen Neigungseinstellungen verloren. Der Roboter bleibt in der alten Lage, wenn einzelne Beine oder Servos angesprochen werden, bis *lean()* oder *tilt()* wiederholt aufgerufen wird.

Im Entwurf wurde geplant, dass das Neigen intern die Methode *setLegNullHeight()* aufruft. Damit wäre es auch möglich, in einer schiefen Lage zu gehen. Das wurde später fallen gelassen, da dabei geringere Neigungswinkel zugelassen werden sollten, um einen Speilraum zum Anheben der Beine noch zu haben. Stattdessen wurde zum Neigen ein zusätzlicher Zustand wie für Gehen und Drehen eingeführt. In diesem Zustand werden Beine parallel angeordnet. Bei dieser Beinaufstellung ist es einfacher und prä-

ziser, die geforderte Neigung einzustellen.

Beim Neigen wird der geforderte Winkel nicht, wie es korrekt wäre, über Funktionen wie arctan() und arcsin() in Winkel für Servos umgewandelt. Stattdessen werden hier Faktoren benutzt, wie beim Einstellen der Höhe oder Position von Beinen. Der Faktor für *lean()* beträgt 4.4 pro Grad, 7.6 für *tilt()*. Diese Werte wurden zunächst rechnerisch ermittelt, dabei wurde der Faktor zur Höheneinstellung pro mm verwendet. Beim Test wurde die Abweichung gemessen und Faktoren wurden angepasst.

Dabei trat durch die Konstruktion der Beine ein positiver Effekt auf. Der Aufbau der Beine führt dazu, dass Servos bei einem positiven Winkel Beine weniger hoch stellen, als bei einem negativen Winkel mit dem gleichen Wert nach unten bewegen. Da beim Neigen ein Teil der Beine nach oben und ein anderer Teil nach unten gestellt eird, kompensieren sich diese Unterschiede und führen zu einer genaueren Winkeleinstellung beim Neigen, als es ohne diesen Effekt mit Faktoren möglich wäre.

Ausrichtung der Beine Bewegungsbefehle Gehen, Drehen und Neigen beginnen mit einer Beinausrichtung, die zum jeweiligen Befehl erforderlich ist. Die Ausrichtung wird nur dann nicht vollzogen, wenn der Roboter bereits in der richtigen Lage steht. So braucht der Roboter zum Beispiel keine wiederholte Ausrichtung der Beine, wenn er nach einem Laufbefehl noch ein Befehl zum Gehen erhält. Wird jedoch zwischendurch ein anderer Bewegungsbefehl aufgerufen (auch Ansteuerung einzelner Beine oder Servos), wird die Ausrichtung wiederholt.

Warteschlange für Befehle Auf dem Roboter ist eine kleine Warteschlange für Bewegungsbefehle Gehen, Drehen und Neigen implementiert. Sie hat die Länge von 1. Das bedeutet, dass zusätzlich zum aktuell ausführenden Befehl ein weiterer Bewegungsbefehl gespeichert werden kann. Dieser wird nach dem Beenden des aktuellen Befehls gestartet. Kommt ein weiterer Bewegungsbefehl an, während die Warteschlange bereits voll ist, ersetzt er den Befehl in der Warteschlange. Wird mit der Methode *stopMovement()* die Bewegung des Roboters unterbrochen, so wird der aktuelle Befehl beendet und die Warteschlange geleert. Will man nur den Befehl in der Warteschlange löschen, kann man ein *step*- oder *turn*-Befehl mit Parameter 0 aufrufen (am besten gleiche Befehlsart, die gerade ausgeführt wird, um eine Ausrichtung der Beine zu vermeiden).

Einfache Befehle wie Ansteuern von einzelnen Beinen oder Servos werden nicht in die Warteschlange aufgenommen. Diese werden gleich nach dem Aufruf ausgeführt. Ist der Roboter in dieser Zeit mit der Ausführung von Gehen, Drehen oder Neigen beschäftigt, wird ein solcher Befehl keine Auswirkung haben. Es wird höchstens zu einer Zuckbewegung eines Beins kommen.

4.3.5 Sensoren

Ultraschallsensor In diesem Abschnitt werden alle Funktionen aufgelistet die zum Konfigurieren und zum Auslesen des Ultraschallsensors zur Verfügung stehen.

```
void setUSMultiplier(int usMultiplier)
```

ändert den aktuellen Verstärkungswert des US-Sensors

`usMultiplier` neuer Verstärkungswert des US-Sensors als int
Wertebereich: 0 - 31
Standardwert: 31

```
void setUSRange(int usRange)
```

schreibt die übergebene maximale Reichweite in das Reichweitenregister des US-Sensors

`usRange` neue maximale Messreichweite des US-Sensors
als int
Wertebereich: 0 - 255
Standardwert: 255

```
void startUSBurst()
```

starte einen US-Impuls und löst damit eine US-Messung aus

```
struct arr_values getUSValues()
```

liest alle Entfernugnsregister des US-Sensors aus und gibt die Messwerte zurück

Rückgabe: die vom US-Sensor gemessenen Entfernungen in cm als int array mit 16 Elementen in der Struktur us_values

```
int getUSValue(int valueID)
```

liest ein bestimmtes Entfernungsregister des US-Sensors aus und gibt den Messwert zurück

`valueID` die ID des auszulesenden Entfernungsregisters
als int
Wertebereich 0 - 15

Rückgabe: die vom US-Sensor gemessene Entfernung in dem gewünschten Register in cm als int

`int getFirstUSValue()`

liest das erste Entfernungsregister des US-Sensors aus und gibt den Messwert zurück

Rückgabe: die vom US-Sensor gemessene Entfernung im ersten Entfernungsregister in cm als int

`int getLSValue()`

liest das Register des Lichtsensors aus und gibt den Messwert zurück

Rückgabe: die vom Lichtsensor gemessene Helligkeit als int

Kompass In diesem Abschnitt werden alle Funktionen aufgelistet die zum Konfigurieren und zum Auslesen des Kompass zur Verfügung stehen.

`void calibrateCompass()`

sendet das Kalibrierungssignal an den Kompass

`int getCompassValue()`

liest den aktuellen Messwert des Kompass aus

Rückgabe: die vom Kompass angezeigte Ausrichtung in Grad als int

Infrarotsensor In diesem Abschnitt werden alle Funktionen zum Auslesen des Infrarotsensors aufgelistet.

int getIRValue()

liest den aktuellen Messwert des IRsensors aus

Rückgabe: Abstand in cm im Bereich zwischen 15 und 140 cm als int, Werte größer gleich 140 bedeuten kein Hindernis

Neigungssensor In diesem Abschnitt werden alle Funktionen zum Auslesen des Neigungssensors aufgelistet.

struct arr_values getAccelerationValues()

liest beide Achsen des Beschleunigungssensors aus und gibt die daraus ermittelte Neigung zurück

Rückgabe: die aktuelle Neigung des Sensors in Grad als int array mit 2 Elementen in der Struktur acc_values

Servo Stromaufnahme In diesem Abschnitt werden alle Funktionen zum Auslesen der Stromaufnahme der Servos aufgelistet.

int getServoCurrent(int servoID)

liest die aktuelle Stromaufnahme eines bestimmten Servos aus

`servoID` die Servo-ID des Servos von dem die Stromaufnahme bestimmt werden soll als int

Rückgabe: die aktuelle Stromaufnahme des gewünschten Servos in mA als int

void getServoCurrents(int *int_array)

liest die aktuelle Stromaufnahme aller Servos aus

`*int_array` Zeiger auf ein Array, wo die 12 Werte der Stromaufnahme abgelegt werden sollen

Kamera In diesem Abschnitt werden alle Funktionen aufgelistet die zum Konfigurieren und zum Auslesen der Kamera zur Verfügung stehen.

`void getPicture(char *picture)`

liest ein Bild Pixel für Pixel aus dem Puffer des Camera-Controller und schreibt die Bildinformationen an die übergebene Adresse

`*picture`	Pointer auf das erste Element des Arrays in das die Bilddaten geschrieben werden sollen; das Array muss 101376 Werte aufnehmen können

`void setCamContrast(int contrastValue)`

schreibt den übergebenen Kontrastwert in das Kontrastregister der Kamera

`contrastValue`	der neue Kontrastwert als int Wertebereich: 0 - 255 Standardwert: 72

`void setCamBrightness(int brightnessValue)`

schreibt den übergebenen Helligkeitswert in das Helligkeitsregister der Kamera

`brightnessValue`	der neue Helligkeitswert als int Wertebereich: 0 - 255 Standardwert: 128

`void setCamSharpness(int sharpnessValue)`

schreibt den übergebenen Schärfewert in das Schärferegister der Kamera

`sharpnessValue`	der neue Schärfewert als int

`void resetCam()`

Resetet die Kamera und setzt alle Register auf die Standardwerte

`void setCam(int reg, int value)`

beschreibt ein beliebiges Register der Kamera

`reg`	die ID des zu beschreibende Registers
`value`	der in das Register zu schreibende Wert

int getCam(int reg)

liest ein beliebiges Register der Kamera aus

`reg`	die ID des zu leseneden Registers
Rückgabe:	der Wert des gewünschten Registers als int

Sensorscan In diesem Abschnitt werden alle Funktionen aufgelistet die zum Auslesen des Sensorscan zur Verfügung stehen.

void getScanValues(int startAngle, int numberMeasurements,short *values)

Führt 3 Messungen mit je 16 Werten für den Ultraschallsensor und maximal 37 Werte für den Infrarotsensor mit 5-Grad-Schritten durch

`startAngle`	der Startwinkel bei dem der Sensorscan starten soll
`numberMeasurements`	Anzahl Messungen für Infrarotsensor in 5-Grad-Schritten
`*values`	short-Array, wo maximal 85 Messwerte gespeichert werden sollen

4.4 Host

4.4.1 Karte und Navigation

Beide Roboter verwenden die Karte gemeinsam. Sie dient der Navigation des Insektenroboters und hilft dem Benutzer bei der Orientierung. In die Karte werden alle Objekte, die mit den Sensoren der Roboter erkannt wurden, eingetragen.

Karte Um den Zugriff auf die Karte aus allen Teilen des Programms (Insekten und 4WD) einfach zu gestalten, wurde die Klasse `MapManager` als Singleton implementiert. Hier können vorhandene Objekte abgefragt und neue eingetragen werden. Darüber hinaus dient diese Klasse als Schnittstelle zum 4WD-Roboter. So kann seine aktuelle Position mit auf der Karte dargestellt werden.

Die Karte ist als ein Raster mit einem kartesischen Koordinatensystem aufgebaut. Ein Feld in diesem Raster entspricht dabei 1 cm^2. Bis auf das Umrechnen zwicken den Schritten des Roboters und den Entfernungen auf der Karte wird der Maßstab in der Karte nicht berücksichtigt. Es kann also bei Bedarf mit geringem Aufwand auf ein feineres Raster umgestellt werden.

Datenbank Alle Daten der Karte werden mit Hilfe der Bibliothek *sqlite* in einer Datei als relationale Datenbank gespeichert. Die Daten können mit der SQL abgefragt und bearbeitet werden. Es wurde nur das Speichern der mit den Abstandssensoren (Infrarot, Laser) erkannten Objekte implementiert. Als Erweiterung könnte die Datenbank weitere Daten aufnehmen. Sinnvoll wäre zum Beispiel das Speichern der aufgenommenen Fotos.

Speichern der Fotos

Das Datenbankschema besteht aus zwei Tabellen. Die erste, `map`, ist für allgemeine Informationen zur Karte vorgesehen. Bist jetzt ist dieses nur der Name der Karte, den man frei wählen kann. In die zweite Tabelle, `objects`, werden die Messungen der Abstandssensoren eingetragen. Für jeden Messung werden folgende Daten erfasst:

- Die Position des Objekts auf der Karte (x/y-Koordinaten)
- Ungenauigkeit der Messung in cm
- Zeitstempel
- Herkunft der Messung (Insekt, 4WD, Benutzer)

Darstellung Die Darstellung der Karte erfolgt so, wie im Entwurf vorgesehen. Jede Messung wird als ein Kreis mit einem Durchmesser entsprechend der Ungenauigkeit dargestellt. Der Mittelpunkt hat den Wert 255 einer Farbe. Dieser Wert nimmt zum Rand hin linear ab. Überlagern sich zwei solcher Kreise, wird die Farbe aus dem Durchschnitt der sich schneidenden Bereiche berechnet (Beispiel in Abbildung 13).

6	0	0	0	85	85	85	0
5	0	0	85	85	170	85	85
4	0	0	85	170	255	170	85
3	0	0	85	85	170	85	85
2	0	0	0	85	85	85	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

(a) Messung A

6	0	0	0	0	0	0	0
5	0	0	42	42	42	0	0
4	0	42	42	85	42	42	0
3	0	42	85	127	85	42	0
2	0	42	42	85	42	42	0
1	0	0	42	42	42	0	0
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

(b) Messung B

6	0	0	0	85	85	85	0
5	0	0	63	63	106	42	42
4	0	42	63	127	148	106	42
3	0	42	85	106	127	63	42
2	0	42	42	85	63	63	0
1	0	0	42	42	42	0	0
0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6

(c) A + B

Abbildung 13: Überlagerung von zwei Messungen

Die Darstellung wurde zuerst mit Hilfe Qt-Klasse `QCanvas` implementiert, musste dann aber neu geschrieben werden. Die Canvas-Klassen waren bei sehr vielen Objekten auf der Karte zu langsam und benötigten viel Speicher. Jetzt wird der sichtbare Bereich der Karte direkt gezeichnet, was auch bei großen Karten (50x50 m^2) mit vielen Objekten performant ist.

Navigation Erhält der Roboter einen Befehl zum Gehen (oder Drehen), so sendet dieser nach jedem Schritt die Anzahl der bisher bei diesem Befehl gemachten Schritte an den Host. Beim drehen wird der Winkel, um den sich der Roboter gedreht hat, gesendet. Mit diesen Informationen kann auf der Karte dargestellt werden was der Roboter im Moment macht und an welcher Position dieser sich befindet.

Die Position wird nach jedem Schritt aktualisiert. Die Berechnung der neuen Position erfolgt dabei jedoch nicht von der Position beim letzten Schritt aus, sondern von der, an dem sich der Roboter am Anfang des Befehls befunden hat. So wird das Aufaddieren von Rundungsfehlern bei längeren Wegen vermieden. Alle Informationen über die Position und Richtung des Roboters wergen in der Klasse `Insect` berechnet und gespeichert.

Wegeablauf Ein vom Wegefindealgorithmus geplanter weg wird auf der Karte angezeigt und kann vom Benutzer verändert werden. Soll ein solcher Weg abgelaufen werden, wird für jede Teilstrecke Richtung und Länge berechnet und die Befehle zum Drehen und Gehen in die Warteschlange gestellt.

4.4.2 Wegefindung

Als Wegefindung soll das Problem bezeichnet werden, auf einer Karte, auf der Hindernisse eingezeichnet sind, einen begehbaren Weg zwischen zwei Punkten zu finden. Die Karte ist dabei in einem Pixelformat gegeben. Jedes Kartenquadrat entspricht einem Quadratzentimenter - das ist aber für die Wegefindung uninteressant. Für jedes Kartenquadrat lässt sich abfragen, ob es passierbar ist oder nicht. Dabei ist zu bedenken, dass die Karte auf der Grundlage von Sensordaten erstellt wird. Das bedeutet, dass größere Hindernisse nicht komplett als nicht passierbar gespeichert sind, sondern nur ihre Ränder.

Wege Um einen Algorithmus schreiben zu können, der Wege findet, muss ersteinmal festgelegt werden, was Wege überhaupt sind. Ein Weg w ist eine Folge von n Wegpunkten: $w := W_1, W_2, ..., W_n$. Die Wegpunkte sind durch ihre Koordinaten gegeben. Solch ein Weg spiegelt die Tatsache wider, dass der Insektenroboter immer nur gerade Strecken geht und sich nur auf der Stelle drehen kann. Dafür ist solch ein Weg optimal. Den Abschnitt zwischen zwei Wegpunkten W_i und W_{i+1} bezeichnet man als *Teilpfad*.

Wegpunkte werden durch die Klasse `WayPoint` dargestellt. Sie ist eine Tochterklasse der Klasse `QPoint`, die im wesentlichen zwei Ganzzahl-Variablen enthält. Wege werden von der Klasse `Path` repräsentiert, die als Tochterklasse von der Template-Klasse `QValueList<WayPoint>`. Eine `QValueList` ist eine einfach verkettete Liste, auf die hauptsächlich über Iteratoren zugeriffen wird. Die Klassen `QPoint` und `QValueList` entstammen dem QT-Paket.

Länge Die Länge eines Weges ist durch die Länge seiner Teilwege festgelegt:

$$\sum_{i=1}^{n-1} \sqrt{(W_{ix} - W_{i+1_x})^2 + (W_{iy} - W_{i+1_y})^2}$$

Passierbarkeit Während die Länge eines Weges allein durch seine Wegpunkte gegeben ist, kann man Aussagen über die Passierbarkeit nur in Abhängigkeit von einer Karte $\mathcal{M}$. Eine Karte ist eine Abbildung

$$\mathcal{M} : Z \times Z \rightarrow \{passierbar, unpassierbar\}$$

Durch die Karte wird festgelegt. ob ein Kartenquadrat passierbar ist. Damit ein Weg passierbar ist, müssen alle Kartenquadrate, die zwischen zwei aufeinanderfolgenden Wegpunkten liegen passierbar sein. Die Punkte, die eine Linie in einem Pixelfeld liegen, können mit Hilfe eines Algorithmus von Bresenham abgelaufen werden. Dieser Algorithmus wurde entwickelt, um Linien ohne Multiplikation und Division zeichnen zu können. Hier wird er dazu verwendet, um schnell alle Kartenquadrate zwischen zwei Wegpunkten ablaufen zu können und deren Passierbarkeit überprüfen zu können: Denn nur wenn alle Kartenquadrate passierbar sind, kann man vom einen Wegpunkt den anderen erreichen. Der Algorithmus wurde in der Klasse `PathPointIterator` implementiert. Mit einem solchen Objekt, das mit einem Iterator auf einen Wegpunkt des Pfades konstuiert wird, werden alle auf dem Pfad gelegenen Punkte abgelaufen.

Aber das genügt nicht, denn es wird ja ein Weg für ein Objekt einer bestimmten Größe gesucht. Es muss also eine Anzahl von Linie parallel zur *Hauptlinie*, die zwischen zwei aufeinanderfolgenden Wegpunkten verläuft überprüft werden. Nur wenn alle Pixel auf der Hauptlinie und auf den dazu parallelen Linie passierbar sind, ist der Teilpfad passierbar. Nur wenn alle Teilpfad eines Weges passierbar sind, ist der gesamte Weg passierbar.

Achsenparallelität Ein Weg w der mit n Wegpunkten heißt *achsenparallel* gdw

$$\forall i, 1 \leq i < n : W_{ix} = W_{i+1_x} \vee W_{iy} = W_{i+1_y}$$

, wenn alle alle Teilpfade parallel zu einer der Achsen des Koordinatensystems sind.

Newton-Wegefinder: Hindernisse umgehen Newton-Wegefinder implementieren ein ziemlich einfaches, aber auf nicht zu komplexen Karten ausreichend erfolgreiches Verfahren. Und weil es so einfach ist, ist es verhältnismäßig schnell. Es ist in Abbildung 14 dargestellt. Der Newton-Wegefinder liegt in zwei ähnlichen, aber doch unterschiedlichen Implementationen vor: `NewtonPF` und `NewtonBackPF`. Es wird zunächst die grundlegende Funktionsweise anhand des `NewtonBackPF` erläutert, dann wird auf die Unterschiede eingegangen.

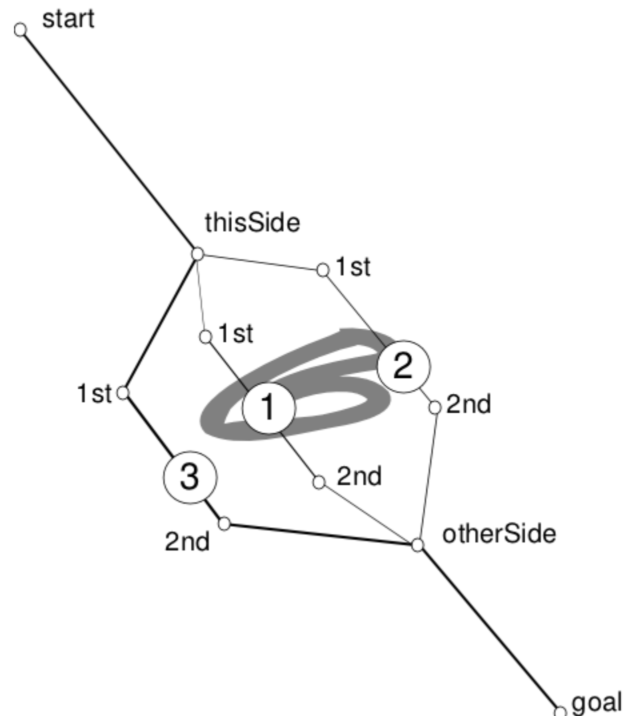


Abbildung 14: Die Arbeitsweise des Newton-Wegefinders

Die Funktionsweise des `NewtonBackPF` Der NewtonBack-Wegefinder legt zunächst einen direkten Weg von `start` nach `goal`. Dann überprüft er, ob der passierbar ist. Wenn nicht, platziert er auf beiden Seiten des Hindernisses in einer bestimmten Entfernung Punkte (`thisSide` und `otherSide`). Dann fügt er zwischen `thisSide` und `otherSide` zwei weitere Punkte ein, die das Hindernis umgehen sollen: Sie heißen firstDetour und secondDetour, in Abbildung 14 abgekürzt als „1st“ und „2nd“. Sie liegen immer auf einer Linie parallel zur Linie von `start` nach `goal`, zunächst nah an dieser Linie. Ist das Wegstück von `thisSide` nach `otherSide` dann immer noch nicht passierbar, versucht der Algorithmus `MAX_WIGGLES` oft, `firstDetour` und `secondDetour` so zu platzieren, dass der Weg von `thisSide` nach `otherSide` passierbar ist, und zwar mit steigendem Abstand zur Linie von `start` nach `goal` und immer abwechselnd mal auf der einen, dann auf der anderen Seite dieser Linie. Im Schaubild sind diese Versuche mit den Zahlen von 1 bis 3 gekennzeichnet. Versuch 3 ist dann im Beispiel im Bild erfolgreich. Passiert das allerdings nicht, werden `thisSide` und `otherSide` noch `MAX_DISTANCES` oft in anderer Entfernung zum Ziel positioniert, um dann erneut mit `firstDetour` und `secondDetour` hin und her zu "wackeln" (daher der Name der Funktion `wiggle()`, die das platzieren von `firstDetour` und `secondDetour` erledigt).

Dieses Verfahren wird dann für alle Hindernisse, die noch auf dem Weg liegen wiederholt. Kann auf diesem Wege kein Erfolg erzielt werden, wird ein Punkt `middle`

so zwischen `start` und `goal` eingefügt, dass er von beiden gleich weit entfernt ist, aber (wenn mehrere Versuche nötig sind) in immer größer werdenden Abstand von der Mitte zwischen `start` und `goal`. Auf die beiden Teilstrecken von `start` nach `middle` und von `middle` nach `goal` wird dann der oben beschriebene Algorithmus angewandt. Diese Erweiterung des Algorithmus erhöht zwar die Erfolgsquote, allerdings auch die Laufzeit, und es kann sein, dass trotz langer Laufzeit kein Weg gefunden wird. Die Wege, die auf diese Weise gefunden werden, sind darüberhinaus gelegentlich umständlicher als nötig. In Anbetracht der Tatsache, dass die Karten, die im Laufe des Projektes anfallen, nicht sehr komplex sind, wurde dieses Feature in der aktuellen Version deaktiviert, kann aber für zukünftige Tuning-Versuche durchaus interessant sein.

Abschließend sei noch gesagt, dass der Newton-Algorithmus ungeeignet ist, um Wege in Labyrinthen zu finden - aber das ist ja auch nicht das Umfeld des Insekten. Er ist dagegen gut in der Lage, Wege um einzelne, allein stehende Hindernisse herumzufinden, und damit in der typischen Umgebung des Roboters von Nutzen. Der Name „Newton“ geht übrigens auf das Newtonsche Näherungsverfahren zur Nullstellenberechnung zurück: In beiden Verfahren wird um einen Punkt von Interesse „herumgewackelt“ - auch wenn es hier ein Hindernis ist und die Abstände zunehmen, während sie beim Namensgeber kleiner werden und ein Nullpunkt Ziel der Bemühungen ist.

Die Unterschiede der beiden Implementationen Der `NewtonPF` war die erste Implementation der zu Grunde liegenden Idee und bestand (und besteht immer noch) im Wesentlichen aus einer großen Funktion. In der Absicht, das schon einigermaßen erfolgreiche Verfahren um Backtracking zu erreichen, wurde der Code des `NewtonPF` in teilweise übernommen, aber auf mehrere Funktionen aufgeteilt, die sich selbst und gegenseitig rekursiv aufrufen. Die neue Implementation, in Hinblick auf das noch zu implementierende Backtracking `NewtonBackPF` genannt, hätte exakt die selben Resultate liefern sollen wie der Vorfahre `NewtonPF`, dennoch unterschieden sie sich. Da die Unterschiede nur marginal waren, wurden sie nicht weiter beachtet. Der `NewtonPF` wurde nur mehr aus nostalgischen Gründen im Projekt belassen.

Doch nachdem die Versuche, das Backtraking im `NewtonBackPF` zu implementieren nicht erfolgreich waren, der Wegefinder aber mit anderen Methoden schneller und erfolgreicher gemacht worden war, fiel auch dem `NewtonPF` wieder eine echte Funktion zu: In Anbetracht der eingeschränkten Genauigkeit und des hohen Zeitaufwandes die mit Drehungen um beliebige (ganzzahlige) Drehungen beim Roboter einhergehen, konzentrieten sich die Bemühungen auf die Modifizierung des Algorithmus dahingehend, nur solche Wege zu produzieren, die ausschließlich 90°-Drehungen enthalten. Zu diesem Zwecke wurde beim `NewtonPF` der `squarifier` eingeführt. Dies ist ein Wegpunkt, der den Pfad achsenparallel macht Der `squarifier` wird, bevor der eigentliche Algorithmus anfängt, zwischen `start` und `goal` eingefügt. Er hat entweder die x-Koordinate von `start` und die y-Koordinate von `goal` oder umgekehrt (zunächst wird das eine probiert und dann, wenn das nicht funktioniert, das andere). Da `firstDetour` und `secondDetour` passend gewählt werden (die Teilpfade `start-thisSide` und `thisSide-firstDetour` bilden einen rechten Winkel, analog

bei otherSide), besteht der Pfad, wegen squarifier nur aus Teilpfaden, die parallel zu den Achsen sind.

Aufgrund seiner Struktur, war es beim NewtonPF viel einfacher, eine weitere weitere Modifikation zu implementieren. Um die Wege nicht unnötig zu verkomplizieren, wird für jede probierte Entfernung zunächst versucht, goal (bzw bei mehreren zu umgehenden Hindernissen den ersten unerreichbaren Wegpunkt hinter dem aktuellen Hindernis) selbst als otherSide zu verwenden. Dies wird als otherSide-Auslassung bezeichnet.

Die Veränderunden am Algorithmus, um nur *quadratische* Pfade - Pfade, in denen nur 90°-Drehungen vorkommen - zu erzeugen, wurden so implementiert, dass sie nicht immer Anwendung finden, sondern nur, wenn sich der Wegefinder im squareMode befindet. Beim NewtonPF hat der squareMode zur Folge, dass achsenparallele Pfade produziert werden, beim NewtonBackPF auch im squareMode ist die allererste Drehung in einem Pfad eine um einen beliebige Gradzahl, da die Verwendung eines squarifiers noch nicht implementiert wurde.

In NewtonBackPF gab es noch eine Veränderung, die die Performance in einem Maße reduziert hat, das den Zuwachs an Erfolg nicht rechtfertigte. Darum wurde sie deaktiviert. Weil sie aber Potenzial hat, wurde sie nicht gelöscht und soll hier beschrieben werden. Um der besseren Lesbarkeit Willen wird sie trotz ihrer Deaktivierung so beschrieben, als wäre sie noch aktiv. Wenn der erste Wegefindeversuch fehlschlägt, wird nicht ein unpassierbarer Pfad zurückgeliefert, sondern es wird von neuem mit der Suche begonnnen. Vorher wird allerdings zwischen Start- und Zielpunkt ein neuer Wegpunkt eingefügt, der zu beiden den gleichen Abstand hat. Schlägt auch der erneute Versuch fehl, so wird noch einige Male mit einem anderen Zwischenpunkt probiert. Dies findet in der Funktion NewtonBackPF::findPathNewtonBack() statt.

Wie in diesem Abschnitt beschrieben, haben sich die Unterschiede zwischen den beiden Newton-Implementationen historisch herausgebildet. Da keine grundlegenden Unterschiede bestehen, ist es durchaus möglich beide Ansätze wieder in einer Klasse zu vereinen.

Momentan erfüllen beide Varianten eine spezifische Aufgabe, da nur der NewtonPF achsenparallele Pfade produzieren kann, der NewtonBackPF erfolgreicher beliebige Pfade.

Verbesserungsmöglichkeiten

Newton

- NewtonPF durch Veränderungen an NewtonBackPF überflüssig machen:
  - NewtonBackPF so umbauen, das es achsenparallele Pfade erzeugt (einfach)
  - otherSide-Auslassung
- Backtracking

Wegevereinfachung Die Wege, die von den Newton-Wegefindern prodzuiert werden, enthalten konstruktionsbedingt meistens mehr Wegepunkte als nötig. Darum wird

nach dem Finden eines Pfades versucht, diesen zu vereinfachen. Dazu stellt die Klasse `Path` einige Funktionen zur Vereinfachenung zur Verfügung. Einige der Vereinfachnungen sind nicht für achsenparallele Pfade geeignet (denn die Achsenparallelität soll ja von den Vereinfachungen erhalten werden), einige nur für achsenparallele Pfade (da sie zusätzliche Annahmen ermöglichen) und einige sind für alle Pfade geeignet.

Nicht für achsenparallele Pfade geeignet sind:

`tryToEraseWayPoints()` Löscht einen Punkt. Wenn der Weg passierbar bleibt, wird der nächste probiert, sonst wird der gelöschte wieder eingefügt. Iteriert den Pfad immer wieder, so lange, bis keine Verbesserung mehr möglich ist.

`tryToEraseMultipleWayPoints(int number)` Arbeitet genauso wie die vorhergehende Funktion, nur dass immer `number` Wegpunkte auf einmal entfernt werden.

`eraseMinorSubPaths(int minLength)` Löscht alle Teilpfade, die echt weniger als `minLength` Pixel lang sind.

Nur für achsenparalle Pfade geeignet sind:

`straighten()` Manchmal sind Pfade, die eigentlich achsenparallel sein sollten, wegen Rundungsfehlern nicht wirklich achsenparallel. Diese Funktion behebt das.

`eraseSuperfluousPoints()` Wenn ein Punkt auf der geraden Verbindung zwischen zwei anderen Punkten liegt, wird er gelöscht. Könnte noch so erweitert werden, dass beliebige Pfade vereinfachbar sind.

Pfadvereinfachung

Für beliebige Pfade geeignet sind:

`bridgeValleys()` Macht aus zwei Hindernisumgehungen eine

`eraseLoops()` Wenn ein Wegpunkt zweimal vorkommt, werden alle Wegpunkte, die dazwischen liegen gelöscht.

`eraseImplicitLoops()` Wenn sich der Weg kreuzt, wird an der Stelle ein Wegpunkt eingefügt und die Wegpunkte auf der Schlaufe entfernt.

`equalizePlateaus()` Fasst mehrere Bewegungen in eine Richtung zusammen[2]

Alle diese Funktionen haben den Rückgabetyp `bool` und liefern genau dann `true` zurück, wenn sie den Pfad vereinfachen konnten.

[2] Aus der Bewegungsfolge Norden-Westen-Süden-Westen-Süden wird Norden-Westen-Süden - aber natürlich nur, wenn der Weg passierbar bleibt.

Der A-Stern-Algorithmus Da der A-Stern-Wegefindealgorithmus sehr erfolgreich ist (er findet immer den kürzesten Weg zwischen zwei Punkten, wenn es einen gibt), wurde er auch implementiert. Er hat sich allerdings als zu langsam herausgestellt, er hat ja auch eine Laufzeitkomplexität in O-VON-JAWASDENN???

Die Funktionsweise des A-Stern-Algorithmus ist schon vielfach beschrieben worden. Eine Modifikation verdient allerdings noch Erwähnung: Um ersten die Laufzeit zu verbessern und zweitens die produzierten Wege für den Roboter besser begehbar zu machen, arbeitet die Implementation des A-Stern-Algorithmus, `AStarPF`, nicht auf allen Punkten der Karte, sondern nur auf einem Teil. Dies geschieht mit Hilfe sogenannter Gitter, die im nächsten Abschnitt beschrieben werden.

`AStarPF`

In einem Durchlauf des `AStarPF` werden zunächst mehrere Gittergrößen ermittelt, die gut zu den Start- und Zielpunkten passen - im Konstruktor von `HexaGrid`. Dann wird auf einer nach der anderen der A-Stern-Algorithmus laufen gelassen. Eine Verbesserung der Auswahl der Gittergrößen, wobei große Größen bevorzugt werden sollten, würde also eine erhebliche Beschleunigung des `AStarPF` mit sich bringen:

- wenn keine unpassenden Gittergrößen ausprobiert werden, verringert das die Laufzeit linear
- die Größe des Gitters ist antiproportional zur Größe des Suchraums

Außerdem kann man in den A-Stern-Algorithmus eine Komplexitätsbeschränkung einbauen, die den Algorithmus erfolglos abbricht wenn der Pfad zu kompliziert wird. Dies ist sinnvoll, da der Roboter nicht beliebig komplizierte Wege laufen kann.

Wegefinder

Allgemeine Verbesserungsmöglichkeiten für Wegefinder In der aktuellen Version der Hostsoftware werden Wegesuch-Algorithmen immer auf die folgende Weise verwendet:

1. Der Benutzer bestimmt einen Zielpunkt
2. Der Algorithmus läuft - der Benutzer wartet
3. Ein Weg wird ausgegeben (im günstigen Fall)

Dieses Schema könnte man durch den Einsatz von Threads verändern. Das schon in der einfachsten Variante den Vorteil, dass der Benutzer während der Berechnung etwas anderes tun kann und nicht auf das Programm warten muss. Darüber hinaus gibt es bei der Verwendung von Threads noch mehrere zwei Verbesserungsmöglichkeiten:

- Parallelität verschiedener Algorithmen um die Effektivität zu steigern
- Anzeige und Begehung von Teilpfaden (während der Computer noch rechnet)

Bei der ersten Methode ist zu bedenken, dass sich die Threads im Allgemeinen den selben Prozessor teilen. Wenn sie also im wesentlichen die selben Pfade ausprobieren (wie etwa `NewtonPF` und `NewtonBackPF`), ist Parallelität Zeitverschwendung. Die eingesetzten Algorithmen müssen sich ergänzen, das heißt, dass der einen in Situationen

Wege findet, in denen der andere erfolglos ist und umgekehrt. Dies ist schwierig zu erreichen. Die zweite Methode hat nach Meinung des Autors mehr Potenzial. Ein einfacher Ansatz ist zum Beispiel, zunächst einen einfachen Weg zu finden und anzuzeigen und dann im Hintergrund zu versuchen in zu verkürzen - entweder durch spezielle Funktionen, oder indem man mit einem langsameren, aber effektiveren Algorithmus das ursprüngliche Wegesuchproblem nochmal löst. Dies kann auch noch geschehen, während der Roboter schon auf dem Pfad unterwegs ist. Auch kommt es vor, dass eine schnelle Suche einen Pfad findet, der zwar nicht komplett passierbar ist, aber im ersten Teil. Den zweiten Teil könnte man dann, während der Pfad schon angezeigt wird, mit einen langsameren, aber effektiveren Algorithmus überprüfen.

Gitter Der A-Stern-Algorithmus sucht Wege, in dem er für jeden momentan interessanten Punkt untersucht, wie günstig die Wege zu allen seinen Nachbarn sind. Dadurch entstehen Wege, die typischerweise sehr wenig gerade Teilstrecken enthalten. Das hätte für den Roboter bedeutet, dass er sich sehr häufig drehen muss, was sehr zeitaufwändig ist. Außerdem ist die minimale Drehung, die der Roboter ausführen kann 1°. Dies ist allerdings unverhältnismäßig zeitaufwändiger als eine 10°-Drehung - und ungenauer. Daher galt es, die Anzahl der Drehungen zu beschränken und nur Drehungen um durch zehn teilbare Gradzahlen zuzulassen. Dies geschieht über Gitter.

Definition Ein Gitter entsteht duch eine Nachbarrelation ν. Eine *Nachbarrelation* ist eine symmetrische, intransitive, nicht reflexive Relation auf den Punkten der Karte:

$$\nu \subset Z \times Z$$

Eine Nachbarrelation induziert eine Äquvalenzrelation $\equiv_\nu$:

$$p_a \equiv_\nu p_b :\Leftrightarrow \exists p_1, ..., p_n : p_1 = p_a, p_n = p_b, \forall i, 0 < i < n : \nu(p_i, p_{i+1})$$

, Die Äquivalenzklassen von $\equiv_\nu$ nennt man *Gitter*. Ein Gitter ist nur für einen Wegefindealgorithmus geeignet, wenn die Nachbarschaftrelation komplett und gleichmäßig ist:

$$\nu istkomplett :\Leftrightarrow \forall p \in Z \times Z : \exists p' : \nu(p, p')$$

$$\nu istgleichmäßig :\Leftrightarrow \forall p, p' \in Z \times Z : |\nu(p)| = |\nu(p')|$$

Ist eine Nachbarschaftrelation gleichmäßig, so hat sie den *Nachbarschaftsgrad* $N := |\nu((0,0))|$.

Ein Gitter kann die oben gestellten Anforderungen, also Beschränkung der möglichen Drehungen und lange gerade Teipfade nur erfüllen, wenn es klug gewählt ist. Ist das gegeben, darf ein Pfad auch nur aus Teilpfaden bestehen, die zwischen benachbarten Punkten verlaufen.

Implementierung Es gibt eine Klasse `Grid`, von der alle Gitter-Klassen abgeleitet werden. Die Klasse `Grid` enthält die abstrakten Funktionen `numberOfNeighbors()` und `neighborNr()`, die von allen Gittern implementiert werden müssen. Und durch

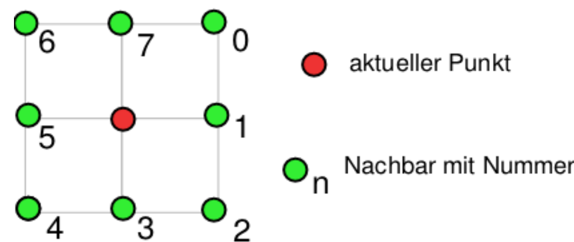


Abbildung 15: Die Punkte eines `CheckerGrid`, Kantenlänge 4

`numberOfNeighbors()` erfährt man deren Nachbarschaftsgrad und kann dann alle Nachbarn eines Punktes[3] mit der Funktion `neighborNr()` ermitteln: ihr wird ein Punkt und einen Nummer übergeben und sie liefert einen Punkt zurück. Die Nachbarn müssen zu diesem Zweck durchnummeriert sein.

Es gibt keine Funktion, mit der ermittelt werden kann, ob zwei Punkte im selben Gitter liegen. Da der A-Stern-Algorithums aber nie beliebige Punkte behandelt, sondern sich immer von Nachbar zu Nachbar vorarbeitet, stellt das kein Problem dar.

Ein Problem, das sich aber sehr wohl stellt, ist, dass sich nicht mehr alle Punkte von allen anderen aus erreichen lassen. Wenn also die Größe des Gitters ungünstig gewählt wurde, gibt es keine paarweise benachbarten Punkte zwischen Start und Ziel eines zu suchenden Weges. Darum gibt es einerseits die Funktionen `advisedSizeNr()` und `numberOfAdvisedSizes()`. Mit ihnen schlägt eine Gitter-Klasse Größen vor, bei denen der Zielpunkt auch auf dem Gitter liegt. Da das nur in den seltesten Fällen möglich ist, auch einige Größen vorgeschlagen, bei denen der Abstand des Zieles zum nächsten Gitterpunkt relativ klein ist. Wurde solch eine Gittergröße gewählt, kommt `asCloseAsItGets()` ins Spiel: ihr werden zwei Punkte übergeben, von denen der eine als auf dem Gitter befindlich angenommen wird. Gibt es keinen anderen Punkt auf dem Gitter, der näher am zweiten übergebenen Punkt liegt, liefert `asCloseAsItGets()` den Wert `true` zurück, sonst `false`.

Das Gitter, das von der Klasse `CheckerGrid` implementiert wird, könnte man als die Mutter aller Gitter bezeichnen: Das Schachbrett. Der einzige Unterschied ist, dass ein Schachbrett aus Felder besteht, ein `CheckerGrid` aber aus Punkten. Trotzdem kann man sich einen Weg auf einem `CheckerGrid` vorstellen wie den Weg eines Königs auf einem riesigen Schachbrett: er darf gerade und diagonal in jede Richtung ziehen. Die Anzahl der Nachbarn ist daher naheliegend: 8. In Abbildung 15 ist dargestellt, welche Punkte eines Pixelfeldes zu einem `CheckerGrid` gehören.

Es muss wohl kaum erwähnt werden, dass ein `CheckerGrid` besonders gut zu einem Pixelfeld, wie die verwendete Karte eins ist, passt. Allerdings erzeugt der Weg zu einem diagonalen Nachbarn einen Winkel von 45° - unverhältnismäßig großer Aufwand für den Roboter. Eine Beschränkung auf „gradlinige“ Nachbarn würde aber wiederum zu große Umwege mit sich bringen. Darum wurde das folgende Gitter ersonnen:

Ein `HexaGrid` ist eigentlich ein Dreiecksgitter. Aber da sechs Dreiecke zusammen

[3]also die Punkte, mit denen er in Nachbarschaftsrelation steht

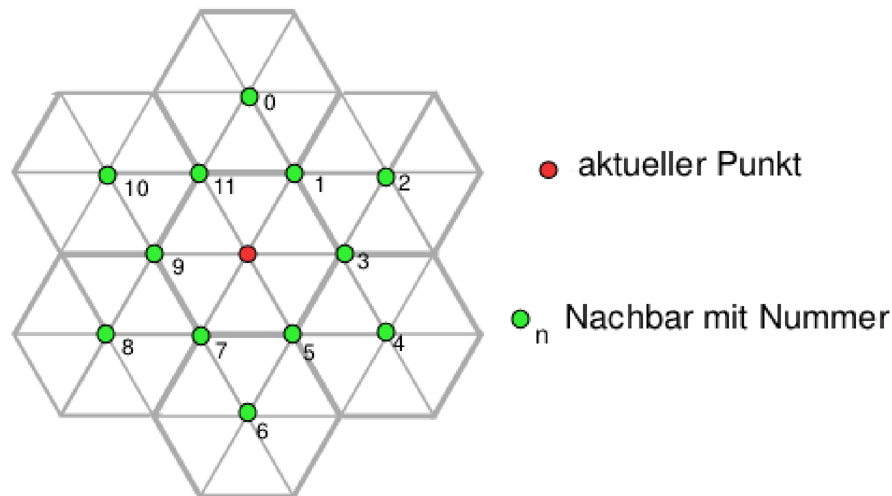


Abbildung 16: Nachbarschaft in einem Secheckfeld

ein Sechseck ergeben, entsteht schnell der Eindruck eines Secheckgitters. Und da daraus die Idee entstanden ist, wurde ein passender Name gewählt. Das Konzept eines Secheckgitters ist folgendes: Das Feld besteht aus regelmäßigen Sechsecken, die wiederum jedes aus sechs gleichseitigen Dreiecken bestehen. Der Punkt in der Mitte eines Sechecks hat zwölf Nachbarn: die Ecken des Sechsecks, in dessen Mitte er liegt und die Mittelpunkte der sechs Sechsecke, die an dieses angrenzen. Dieses Schema ist in Abbildung 16 dargestellt. Jeder der grünen Punkte im Bild hat wiederum zwölf Nachbarn, von denen einer der rote und vier andere grüne sind. Von Nachbar Nr. 9 aus kann man zum Beispiel zu den Punkten 7,8,10,11 und zum roten Punkte gelangen; diese Punkte sind fünf der Ecken des Sechsecks, dessen Mittelpunkt Nr. 9 ist. Man kann auch von 9 zu 5 gelangen, da Nr. 5 der Mittelpunkt eines Sechsecks ist, das an das grenzt, dessen Mittelpunkt Nr. 9 ist.

Bei der Umsetzung in eine numerische Umgebung war darauf zu achten, dass nicht einfach gerundet wird, da es sonst kein echtes Gitter entsteht. In Abbildung 17 ist die Umsetzung zu sehen. Hier ist gegeben, dass der Schritt von a über b genauso nach c führt, wie der Schritt von a direkt nach c. Das ist nicht selbstverständlich. Sei s die Seitenlänge der beteiligten Sechsecke. Der Schritt von a direkt nach c ist also $\binom{s}{0}$, der von a über b nach c ist $\binom{\frac{s}{2}}{\frac{\sqrt{3}}{2}} + \binom{\frac{s}{2}}{-\frac{\sqrt{3}}{2}}$ Würde ein $\frac{s}{2} \notin Z$ in beiden Summanden gleich gerundet, ergäbe die Summe nicht mehr s. Durch die Umsetzung sind die entstehenden Drehungen nicht mehr die ohne Rest durch 30 teilbaren Gradzahlen, die sie noch im Kontinuum waren. Allerdings nähern sie sich daran umso mehr an, je größer s wird.

Die Gitter haben also beide ihre Vor- und Nachteile: Das `CheckerGrid` erzeugt mit hoher Genauigkeit Drehungen um Winkel, die durch 45 teilbar sind - was für den Roboter komplizierter ist als nötig. Das `HexaGrid` wiederum erzeugt zwar günstige Winkel (sie sind alle durch 30 teilbar), aber nur in der Theorie, bei numerisch umgesetzte Pfa-

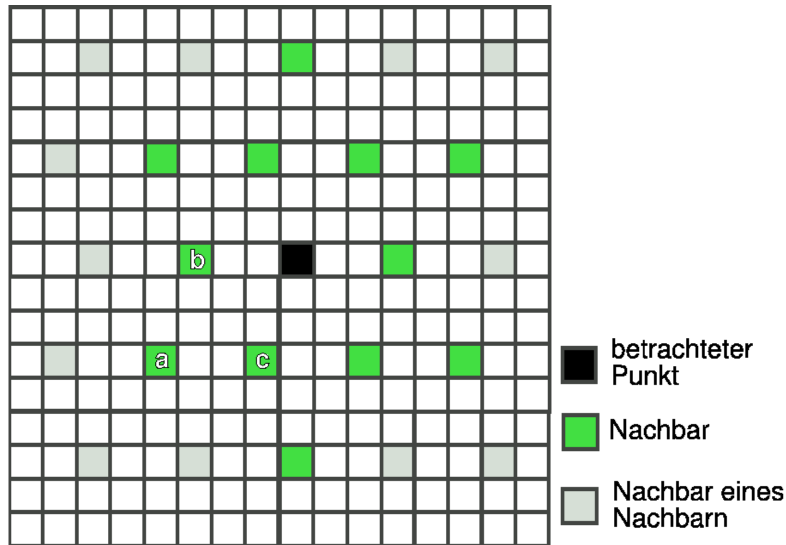


Abbildung 17: Umsetzung des Secheckgitters in Pixelfeld, $s = 3$

de weichen die Gradzahlen leicht ab. Ein Gitter, das die Vorteile beider Ansätze in sich vereint, konnte noch nicht erdacht werden.

Verwendung Bis jetzt werden Gitter nur im Zusammenhang mit dem A-Stern-Algorithmus eingesetzt. Es wäre sicher einen Versuch wert, auch die Newton-Algorithmen mit dieser Einschränkung laufen zu lassen, da sie dadurch weniger Möglichkeiten durchprobieren müssen und schneller werden. Dabei entsteht allerdings das Problem, das Gitter richtig zu wählen: Nicht zu klein, damit es sich lohnt, aber nicht zu groß, damit der Weg zwischen zwei nah beieinander stehenden Hindernissen nicht schon deswegen unmöglich ist, weil dazwischen keine Punkte des Gitters liegen. Man sieht, dass die Wahl einer Gittergröße einfacher ist, je weniger Hindernisse sich in der Umgebung befinden.

Verbesserungsmöglichkeiten In der jetztigen Version sind Gitter in Bezug auf das Koordinatensystem immer gleich ausgerichtet, nämlich wie in den Abbildungen 17 bzw 15. Man könnte es möglich machen, dass Gitter auch gedreht verwendet werden können. Dann könnten sie immer so angelegt werden, dass sowohl Start- als auch Zielpunkt auf dem Gitter liegen. Auch würden in manchen Fällen Umwege vermieden. Allerdings müsste der Roboter bei einem gedrehten Gitter zu Beginn des Pfades eine beliebgradige Drehung machen. Ein weiterer Nachteil ist, dass die Winkel insbesondere bei kleineren Teilpfaden nicht mehr die glatten Zahlen sind, die theoretisch von Wegefinder erzeugt werden können.

Gitter

4.4.3 Sensorwerte

Nach dem erfolgreichen Verbinden mit dem Insekt wird sofort das Sensorfenster angezeigt. Hier werden alle Daten angezeigt, die das Insekt liefert. Diese Daten sind:

- Die Ultraschallwerte (es ist möglich bis zu 16 Werten anzuzeigen)
- Die Neigung (in x-, und y-Richtung)
- Der Kompass
- Der Infrarotsensor
- Die Stromaufnahme der 12 Beinservos
- Die aktuelle Akkuspannung.

Die Bedienung des Fensters ist einfach und eigentlich fast selbsterklärend. Die Akkuspannung wird beim anklicken des „Werte holen"-Buttons immer geholt. Bei allen anderen Werten muß ein Haken im Kasten vor dem Sensornamen gesetzt werden um die Daten zu holen. Stromverbrauch und Ultraschall sind ein- und ausklappbar aufgebaut. Alle Sensorwerte sind mittels Qt als QCheckListItem dargestellt. Bei anklicken des „Werte holen"-Buttons wird die Methode buttonNewValuePressed() aus der SensorGUI-Klasse aufgerufen. Hier wird die Liste neu beschrieben.

4.4.4 Fotofenster

In diesem Fenster werden die Fotos angezeigt. Es is außerdem möglich neue Fotos vom Insekt anzufordern. Dies kann auf verschiedene Weise passieren. Als erste Möglichkeit gibt es die Richtungsbuttons rund um das Foto. Hier bewegt sich das Insekt in die gewünschte Richtung und schießt ein neues Foto.
Die zweite und gleichzeitig anspruchsvollere Methode ein neues Bild anzufordern sind die drei Positionierungsbuttons in der rechten unteren Ecke. Der linke Button sorgt für eine grobe Positionsverbesserung. Hier wurde das Foto in 16 Bereiche eingeteilt. Der Benutzer malt auf das Foto nun ein Rechteck. Je nachdem, welche der Bereiche eingeschlossen oder berührt wurden, wird eine Bewegung des Insekts ausgeführt und danach ein Foto geschossen.
Der mittlere Button ist die genaue Positionsverbesserung. Hier wird wieder ein Rechteck vom Benutzer eingegeben. Nun wird die Mitte des Rechteckes berechnet und das Insekt genau darauf ausgerichtet. Als nächstes wird der Infrarotsensor ausgelöst, und mit dem Rückgabewert und der Größe des aufgezogenen Rechteckes die eine davon abhängende Zahl an Schritten auf das Ziel zugegangen. Abschließend wird wieder das Foto geschossen.
Die letzte Möglichkeit ist die Numberplate-Positionsverbesserung. Wenn ein Nummernschild zu sehen ist, kann man hier die Eckpunkte des Nummernschildes eingeben. Das Insekt versucht nun selbstständig sich auf das Nummernschild auszurichten.
Die drei Positionsverbesserungsmethoden werden vom Benutzer beim ersten klicken

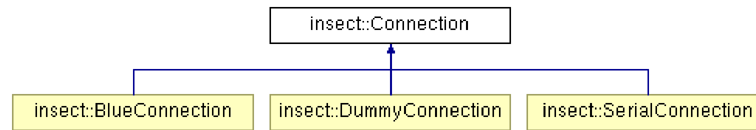


Abbildung 18: Klassendiagramm für Implementierung der Kommunikation

auf das Foto aufgerufen. Es wird dann die Methode mousePressEvent() aufgerufen, die je nach ausgewählter Methode ein Rechteck oder die vier Eckpunkte setzt. Bei der Nummernschild-Auswahl z.B. wird makeNumberplate() und danach moveNumberplate() ausgeführt, die für das zeichnen und die daraus resultierende Bewegung sorgen. Die grobe Neupositionierung führt moveCoarse() aus und die genaue Positionierung moveFine() und danach moveIR(). Die Methode moveFine() bringt das Insekt in Position und löst das Infrarotsignal aus. Mittels moveIR() wird auf den Sensorwert reagiert, die Bewegung ausgeführt und das neue Foto geschossen.

In der linken unteren Ecke befinden sich die Buttons zum Laden und speichern der Fotos. Die Fotos werden in dem Ordner /fotos gespeichert.
Wenn mindestens 2 Fotos geschossen wurden werden die Bildauswahl-Buttons anwählbar. Mit ihnen ist es möglich durch die Historie der geschossenen Bilder zu schalten. Rechts daneben befindet sich die Nummer des Fotos und die Aufnahmezeit.

4.4.5 Kommunikation

Kommunikation Host kann auf zwei verschiedenen Wegen mit dem Insektenroboter kommunizieren: über Bluetooth und über die serielle Schnittstelle mittels eines Nullmodemkabels. Auf dem Host ist dies über eine Schnittstellenklasse `Connection` implementiert (s. Abb. 18). Dieses Interface beinhaltet Methoden zum Senden und zum Empfangen der Nachrichten, die jeweils in den Klassen `BlueConnection` für Bluetooth und `SerialConnection` für serielle Schnittstelle implementiert werden. Die Klasse `InsectCommuThread`, die durch das Verbinden instantiiert wird überwacht ständig, ob die Daten zum Empfangen auf der jeweiligen Schnittstelle vorliegen. Sobald dies der Fall ist, werden sie an die `Connection` übergeben, wo sie gepuffert, ausgewertet und weiterverteilt an die Empfängerklassen werden. Es gibt eine Zeitbegrenzung, die für das Ankommen der Daten gilt, dies ist vor allem beim Empfang von größeren Datenmengen, wie Fotos wichtig.

Die gewünschte Verbindungsart wählt man in dem Dialog „Verbinden“ aus. Nach dem man die benötigten Parameter dort eingegeben und auf „Fertigstellen“ geklickt hat, wird eine Verbindung zum Roboter aufgebaut. Möchte man die Verbindung trennen, kann man es mit dem Aufruf im Hauptmenü „Insekt -> Insektname -> Verbindung beenden“ machen. Momentan ist möglich sich nur mit einem Roboter zu verbinden. Die Software durfte jedoch relativ problemlos auf die Verbindung mit mehreren Robotern zu erweitern sein.

Die Klasse `DummyConnection` umgeht das Aufbauen einer Verbindung zum Ro-

boter. Dies ist nur dazu nötig, um mit dem Programm arbeiten zu können ohne den Roboter anzusteuern.

Für die Kommunikation mit dem Roboter existiert ein von uns entwickeltes, einfaches Kommunikationsprotokoll. Dieses ist im Kapitel 4.3.1 im Abschnitt **Protokoll** beschrieben.

Kommunikation über die serielle Schnittstelle Eine Möglichkeit sich mit dem Roboter zu verbinden ist über die serielle Schnittstelle des Rechners und die des Roboters mit Hilfe eines Nullmodemkabels. Die Verbindung über Nullmodemkabel war in der Anfangsphase der Implementierung realisiert, da es keine andere Möglichkeit zum Kommunizieren mit dem Entwicklungsboard gab. Sobald Hardwareseitig auf dem Roboter Bluetooth implementiert war, wurde die Kommunikation über die serielle Schnittstelle nicht mehr weiterentwickelt. Die Implementierung der seriellen Verbindung befindet sich in der Klasse `SerialConnection`.

Kommunikation über Bluetooth Die zweite und sicherlich interessantere Möglichkeit zur Kommunikation mit dem Roboter ist über Bluetooth. Die Implementierung für diese Verbindungsart ist in der Klasse `BlueConnection` festgeschrieben. Diese nutzt die Verbindung über das **rfcomm**-Layer der Bluetooth-Implementierung für Linux **BlueZ**. Bevor man sich über Bluetooth mit dem Roboter verbindet, kann man sich in dem Dialog „Verbinden“ alle BT-Geräte in Reichweite ausgeben lassen und dann das richtige aus der Liste auswählen. Die ist möglich unter Verwendung des HCI-Layers des BlueZ. Weitere Informationen über **BlueZ**, **rfcomm** und **HCI** können der zahlreichen Dokumentation aus dem Internet entnommen werden.

Befehle Daten, die vom Host aus an des Roboter gesendet werden sind in erster Linie Befehle. Diese können in zwei Gruppen unterteilt werden: die für die Steuerung und Bewegung des Roboters verantwortlich sind und die Befehle, die verschiedene Informationen vom Roboter anfordern. Alle Bewegungsbefehle und einige Informationsbefehl können aus der Steuerungs-GUI per Knopfdruck versendet werden. Dabei kann man bei denn Bewegungsbefehlen noch auswählen, ob sie in eine Abarbeitungsschlange kommen oder sofort ausgeführt werden sollen. So, dass die Befehle, die davor gesendet wurden und evtl. noch ausgeführt werden unterbrochen und durch neue Order ersetzt werden. Informationsbefehle, zum Auslesen von Sensordaten oder zum Aufnehmen der Fotos werden sofort ausgeführt. Andere Möglichkeit ist die Befehle über eine Befehlskonsole einzugeben. Eine Auflistung der Befehle erhält man, wenn man als Befehl *help* eingibt, sowie aus der Dokumentation für die Insektensoftware. Einige spezielle Befehle, die zum Einstellen verschiedener Funktionen des Roboters benötigt werden, kann man zur Zeit nur über die Konsole versenden.

Während der Ausführung eines Bewegungsbefehls versendet der Roboter Meldungen über den aktuellen Status des Ausführens. Nach dem Versenden eines Informationsbefehls erwartet der Host die angeforderte Information vom Roboter. Dies können

die Sensorenwerte oder Fotos sein. Von selbst sendet der Roboter nur das Notstop Signal. Alle ankommenden Daten werden auch auf der Konsole ausgegeben.

Die Verarbeitung von Befehlen ist in den Klassen `Command`, `CmdManager`, `CmdQueueCtrl` implementiert. Befehle werden kodiert an den Roboter übertragen. Die Befehle selbst und deren Kodierungen findet man in einer gemeinsam von Host- und von Robotersoftwareprogrammierern genutzten Datei `insektkommandos.h`. Man kann relativ einfach die Software auf weitere Befehle erweitern. Auf der Konsole kann man die Befehlskodes auch direkt 'Raw' eingeben, so dass sie genau so wie eingegeben auch an den Roboter gesendet werden.

Falls der Host von dem Roboter keine Bestätigung über den Empfang des Befehls erhält, unternimmt er 5 weitere Versuche, mit etwas zeit dazwischen, das Kommando doch noch zu verschicken. Danach gibt er eine Fehlermeldung aus und stoppt die Warteschlange.

Verbesserungen und Erweiterungen In der Behandlung der Befehle kann man noch viele Verbesserungen durchführen. So sind z.B. nicht alle Befehle über GUI ausführbar. Eine intelligentere Warteschlange, in der nicht nur Bewegungsbefehle abgelegt werden können wäre eine weitere nützliche Erweiterung.

Die Kommunikation über Bluetooth ist nicht immer zuverlässig. Mit steigender Reichweite häufen sich die Übertragungsfehler. Und die Übertragungs- und Kommunikationsfehler sollten noch besser abgefangen und behandelt werden als in der jetzigen Version des Programms. Ein neues robusteres Kommunikationsprotokol wäre ein weiterer Verbesserungsvorschlag.

Es gibt sicherlich noch viel mehr, was auf der Basis der momentanen Implementierung noch weiter verbessert und erweitert werden kann.

5 Projektablauf

In diesem Abschnitt wird vorgestellt, wie die Projektgruppe die Realisierung der gestellten Aufgabe geplant hat, wie sie diese Aufgabe in Teilaufgaben zerlegt hat, wie sie Zeitpunkte der Fertigstellung bestimmt hat, und letztendlich, wie diese Teilaufgaben in Wirklichkeit bewältigt wurden.

Zunächst wird die Organisation von gemeinsamen Treffen beschrieben. Darauf folgt die Vorstellung eines groben Projektplans. Abschließend werden Meilensteintrenddiagramme vorgestellt. Anhand dieser wird erklärt, wo Schwierigkeiten aufgetreten sind und wie sich diese auf die Planung ausgewirkt haben.

5.1 Regelmäßiges Treffen

Nachdem die erste Seminarphase gelaufen war, traf sich die Gruppe am 30. April 2004 zum erten Mal, um mit der Lösung der gestellten Aufgabe zu beginnen. Es gab viele unklare Punkte, denn die Aufgabe war allgemein gestellt: man sollte einen Insektenroboter entwickeln. Solche Fragen wie „Welche Bestandteile nehme ich?“, „Was dürfen sie

kosten?", „Wo kann ich sie beschaffen?" war die erste Konfrontation mit der Aufgabe. Vor allem Kosten- und Dimensionsfragen waren von entscheidender Bedeutung. Denn das Budget war auf 500 € begrenzt und unser Roboter durfte nicht zu groß werden, um vom 4WD-Roboter transportiert werden zu können. So beschloss die Gruppe, sich zwei Mal pro Woche zu treffen, bis diese heiklen Fragen beantwortet werden konnten.

Einmal in der Woche hat unser Betreuer, Andreas Schallenberg, bei den Treffen zur Seite gestanden und uns bei vielen Entscheidungen Hilfe geleistet. Manchmal ist Herr Nebel vorbeigekommen, um unsere Fortschritte zu sehen. Fast vier Monate lang hat sich die Gruppe zwei Mal pro Woche getroffen, bis letztendlich Ein Dokument vorlag, welcher diese Fragen beantwortete, der Entwurf.

Ab Mitte August traff man sich nur ein Mal pro Woche. Denn von nun an gab es viel zu tun: die Implementierungsphase hat angefangen.

5.2 Projektplan

Der erste Projektplan entstand am Ende der Anforderungsdefinition-Phase. Er ist in der Abbildung 19 zu sehen. Im Folgenden werden einzelne Phasen des Projekts näher vorgestellt. Dabei werden auch Zuständigkeiten der Gruppenmitglieder genannt.

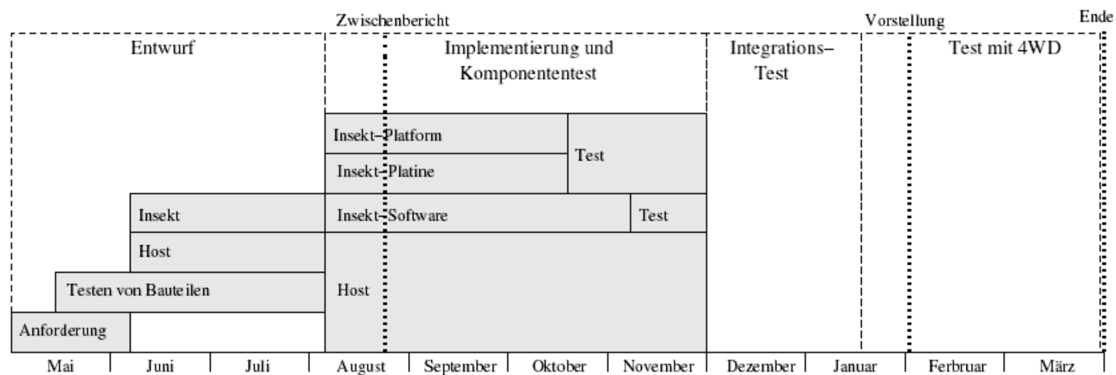


Abbildung 19: Grober Projektplan vom 8. Juni 2004

5.2.1 Anforderungsdefinition

Die Anforderungsdefinition kam in drei Zyklen zustande. Die erste Version erschien Mitte Mai 2004. Als Antwort darauf kam von den Betreuern eine Liste mit Anmerkungen. Die zweite Iteration endete Anfang Juni, wieder gefolgt von Anmerkungen der Betreuer. Während die zweite Version der Anforderungsdefinition noch überarbeitet wurde, fing die Gruppe bereits mit der Entwurf-Phase an. Die letzte Version der Anforderungsdefinition erschien am 1. Juli 2004.

Projektbudget Die Anforderungsdefinition enthielt zusätzlich zu den Aufgabenstellungen eine Liste mit technischen Daten der Bestandteile und eine grobe Kostenschät-

zung. Nach dieser Kostenschätzung würde das zur Verfügung stehende Budget um 387 € überschritten werden. Unsere Kostenschätzung ging an die EWE-Projektförderung. Leider hat das Projekt keine Unterstützung von EWE bekommen. Zum Glück konnte unser Budget doch noch auf 1000 € vergrößert werden, so dass wir keine Einsparungen an Bestandteilen vornehmen mussten.

Zum Ende des Projekts wurde tatsächlich 665,04 € für den Insektenroboter ausgegeben. Hier sind aber keine Versandkosten enthalten, wie es im Kostenplan der Fall war. Ohne Versandkosten wurde eine Summe von 837 € erwartet. Dank der niedrigeren Servo-Preisen konnte etwas eingespart werden.

Zusätlich wurden aber noch 6 Servos bestellt, weil sie unerwartet schnell defekt wurden. Und ein neuer Ultraschallsensor musste besorgt werden, weil sein Vorgänger aus einem unerklärten Grund einen Elektronikfehler bekam. Somit belaufen sich die Gesamtkosten für den Insektenroboter auf 780,26 €. Das liegt immer noch unterhalb der eingeplanten Ausgaben.

Zuständigkeiten In der Anforderungsdefinition-Phase traten zwei Aufgabenabschnitte auf. Im ersten Abschnitt wurde Information über Sensoren, Aktoren, Plattform und elektronische Bauteile gewonnen. Dabei sah die Aufgabenverteilung folgendermaßen aus:

- **Sensoren:** Carsten
- **Bluetooth:** Jann
- **Akkus:** Dimitri
- **Elektronik:** Henning
- **Plattform-Bausätze:** Michael
- **Servos:** Dietrich
- **Gangarten:** Anton

Im zweiten Abschnitt beschäftigte sich die Gruppe mit Entscheidungen und Niederschreiben für die Anforderungsdefinition zu diesen Themen:

- **Kartographie/Navigation:** Dietrich, Dimitri, Jann
- **Insektenplattform:** Anton, Carsten, Gerold
- **Platine:** Henning
- **Host-System:** Michael
- **Code/Test:** Andreas, Martin

5.2.2 Entwurf

Die Entwurfsphase begann Anfang Juni und sollte 2 Monate dauern. Diese Planung wurde zum größten Teil eingehalten. Es wurden nach dem Zieltermin am 8. August noch einige kleine Änderungen gemacht, während die meisten Gruppenmitglieder mit der Implementierung begonnen haben. Die korrigierte Version des Entwurfs wurde am 19. August veröffentlicht.

Zuständigkeiten Der Entwurf wurde in vier große Teilbereiche aufgeteilt: Plattform, Platine, Roboter-Software und Host-Software. Die Ausarbeitung der einzelnen Bereiche geschah wie folgt:

- **Plattform**
  - Dietrich
- **Platine**
  - Henning
- **Roboter-Software**
  - **FPGA:** Gerold
  - **Sensoren:** Carsten
  - **Aktoren:** Dimitri
  - **Kommunikation:** Andreas
- **Host-Software**
  - **Robotersteuerung:** Jann
  - **Sensordaten, Fotos:** Michael
  - **Kommunikation:** Martin
  - **Kartographie, Navigation:** Anton

Testen von Bauteilen Im Projektplan in Abbildung 19 ist ein Bereich „Testen von Bauteilen" in der Entwurfsphase zu sehen. Hier handelt es sich um einige hilfreiche Tests, die unsere Entscheidungen beeinflusst haben. Es wurde zum Beispiel ein Stromaufnahme-Test von zwei Servos durchgeführt (Abbildung 4). Hiermit wurde bestätigt, dass über die Messung der Stromaufnahme von Servos auf Bodenkontakt geschlossen werden kann. Somit hat die Gruppe entschieden, keine Druckkontakte an Beinen zu verwenden. Stattdessen sollte über eine elektronische Schaltung Stromfluss durch Servos gemessen werden.

5.2.3 Implementierung

Die Implementierung hat Anfang August begonnen. Hier wurde wie im Entwurf nach vier großen Arbeitsbereichen unterschieden. Die Aufgabenzuteilung wurde bei der Implementierung beibehalten. Lediglich in einigen Aufgaben gab es Abweichungen von diesem Plan.

Meilensteinplan Zu Beginn der Implementierungsphase wurde ein Meilensteinplan aufgestellt. Er enthielt 42 Meilensteine mit 102 Arbeitspaketen. Der abgeschätzte Aufwand belief sich auf 1186 Manntage.

Zu den Meilensteinen wurde jeweils ein Erfüllungskriterium und geschätztes Datum der Fertigstellung angegeben. Jeder Meilenstein setzte sich aus einigen Arbeitspaketen zusammen. Zu jedem Arbeitspaket wurde Aufgabe, Ergebnis, abgeschätzte Dauer, empfohlene Anzahl von Personen und Bearbeiter eingetragen. Außerdem sollte jedes Arbeitspaket Abhängigkeiten von anderen Meilensteinen oder Arbeitspaketen angeben.

Das Eintragen von Meilensteindaten sollten die Zuständigen der Aufgaben jeder für seinen Teil übernehmen, so nach Gruppenbeschluss. Wie es sich herausgestellt hat, war das keine gute Entscheidung. So wurde oft der Aufwand unterschätzt und die Dauer der Arbeitspakete zu optimistisch angegeben. Es wurden Abhängigkeiten nicht erkannt oder blieben bei Angaben von Fertigstellungsdaten unberücksichtigt.

Meilensteinplanung am besten einer Person überlassen

Erst nach einigen Monaten hat der Meilensteinbeauftragte, Dimitri, die Pflege des Meilensteinplans vollständig übernommen und Abhängigkeiten sowie Arbeitsaufwand überprüft und ergänzt. So kam einige bittere Wahrheit ans Licht, dass einige Meilensteine gar nicht so früh gelegt werden sollten, weil sie von anderen abhingen, welche selbst erst spät erreicht werden konnten.

Im Abschnitt 5.3 wird die Entwicklung der Meilensteine anhand Meilensteintrenddiagramme erklärt. Für nähere Informationen zu Meilensteinen wird auf Meilenstein-Dokument verwiesen.

5.2.4 Zwischenbericht

Die erste Version des Zwischenberichts wurde Mitte September erstellt. Nach dem Projektplan in Abbildung 19 sollte er bereits Ende August erscheinen. Da die Gruppe erst Anfang September nähere Informationen zum Inhalt des Berichts erhalten hat, kamm es zu dieser Verspätung.

Die letzte Version des Zwischenberichts erschien am 4. November 2004, nachdem die zweite Seminarphase gelaufen war. Die Bearbeitung des Berichts wurde vollständig von Jann übernommen.

5.2.5 Integration und Test

Im Projektplan wurde Integration der Roboter- und Host-Systeme von Anfang Dezember bis Mitte Januar vorgesehen. So genau lässt sich diese Phase jedoch nicht identifi-

zieren. Die Integration fand nicht als ein unteilbares Stück statt, sondern ist allmählich eingetreten.

Die Implementierung, Integration und Test begannen ab Dezember immer mehr zu verschmelzen. Der Schwerpunkt verlagerte sich mit der Zeit zum Test. Das Ende der Integrations- und Testphase rückte aber stets weiter an das Ende des Projektes heran. Der Hauptgrund dafür war die Verzögerung bei der Fertigung der Platine.

5.2.6 Test mit 4WD

Die Phase „Test mit 4WD" ist leider komplett ausgefallen. Das lag daran, dass die 4WD-Gruppe uns keine Rampe zur Verfügung gestellt hat.

5.2.7 Zusammenfassung

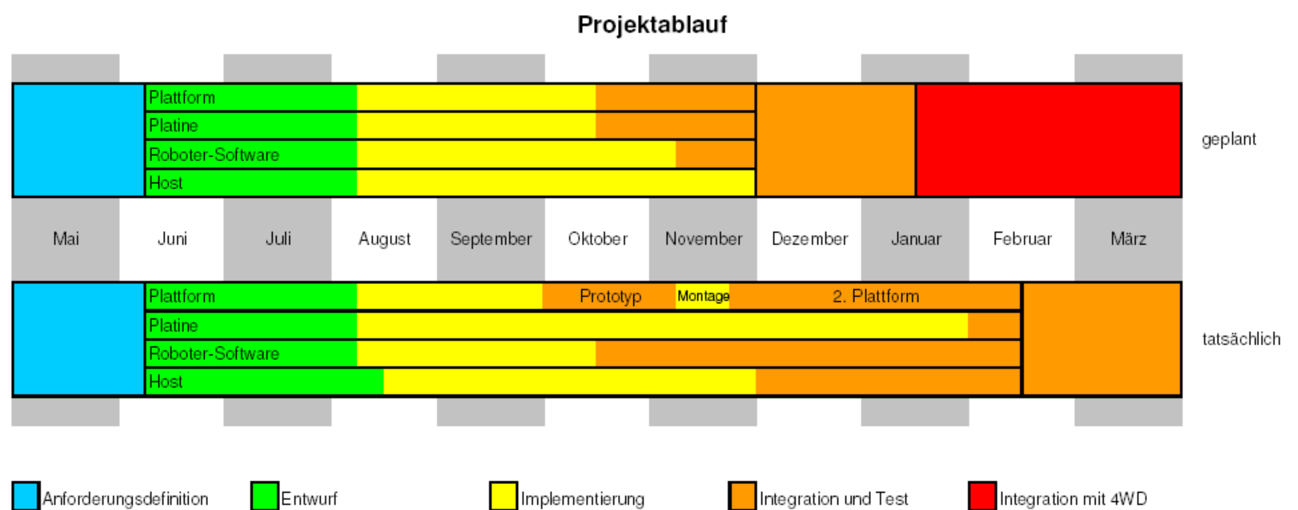


Abbildung 20: Geplanter und tatsächlicher Ablauf des Projekts

Die Abbildung 20 stellt einen Vergleich zwischen dem geplanten und dem tatsächlichen Ablauf des Projekts dar. Wie man sieht, entspricht der tatsächliche Ablauf nicht dem geplanten. Es kam zu Verschiebungen der Phasen in Richtung Projektende. Besonders große Verzögerung entstand bei der Platine. Der Aufwand war viel größer als erwartet, und die Erfahrung hat gefehlt; denn dieser Bereich war für jeden von uns ein unbekanntes Terrain. Die späte Platinenfertigung hat die Integrationsphase stark nach hinten verschoben. Aber nicht nur an der Platine hat es gelegen. Es gab immer wieder unerwartete Probleme und Fehler, welche unseren Zeitplan durcheinander gebracht haben, so zum Beispiel die Implementierung vom I^2C-Bus auf dem Roboter oder ein plötzlicher Ausfall der Plattform wegen defekter Servos.

Man könnte aber auch spekulieren, dass das Projekt trotzdem in Verzug geraten würde, wenn die Platine rechtzeitig fertig wäre. Denn man kann nich genau sagen, ob alle

anderen Teilaufgaben zum geplanten Fertigungstermin der Platine auch einsatzbereit wären. Da die Platine erst spät einsatzbereit war, fällt es kaum ins Licht, dass andere Komponenten vielleicht auch später implementiert wurden.

5.3 Meilensteinentwicklung

In diesem Abschnitt wird die Entwicklung von Meilensteinplanung vorgestellt. Hier wird erklärt, warum es zu Verschiebungen der Fertigstellung bei vielen Meilensteinen gekommen ist. Für nähere Informationen zu Meilensteinplanung siehe einzelnes Dokument „Projektplan“.

5.3.1 Ausgewählte Meilensteine

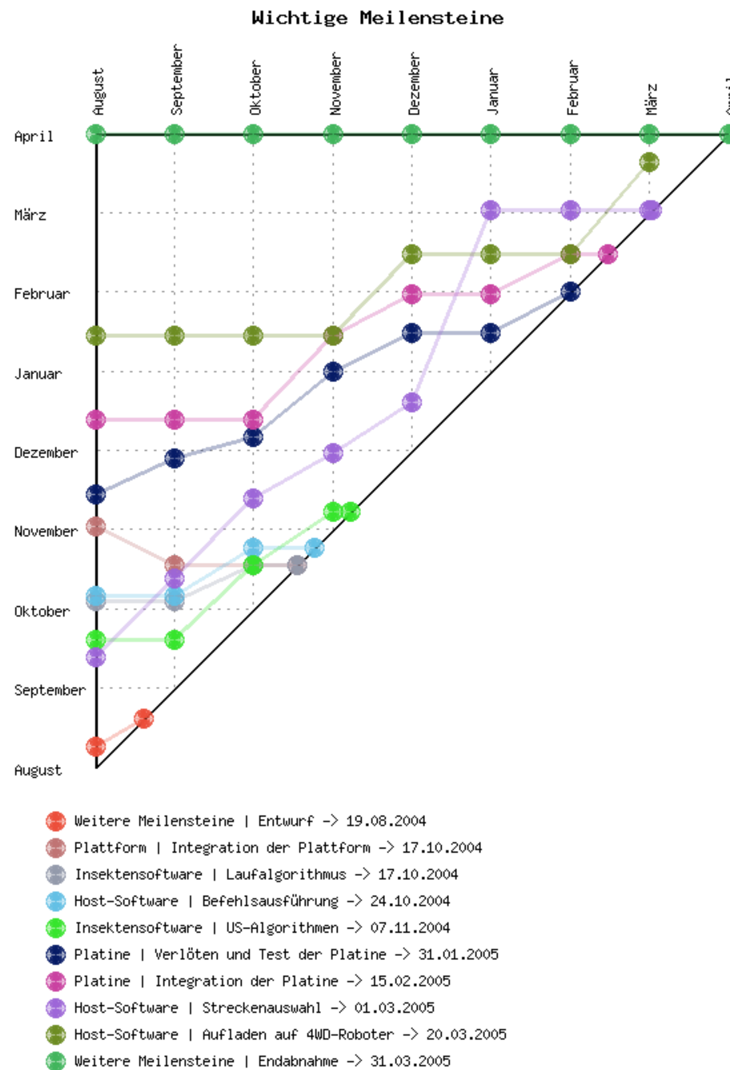


Abbildung 21: Ausgewählte Meilensteine

Die Abbildung 21 stellt einige wichtige Meilensteine für das Projekt dar. Hier sind aussagekräftige Meilensteine aus vier Arbeitsbereichen enthalten: Plattform, Platine,

Roboter-Software und Host. Außerdem sind Phasen-Meilensteine Entwurf, Aufladen auf 4WD-Roboter und Endabnahme dargestellt. Es ist eine permanente Verschiebung besonders in Meilensteinen Verlöten und Test der Platine, Integration der Platine und Streckenauswahl zu erkennen. Die Verschiebungen bei den Platine-Meilensteinen entstanden wegen des unterschätzten Aufwandes bei der Entwicklung von Platinen. Es fehlte an Erfahrung und Kenntnissen in diesem Bereich. Die Verschiebung bei der Streckenauswahl ergab sich durch Verspätung der Kartendarstellung.

5.3.2 Plattform und Platine

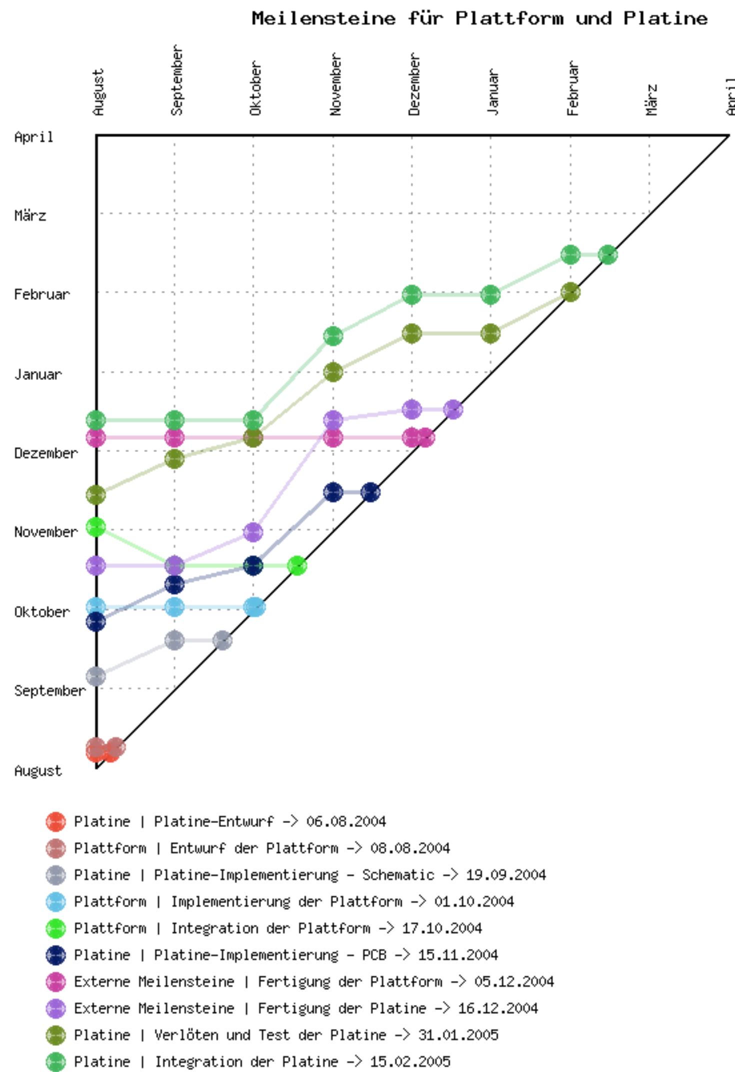


Abbildung 22: Meilensteine für Plattform und Platine

Bei der Plattform-Meilensteinen kam es zu keiner Verzögerunge, im Gegensatz. Integration der Plattform war früher als geplant abgeschlossen. Das liegt daran, dass an der Erstellung der Plattform außer Dietrich zusätzlich noch Jann und Michael gearbeitet haben.

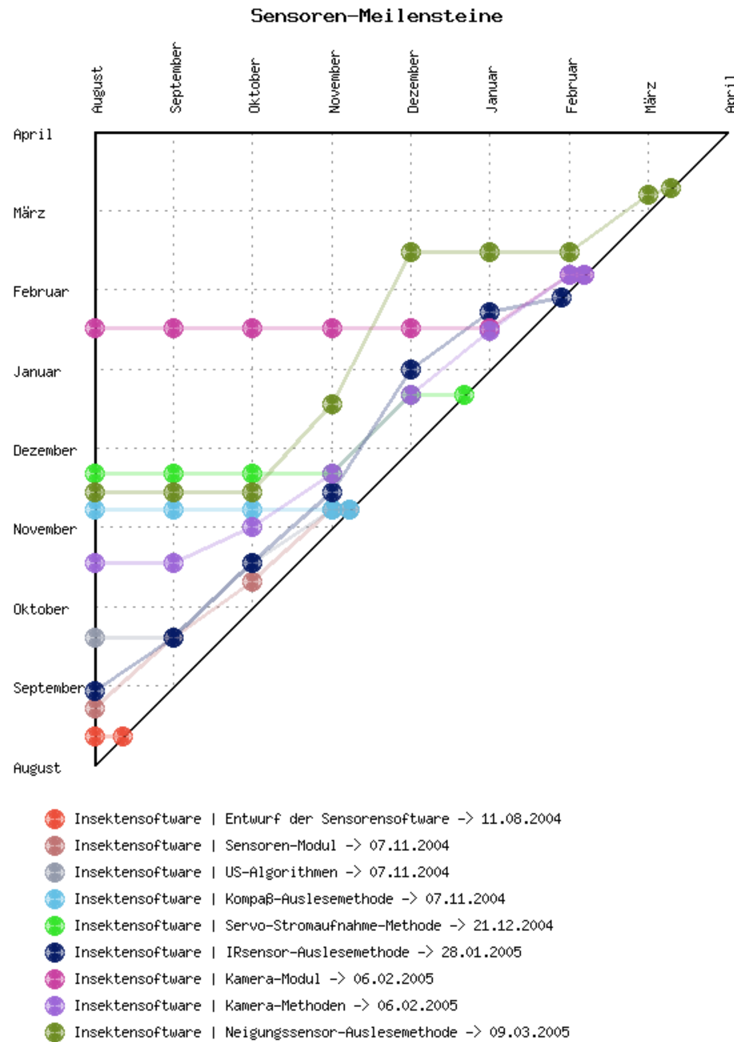


Abbildung 23: Sensoren-Meilensteine

Bei der Platine ist die Planung nicht so günstig ausgefallen. Der Aufwand bei der Platine-Entwicklung sorgte für Rückschläge in der Planung. Henning hat daran allein gearbeitet und hatte keine große Erfahrung daran, wie jeder andere in der Gruppe auch. Die größte Herausforderung war dabei das Routen von Leiterbahnen. Zunächst wurde versucht, mit einer doppellagigen Platine klarzukommen, doch der Versuch ist gescheitert. Später wurde das Routen auf eine vierlagige Platine wiederholt. Dies führte zu einem Zeitverlust.

5.3.3 Sensoren

Die Verzögerungen bei den Sensoren-Meilensteinen lassen sich auf zwei Probleme zurückführen. Die erste Schwierigkeit war die Implementierung von I^2C-Bus. Carsten und Gerold ist lange Zeit nicht gelungen den Bus zum Laufen zu bekommen. Unterschiedliches Signal-Laufzeitverhalten auf dem FPGA führte in eine Sackgasse. Erst nachdem sie ein anderes Vorgehen einsetzten, konnten sie das Sensoren-Modul erfolgreich implementieren. Somit konnten bald Ultraschall- und Kompasssensor ausgelesen werden.

Die zweite Schwierigkeit bereitete der DMA-Kontroller, welcher zum Übertragen von Bildern aus der Kamera direkt in den Speicher benutzt werden sollte. Unzureichende Dokumentation und fehlerhafte Quartus-Software ist hier zum Stolperstein geworden.

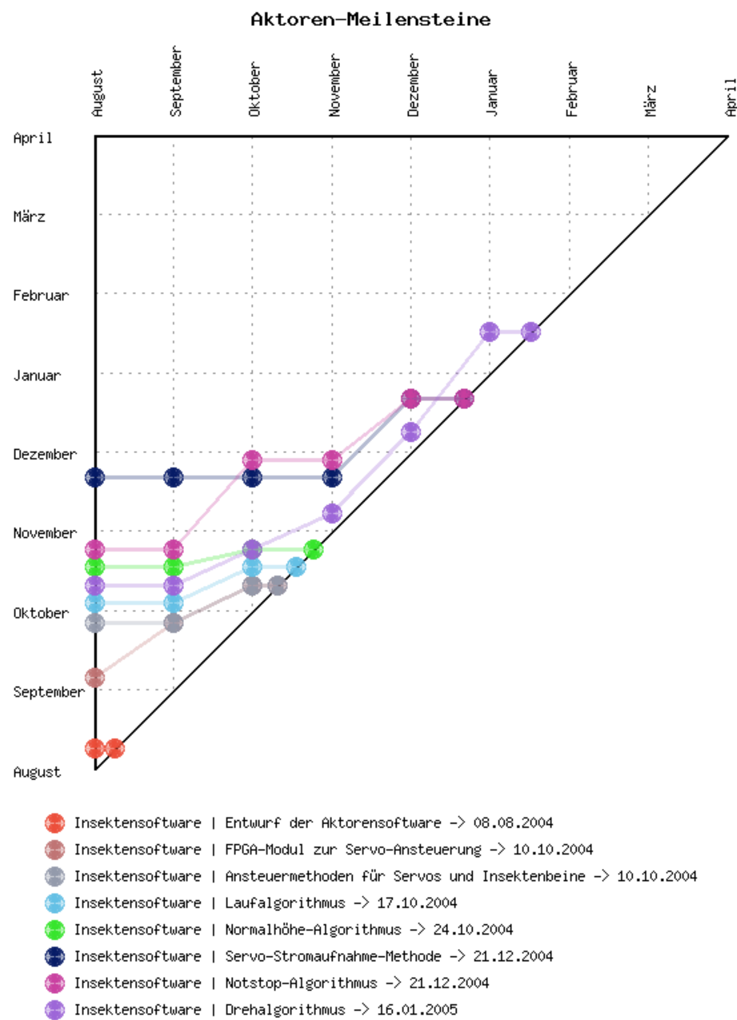


Abbildung 24: Aktoren-Meilensteine

Die Verzögerungen beim Infrarot-Sensor und bei der Stromaufnahme von Servos sind auf die oberen Probleme indirekt zurückzuführen. Da Carsten und Gerold mit Bewältigung dieser Probleme beschäftigt waren, hat sich niemand zunächst um Infrarot-Sensor und Stromaufnahme-Methoden gekümmert. Später hat Dimitri sich damit auseinandergesetzt.

Die Implementierung von Neigungssensor-Methode konnte erst fertiggestellt werden, nachdem die Platine bereitstand. Da Platine erst spät fertig wurde, ist auch hier zur Verzögerung gekommen.

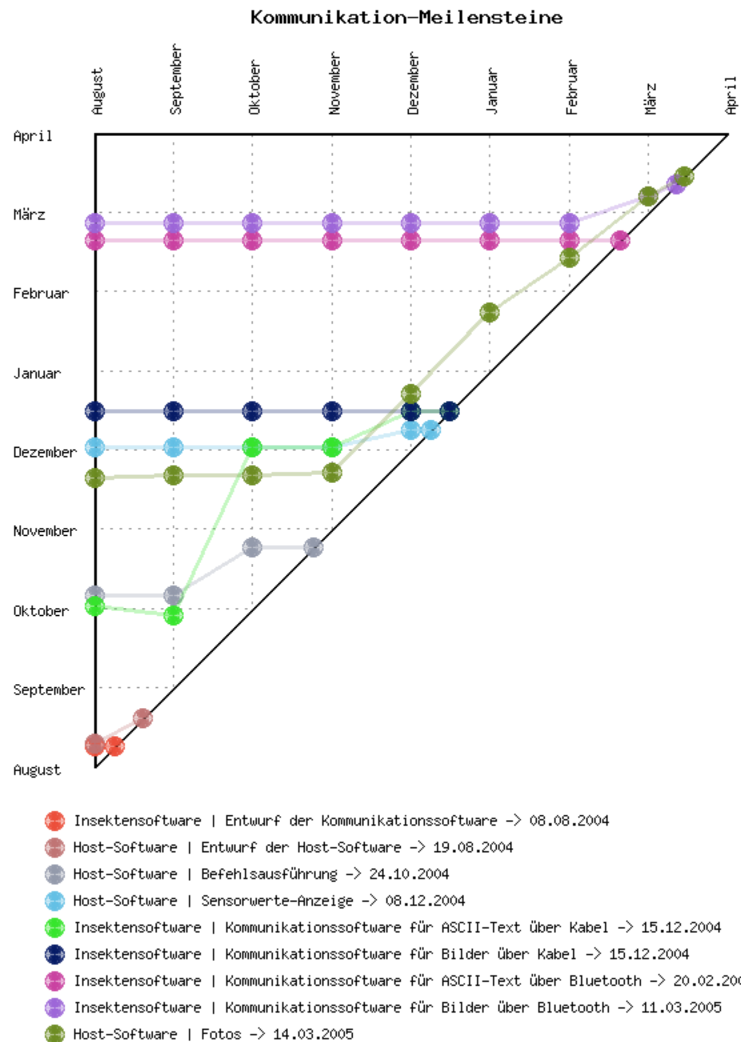


Abbildung 25: Kommunikation-Meilensteine

5.3.4 Aktoren

Die Verschiebung beim Drehalgorithmus entstand, weil im Laufe der Implementierung eine andere Methode zum Drehen entdeckt wurde. Die Lösung wurde komplexer und erforderte mehr Zeit. Außerdem war Dimitri gleichzeitig mit der Messung der Stromaufnahme von Servos und dem Infrarot-Sensor beschäftigt.

Der Notstop-Algorithmus konnte erst implementiert werden, nachdem die Stromaufnahme von Servos gemessen werden konnte.

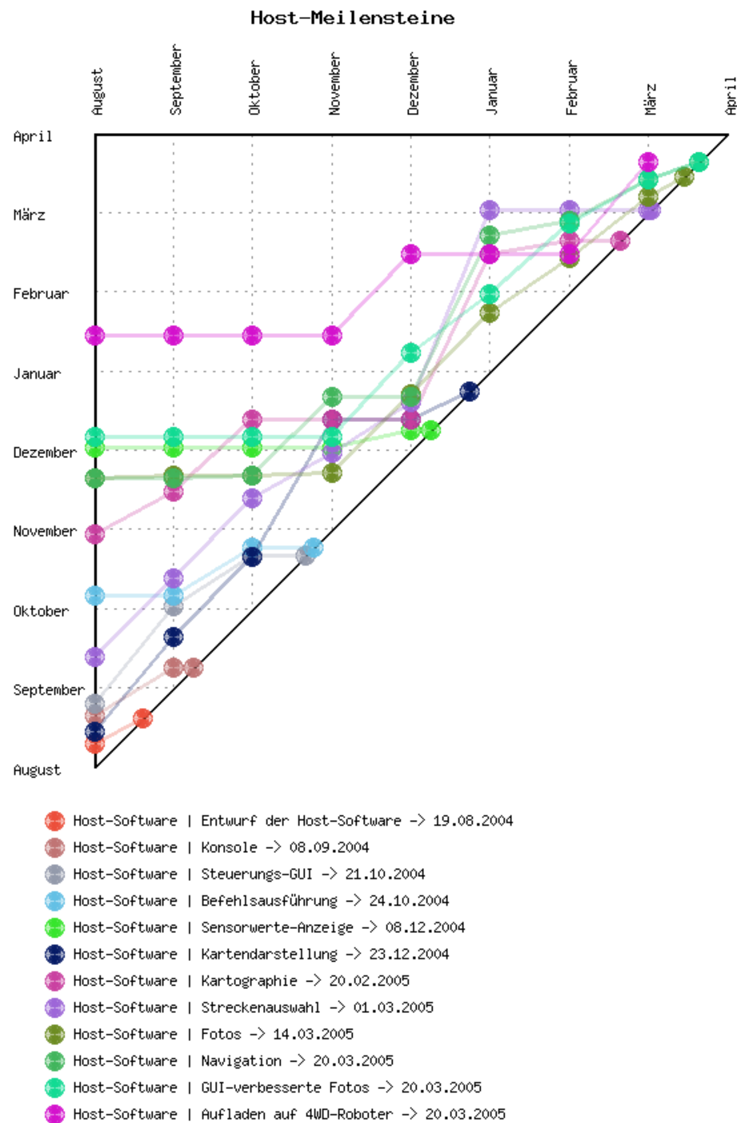


Abbildung 26: Host-Meilensteine

Die Fertigstellung vom FPGA-Modul zur Servo-Ansteuerung hat sich verzögert, weil

Carsten und Gerold mit dem I^2C-Bus beschäftigt waren. Das Modul hat später Dimitri implementiert.

5.3.5 Kommunikation

In der Kommunikation sind keine besonderen Verzögerungen aufgetreten. Leider konnte das Übertragen von Bildern erst spät getestet werden, weil es Schwierigkeiten mit dem I^2C-Bus auf der Platine gegeben hat.

5.3.6 Host

Einige Host-Meilensteine konnten nicht rechtzeitig eingehalten werden, weil die Platine erst spät fertig wurde. Es kam auch zu Fehlern im Meilensteinplan, da einige Abhängigkeiten übersehen worden sind, so zum Beispiel bei Navigation.

Die Streckenauswahl verzögerte sich wegen fehlender Kartendarstellung. Kartographie konnte schlecht ohne richtige Daten vom Roboter getestet werden. Da von 4WD-Gruppe keine Karte erzeugt werden konnte, mussten Anton und Jann selbst Testdaten erzeugen. Erst nachdem Scannen der Umgebung mit dem Infrarot-Sensor implementiert wurde, konnte mit realistischen Daten getesten werden.

Die Verzögerungen beim Host sind auch darauf zurückzuführen, dass an der Host-Software zunächst nur drei Gruppenteilnehmer gearbeitet haben: Anton, Jann und Michael. Erst später ist Dietrich zur Hilfe gekommen und hat die Kommunikation auf dem Host gepflegt.

Aufladen auf 4WD-Roboter konnte leider nicht erreicht werden, da 4WD-Gruppe keine Rampe zur Verfügung gestellt hat.

6 Bewertung des Systems

Dieses Kapitel soll nur einen kurzen Überblick darüber geben, was realisiert wurde und was nicht. Es ist für folgende Projektgruppen sinnvoll um einen ersten Überblick zu erhalten. Insbesondere der Abschnitt „Bekannte Probleme“ soll aufzeigen worauf in Zukunft geachtet werden sollte.

6.1 Roboter

6.1.1 Erfolge

Im Prinzip sind alle Anforderungen des Entwurfs realisiert worden.
Der Roboter kann kurz gesagt vom Host gesteuert laufen, die Umgebung kartieren, Fotos machen und auf Hindernisse reagieren. Die Kommunikation läuft über eine genau definierte Schnittstelle über die Kommandos an den Roboter gesendet und Sensorwerte bzw. Zustände abgefragt werden können. Die Befehlsmenge ist leicht erweiterbar, da genau für jeden Befehl eine Methode existiert (siehe Kapitel Kommunikation). Normalerweise läuft die Verbindung über Bluetooth, es kann aber auch zu Testzwecken bespielsweise ein 0-Modem-Kabel verwendet werden.
Aus der Not entstanden ist die Komprimierung der Bilder. Ursprünglich war dies nicht geplant. Die eingeschränkte Bandbreite bedingt durch den UART zwang uns dazu die Datenmenge zu minimieren (siehe Kapitel Komprimierung der Bilder).
Das Erkennen der Hindernisse mit dem Ultraschall- und Infrarot-Sensor funktioniert sehr gut. Der Bewegungsapparat, drehen, aufen funktioniert abgesehen von dem Problem mit den Servos zu unserer Zufriedenheit (siehe Bekannte Probleme).

6.1.2 Bekannte Probleme

Die Realisierung der Rampe war durch das Fehlen der Auflademöglichkeit seitens der 4WD-Gruppe nicht möglich. Auch die Kommunikation mit dem 4WD-Roboter existiert nicht.
Ein weiteres Problem stellte die Übertragung per Bluetooth dar. Wir setzten vorraus, dass ein 100m-Bluetooth-Modul (BTM) auch über eine Distanz von 100m Daten übertragen kann. Ernüchternd mussten wir jedoch feststellen, dass dies nicht der Fall ist. Ab einer Distanz von etwa 10m brach die Kommunikation teilweise ab. Es wird wohl daran liegen, dass das BTM unterhalb der Platine befestigt ist und somit nur seitlich abstrahlen kann. Dies war auch eigentlich nicht so geplant. Ursprünglich sollte das BTM oben auf der Platine sitzen. Durch einen Fehler auf der Platine war dieser Stecker falsch belegt und das BTM musste schließlich via Erweiterungsplatine auf der unteren Seite angeschlossen werden.
Das größte Problem stellten allerdings die Servos dar. Es war eine Vorgabe den Roboter generell klein zu halten. Aus diesem Grund haben wir auch die kleinsten Servos benutzt. Da Servos aber offensichtlich nicht dafür gebaut sind Beine von Robotern zu bewegen, gingen diese ständig kaputt. Verschärft wurde diese Problematik durch

den Einsatz von Plastikgetrieben. Diese sind zwar preiswert, qualitativ aber minderwertiger als die teureren Metallgetriebe. Eine zukünftige Projektgruppe sollte dies berücksichtigen und eventuell überdenken. Insbesondere weil ein Roboter der nicht läuft einen schlechten Gesamteindruck erzeugt, selbst wenn alle anderen Funktionalitäten korrekt arbeiten.

6.2 Host

6.2.1 Erfolge

Auch auf der Host-Seite ist im Prinzip alles was im Entwurf geplant war, auch umgesetzt worden. Die Präsentation mittels einer GUI, die Kommunikation mit dem Insekt, die Navigation des Insekts, die Kartenhaltung und das Anzeigen der Fotos und Sensorwerte. Für alle diese Aufgaben gibt es eigene Fenster.
Während der Programmierung sind uns noch weitere Zusatzfunktionen in den Sinn gekommen, von denen einige umgesetzt wurden und andere nicht mehr realisiert werden konnten. Auf der Karte sollten auch die Stellen markiert werden an denen ein Foto geschossen wurde. Durch einen Klick auf diesen Punkt hätte das Foto nochmals angezeigt werden sollen. Diese Funktion wäre nett gewesen, aber leider konnten wir es in der Zeit nicht mehr realisieren.
Was zusätzlich von uns noch eingeführt wurde ist die Anzeige der restlichen Akkuspannung des Insekts. Außerdem haben wir nicht nur einen Wegfindungsalgorithmus wie geplant, sopndern können zwischen 4 verschiedenen Algorithmen auswählen, die sich in der Qualität des berechneten Weges und der benötigten Rechenzeit doch erheblich unterscheiden.

6.2.2 Bekannte Probleme

Nach dem Programmende wird ein Speicherzugriffsfehler gemeldet. Er wirkt sich während der Ausführung nicht aus. Nachdem wir uns mit dem Problem befaßt haben, haben wir festgestellt, das es wahrscheinlich beim Löschen der Fenster auftritt. Es wird wahrscheinlich ein Fenster doppelt gelöscht, entweder von uns oder der FourWheel-Gruppe.

7 Erfahrungssammlung

7.1 Anton Abaschin

Bei der Auswahl der Projektgruppe fiel meine Entscheidung auf Smart-Systems vor allem aus dem Anreiz "etwas zum Anfassen" zu entwickeln. Auch mein Interesse an hardwarenahen Softwareentwicklung mache die Wahl einfacher.

In der Seminarphase zu Beginn der Projektgruppe habe ich mich mit Insektenrobotern und vor allem ihrem mechanischen Aufbau beschäftigt. Deswegen habe ich mich bis zur Entwurfsphase an der Konzeption unserer Plattform beteiligt. Da ich, wie alle aus der Gruppe, keine Erfahrung mit CAD mechanischer Komponenten hatte, war diese Aufgabe mit viel Einarbeitungsaufwand verbunden. Ab dem Entwurf habe ich mich dann hauptsächlich mit der Entwicklung auf dem Host beschäftigt. Zu diesem Thema war auch mein zweiter Seminarvortrag.

Lieder konnte ich mich nicht sehr viel an anderen Teilen des Systems zu beteiligen, was mich auch interessiert hat. Obwohl man fast nur in seinem eigenen Bereich entwickelt hat, was es dadurch, dass wir die meisten Entscheidungen und den Verlauf der Arbeiten in der Gruppe diskutiert haben, möglich einen Überblick über alle Teile des Systems zu bekommen.

Bis auf die etwas langatmige Anfangsphase (Anforderungsdefinition, Entwurf) bin ich mit der PG sehr zufrieden. Die Projektgruppe war zwar mit viel Arbeit verbunden, was ich aber gelohnt hat. Ich konnte viel aus den verschiedenen Berreichen lernen und Stimmung in der Gruppe war sehr gut. Das einzige, was mich mir noch gewünscht hätte, ist mehr Zeit um die vielen Ideen, die man noch hat umzusetzen. Aber die fehl wohl bei jedem Projekt.

7.2 Carsten Kellermann

Ich habe mich während der Projektgruppe hauptsächlich um die Auswahl und Implementierung der Sensoren gekümmert. Die meiste Zeit habe ich mit der Quartus-Software und den Entwicklungsboards von Altera gearbeitet.

Zu Beginn der Projektgruppe hatte ich noch keine Erfahrung mit VHDL. Aus diesem Grund traten bei der Implementierung der ersten Sensormodule und der Programmierung des FPGA immer wieder Probleme auf, die die Vertigstellung des I^2C-Bus stark verzögerten. Am Anfang nutzten Gerold, der mit mir die Sensormodule entwarf, und ich das Programm Sonata von Symphony EDA um den I^2C-Bus zu simulieren bevor wir ihn dem Quartusprojekt hinzufügten. Dabei stellte sich leider heraus, dass eine erfolgreiche Simulation mit Sonata nicht bedeutete, dass Quartus den VHDL-Code kompilieren konnte. Zusätzlich bereitete mir anfänglich die parallelität der Prozesse in VHDL einige Probleme bei der Implementierung. Das hin und her zwischen Sonata und Quartus erschwerte die Implementierung zusätzlich und lies die erste Version des I^2C-Bus recht schnell sehr undurchsichtig werden.

Später nutzten wir für die Implementierung und die Simulation der Sensormodule nur noch die Quartussofware. Ausserdem implementierten wir die zweite Version des I^2C-Bus in vielen kleinen Zustandsautomaten, die auf verschiedene Prozesse verteilt wurden. Dieses, und die unsere gestiegenen VHDL-Kenntinisse, führten zu einer deutlich kürzeren Implementierungszeit für die anderen Sensormodule.

Insgesamt betrachtet hat es mir sehr viel Spass gemacht an dem Roboter zu arbeiten. Ich habe während der Projektgruppe recht viel dazulernen können. Zu den neu erworbenen VHDL- und C-Kenntnissen habe ich auch einiges aus dem Bereich Elektrotechnik und Mechanik dazugelernt. Auch die Zusammenarbeit unserer Gruppe, sowie deren Betreuung, hat meiner Meinung nach recht gut funktioniert.

Als einziges Manko möchte ich nur die Verfügbarkeit der Entwicklungsboards und der dazugehörigen Computer mit installierter Quartussoftware anmerken. Drei Entwicklungsboards waren für die grosse Projektgruppe zeitweise zu wenig.

7.3 Henning Kleen

Im Verlauf der Projektgruppe lag meine Hauptaufgabe in der Erstellung und Bestückung der Platine für den Insektenroboter. Da ich im Vorhinein keinerlei Erfahrungen im Design von Leiterplatten-Layouts hatte erwies sich diese Aufgabe oft als große Herausforderung.
Das Design der Leiterplatte wurde mit CADSTAR 6.0 entworfen. Das verwendete Tool machte eine längere Einarbeitungsphase nötig, da die Bedienung oftmals gewöhnungsbedürftig war. Besonders für den Entwurf der Schaltregler wurde ein Tool des Herstellers benutzt, um die Schaltung simulieren zu können, bevor das eigentliche Platinenlayout fertig ist. Dasselbe Simulationswerkzeug wurde auch verwendet um die auf der Platine vorhandenen analogen Schaltungsteile zu simulieren. Im Laufe des Projektes traten immer wieder Probleme mit CADSTAR zu tage, oftmals aufgrund dessen, dass sich das Bedienkonzept nicht an den etablierten Standards orientiert. Weiterhin waren öfter Abstürze des Programs zu verzeichnen.

Beim Entwurf der Platine wurde zunächst eine Bibliothek mit allen benötigten Bauteilen erzeugt, da die mitgelieferten Bauteilbibliotheken nur einen Bruchteil der benötigten Komponenten enthielt. Dieser Schritt erwies sich als zeitaufwendiger als zunächst vermutet, da eine Vielzahl von Dingen beachtet werden musste und die Korrektheit der sog. Footprints sehr wichtig war, da eine spätere Korrektur kaum, oder gar nicht möglich ist. Als nächster Schritt wurde der Schaltplan gezeichnet. Dabei wurde ein hierarchischer Ansatz gewählt. Als problematisch erwies sich die Übersichtlichkeit des Planes. Im Verhältnis zu der gesamten benötigten Zeit für das Platinenlayout ist der Zeitaufwand für die Erstellung des Schaltplans eher gering gewesen. Nachdem der Schaltplan fertig gestellt wurde, wurde begonnen aus dem Schaltplan ein Platinenlayout zu entwickeln. Zunächst wurden die Bauteile auf der für die Platine vorgesehenen Fläche platziert. Leider war es nicht möglich auf Unterstützung durch die

Auto-Placement und Auto-Routing Funktionen von CADSTAR zurückzugreifen, daher musste jede einzelne Leiterbahn von Hand gezeichnet werden. Dies erwies sich als der bei weitestem zeitaufwendigste Schritt bei der Erstellung des Layouts. Auch wurde das fertige Design wieder gründlich kontrolliert um Fehler in den Leiterbahnen zu verhindern. Trotz Kontrollen entstand ein Fehler beim Anschluss des Steckers für das Bluetooth- Modul, so dass dieser Stecker nicht nutzbar ist.

Die aus den Layout-Daten gefertigte Platine wurde dann zum größten Teil selbst Bestückt, entgegen ursprünglichen Befürchtungen erwies es sich als relativ unproblematisch selbst kleine SMD-Bauteile mit einem Rastermaß von 0.65 mm aufzulöten. Einzig FPGA und Bluetoothstecker erwiesen sich mit einem Rastermaß von 0.5 mm als etwas schwieriger zu verarbeiten.

Als Fazit lässt sich sagen, dass viele Schritte beim Layout der Platine erheblich länger dauerten, als zunächst vermutet. Somit kam es zu einer Verspätung des Fertigstellungstermins der Platine von fast 2 Monaten. Trotz der Probleme mit dem Layout-Tool und der mangelnden Erfahrung im Layout solcher Platinen stand am Ende des Projektes eine Platine, die die erwartete Funktionalität bietet. Einzig die Wahl der Logikpegelkonverter ist zu überdenken, da sich beim Testen der Platine herausstellte, dass diese nicht für den I^2C-Bus geeignet waren und sich auch bei anderen externen Komponenten die geringe Treiberstärke der Konverter als problematisch erwies.

Mit die Unterstützung durch die Betreuer war meiner Meinung nach sehr gut, da es zu jeder Zeit möglich war, offene Fragen und Probleme zu klären. Insgesamt bleibt festzustellen, dass es interessant war sich in einen Themenkomplex einzuarbeiten der normalerweise im Informatikstudium wenig behandelt wird.

7.4 Martin Kummer

In der Phase der Anforderungsdefinition war ich in der Hostgruppe. Wir trafen uns einige Male mit der 4WD-Hostgruppe und versuchten grob die Funktionalitäten zu definieren. Mein Schwerpunkt lag auf der Kommunikation. Wenig später wechselte ich dann aber in die Robotergruppe und übernahm dort die Definition und Implementierung der Kommunikation zusammen mit Andreas. Die Aufgabe Kommunikation erwies sich im Laufe der Zeit als ein sehr kontinuierlicher Prozess. Als Schnittstelle zwischen Roboter und Host mussten die Funktionalitäten den Anforderungen ständig angepasst werden. Da die Hardwareschnittstelle mit einem UART realisiert wurde konnten wir die Kommunikation anfangs mit einem Terminalprogramm und einem 0-Modem-Kabel testen und waren somit unabhängig vom Hostsystem. Später konnten wir den Host und den Roboter ebenfalls mit einem 0-Modem-Kabel verbinden, dieses wurde dann schließlich durch das Bluetooth-Modul ersetzt. Nach anfänglichen Konfigurations-Schwierigkeiten funktionierte dies auch ganz gut, allerdings waren wir durch die geringe Reichweite etwas enttäuscht (siehe „Systembewertung").

Eine weitere Aufgabe entstand, als wir feststellten, dass die Übertragung eines Bildes, bedingt durch die Geschwindigkeit des UARTs, knapp 10s dauern sollte. Wir entschlossen uns die Bilder zu komprimieren und implementierten eine einfache Run-Length-Encoding (RLE), die wir dann später noch erweiterten (siehe „Robotersoftware/Bildkompression"). Nicht zuletzt durch die überraschend guten Ergebnisse bedingt fand ich diese Aufgabe sehr interessant. Wir erzielten gut aussehende Bilder mit durchaus sehr hohen Kompressionsraten.
Die Idee Bilder durch Auslassen von Informationen zu komprimieren und diese dann durch Interpolation wieder herzustellen, konnte ich leider nicht zuendeführen, es fehlte dann die Zeit. Außerdem war das Problem bereits durch RLE zufriedenstellend gelöst.

Die Arbeit am Roboter gefiel mir sehr gut, da die Programmierung des eingebetteten Systems recht überschaubar war, es gab nicht mehr als 10 C-Dateien. Mit dem Gesamtergebnis können wir sicher alle zufrieden sein, da wir fast alles was geplant wurde auch realisiert haben. Im Nachhinein betrachtet gibt es zwar immer einige Sachen die man hätte anders implementieren können, ich denke aber das ist bei jedem Projekt so. Für mich war es auf jeden Fall sehr lehrreich, besonders weil ich vorher noch nichts mit eingebetteten Systemen gemacht hatte.
Die Zusammenarbeit mit den anderen Teilnehmern und die Betreuung durch Andreas war sehr gut. Außerdem lief alles immer sehr unbürokratisch ab. Das einzige Problem war eigentlich die Anzahl der Rechner im Verhältnis zur Anzahl der Personen. Besonders gegen Ende waren diese oft besetzt und man musste früh aufstehen um einen Platz zu bekommen.

7.5 Gerold Mauson

allgemein und VHDL

Wie Carsten schon geschrieben habe ich mich mit ihm zusammen um die Programmierung des FPGA's für die einzelnen Sensoren gekümmert. Bei der Erweiterung meiner Grundkenntnisse in VHDL halfen uns die Betreuer der Projektgruppe sehr und ich lernte viel über die Feinheiten von VHDL. Besonders die Implementierung des I^2C Busses war eine grosse Herausforderung, da er bidirektional ist und langsamer läuft. Daher musste man bei der Implementierung mit 2 verschiedenen Clocks arbeiten. Sehr hilfreich beim debuggen der VHDL-Tools war die Benutzung des intern in Quartus zur Verfügung gestellte Simulationsprogramm.

Quartus und Entwicklungsboards

Das arbeiten mit der Quartussoftware und den Entwicklungsboard war, von den unvermeidlich langen Kompilierungsphasen abgesehen, recht angenehm. Allerdings ergaben sich mit der Zeit viele kleine Schwierigkeiten, die die Entwicklung in die Länge zogen. So treiben einige pins des FPGA undokumentiert 5V bzw 3,3V , so dass es

sich empfiehlt die benutzten pins vor dem benutzten ersteinmal durchzumessen. Ein weiterer Stolperstein war die bei Quartus falsch implementierte ESB/ES Version des DMA-Controllers in Stratix (Was uns über 1 woche gekostet hat). Auch das richtige Einbinden des SDRAM-Controllers bereitete Schwierigkeiten. Wie wir feststellten muss der sdram mit einer 40 Grad phasenverschobenen Clock angesteuert werden, damit er richtig funktioniert. Bei allen Fehlersuchen das aufgrund dieser Probleme eine lange Zeit in Anspruch nahm erwies sich besonders das Oszilloskop als nützlich.

Betreuung und Gruppe

Im grossen und ganzen hat mir das Arbeiten am Roboter und an der Quartussoftware sehr viel Spass gemacht und war an vielen Stellen sehr lehrreich. Ich konnte viele Kenntnisse über VHDL, FPGA's, Quartus, Entwerfen von Schaltungen, Löten von Verbindungen, C-Programmierung und bedienen eines Oszilloskops hinzugewinnen. Das Arbeitsklima in der Gruppe war über die gesammte Zeit der Projektgruppe sehr angenehm. Die Betreuung der Gruppe war durchweg sehr angnhem, nützlich und hilfsbereit.

Es war allerdings festzustellen das auf den 3 zu Verfügung gestellten Rechenern zunächst unterschiedliche Quartus Versionen installiert waren, was zu vielen unnötigen Schwierigkeiten und Verzögerungen geführt hat. Die Erreichbarkeit der Schlüssel (am Anfang der Projektgruppe) und die Anzahl an Rechnern (vor allem an Ende der Projektgruppe) führten zwischenzeiltich zu Schwierigkeiten und Verlangsamung der Entwicklung.

7.6 Dimitri Nijasow

Im Laufe der Projektgruppe habe ich Erfahrung in vielen Bereichen gesammelt. Angefangen mit Informationsbeschaffung über Akkus, habe ich in der Entwurfsphase zur Ansteuerung von Servos gewechselt. An dieser Aufgabe habe ich dann bis zum Ende der Projektgruppe gearbeitet. Zusätzlich war ich der Meilensteinbeaftragte der PG. Außerdem habe ich bei der Entwicklung von Sensoren- und Kommunikationssoftware für den Roboter teilweise mitgewirkt.

Die Ansteuerung für Servos war für mich eine interessante Aufgabe. Manchmal ist sie sehr anspruchsvoll gewesen. Zum Beispiel hat das Drehen etwas Mathematikwissen verlangt, um richtige Positionen der Beine auszurechnen. Dabei handelte es sich nur um Steuerung. Wäre das ganze als eine Regelung implementiert, müsste man mit größerem Aufwand rechnen.

Eine Regelung war beim Insektenroboter leider nicht möglich, da wir keine entsprechenden Sensoren eingesetzt haben. Doch über die Stromaufnahme von Servos konnte man auf einen mechanischen Widerstand schließen und auf diese Weise Bodenkontakt herstellen, was ich persönlich ziemlich erstaunlich finde, weil keine zusätlichen Sensoren dafür benötigt werden. Das Potential der Strommessung von Servos haben wir

nicht vollständig ausgeschöpft. Man kann zum Beispiel damit auch feststellen, ob der Roboter gegen ein Hindernis gelaufen ist.

Da ich an der Ansteuerung von Servos gearbeitet habe, war ich daran interessiert, dass die Plattform gut gebaut war. Deswegen habe ich beim Bau des zweiten Prototypen mitgewirkt und einige Veränderungen vorgeschlagen. Später habe ich die Plattform gepflegt und defekte Servos ausgetauscht. Dabei habe ich sogar in die Miniaturwelt der Servos hineingeschaut und aus mehreren defekten Servos einige funktionsfähige gebastelt. Zum Anfang war das Austauschen von Servos eine mühsame Aufgabe. Mit der Zeit habe ich mehr Erfahrung gesammelt und einige Tricks ausgedacht, wie man es schneller machen kann. Jedes Mal habe ich mir jedoch gewünscht, dass wir stärkere Servos genommen hätten oder dass zumindes die Plattform etwas günstiger für Austausch von Servos konstruiert wäre.

In der Aufgabe als Meilensteinbeauftragter bin ich nicht voll dabei gewesen, da ich hauptsächlich mit Aktoren beschäftigt war. Es ist schwierig beide Jobs unter einen Hut zu bekommen. In unserem Fall war das auch so, dass jeder seine Meilensteine pflegen sollte. Das sorgte nur für Chaos. Ich kann nur empfehlen, diese Aufgabe als Hauptaufgabe an einen Teilnehmer zu vergeben. Bei einer Projektgruppe ist die Rolle des Managers schwierig zu bewältigen. Denn Gruppenteilnehmer können nicht ihre ganze Zeit der PG aufwidmen. Das alles zu managen ist auf jeden Fall eine große Herausforderung, doch unmöglich ist das nicht.

7.7 Jann Poppinga

Ich habe diese PG im 8. und 9. Semester gehört. Da ich schon im 3. und 4. Semester das Software-Projekt absolviert hatte, war Projektarbeit schon vertraut. Ich hatte das Gefühl, das wegen dieser Erfahrung, die ja auch bei den anderen PG-Teilnehmern vorhanden war, die Arbeit so gut gelang.

Obwohl nur leichten Druck von oben gab und die Verteilung der Aufgaben im Wesentlichen darauf basierte, dass sich jemand freiwillig meldete, wurde aufgrund der allgemein hohen Motivation viel erreicht. Da vom Erfolgswillen der einzelnen Projektmitglieder abgesehen kein Erfolgsdruck herrschte, gab es innerhalb der Gruppe Vorbehalte, die anderen Mitglieder der in die Pflicht zu nehmen, wenn sie Vorgaben und übernommene Aufgabe nicht zur vollen Zufriedenheit erfüllten. Man musste ja dankbar sein, dass sie sie überhaupt übernommen hatten. Diesem Problem ist meiner Ansicht nach allerdings nicht durch erhöhten Druck seitens der Betreuer beizukommen, da an der Universität Loyalität mit den Komilitonen höher bewertet wird als individuelle Leistung. Es gibt zum Beispiel Übungsgruppen, in denen eines der Mitglieder ohne Murren die gesamte Arbeit erledigt. Wenn eine solche Stimmung vorherrscht, kann auch der beste Meilensteinplan keine Wirkung entfalten.

Dieser Aspekt wog insgesamt aber nur gering: Ich fand die Projektarbeit sehr angenehm. Die allgemeine Motivation war hoch und die Talente waren so breit gestreut, dass sich die Teilnehmer der Gruppe gut ergänzten. Jeder übernahm Aufgaben so, dass insgesamt ein System entstehen konnte, das von der Platine bis zur automatischen Wegefindung den gesamten Umfang des Faches Informatik umfasst.

Die einzigen Haare in der Suppe stellen die Wahl der ungeeigneten Servos und die lange Lieferzeit dar. Durch diese beiden Faktoren war es nicht möglich, genug praktische Erfahrung mit dem Roboter zu sammeln um zum Beispiel die automatische Verbesserung von geschossenen Fotos über die Theorie hinaus einzurichten.

Die Betreuung war auch gut.

7.8 Andreas Prüllage

In der Projektgruppe lag mein Schwerpunkt auf der Kommunikation zwischen dem Host und dem Roboter. Hierbei habe ich mich zusammen mit Martin um die Implementierung auf der Roboterseite gekümmert. Als weiteres haben wir uns mit der Kompression des Fotos beschäftigt.
Unsere Aufgabe lag darin, das korrekte Senden und Empfangen von Befehlen zu realisieren. Hierbei waren insbesondere das Erstellen eines Protokolls sehr wichtig. Die Befehle mussten kodiert werden und ein Handshake wurde implementiert, um zu gewährleisten, das sowohl der Host als auch der Roboter bereit sind miteinander zu kommunizieren. Wir konnten die Sofware auf den Rechnern im Offis entwickeln und den Rechner mit einem 0-Modem-Kabel mit dem Entwicklungsboard verbinden. Hiermit konnten wir die Kommunikation simulieren und sehr gut testen, bis die Platine und das Bluetoothmodul zur Verfügung standen. Im Laufe des Projekts musste die Kommunikation ständig erweitert werden, da auf der einen Seite immer neue Befehle dazugekommen sind und sich auf der anderen Seite die Rückmeldungen zum Host geändert haben.
Des weiteren haben wir uns mit der Kompression des Fotos beschäftigt, da die Übertragung zum Host aufgrund der UART-Schnittstelle zu lange gedauert hat. Hierzu haben wir zwei Methoden entwickelt. Zum einen wird das Bild mittels Run Length Encoding komprimiert, zum anderen haben wir eine Kompression durch Auslassen von Pixeln vorgesehen.
Es war sehr interessant an der Insektensoftware zu arbeiten und ich konnte in dieser Zeit auch meine C-Programmierkenntnisse erweitern. Insgesamt war das Arbeitsklima in der Projektgruppe und auch die Betreuung sehr gut. Ich denke auch, das unsere Arbeit gut ausgefallen ist, da fast alles, was in der Anforungsdefinition geplant war, erreicht werden konnte.

7.9 Dietrich Schuckmann

Während der ersten Phase der Projektgruppe war es meine Aufgabe die mechanische Plattform zu entwerfen und sich später auch um deren Implementierung zu kümmern. In der zweiten Phase der PG, habe ich die Aufgabe übernommen Kommunikation zwischen Host und Insektenroboter auf der Hostseite zu implementieren. Bei dem Entwurf der Plattform habe ich mich mit der CAD-Entwicklung beschäftigt. Es war am Anfang ein wenig stressig, da wir seitens der Betreuer Druck zu spüren bekamen, wir sollen mit dem Entwurf der Plattform möglichst früh fertig werden, um später nicht in Zeitnot zu geraten. Nach der Einarbeitung hat es sogar Spass gemacht mit dem Programm zu ar-

beiten. Eine schöne Erfahrung war auch die Teile des Prototyps nach diesem Entwurf selbst herzustellen. Das war doch eine schöne Abwechslung zum eher theorielästigen Informatikeralltag.

Als die Arbeiten mit der Plattform abgeschlossen wurden, übernahm ich die Implementierung der Kommunikation auf der Hostseite. Ich konnte dort relativ schnell einsteigen, da ich mich vorher schon ein wenig mit Qt-Programmierung unter Linux beschäftigt habe. Was ein bisschen gestört hat, ist die Tatsache das wir nicht genug PC-Arbeitsplätze für unsere PG hatten. Vier Rechner für 2 Projektgruppen, auf den teilweise unterschiedliche Software installiert war und die auch noch auf 2 Räume verteilt waren sind keine optimale Arbeitsumgebung. Zumal man bei der Kommunikation oft gleich 2 Rechner benötigt hat um sie zu testen.

Ansonsten fand ich die Organisation und die Betreuung der Projektgruppe sehr gut. Wir konnten uns alle einbringen und hatten genügend Spielraum für eigene Ideen, nicht nur stumpfes abarbeiten der Vorgaben. Der Druck von der Betreuerseite kam immer zum richtigen Zeitpunkt und an der richtigen Stelle. Lediglich fand ich persönlich das Projektmanagement, besonders in der Anfangsphase für ein wenig übertrieben. Teilweise hat man mehr Zeit damit verbraucht sich mit den Dokumenten, die dem Projektmanagement dienten zu beschäftigen, als mit dem Projekt selbst. Zum Schluss hat es allerdings auch damit alles bestens funktioniert.

Teamarbeit in der Projektgruppe war sehr gut. Es gab eigentlich keine Person in der PG, die nicht dazu gehörte. Alle haben stets ihre Arbeit ohne großen Verspätungen und zumindest zufriedenstellend erledigt. Falls es etwas zu tun gab, haben sich fast immer Freiwillige gefunden, die dann die Arbeit übernahmen. Keiner musste dazu mit Strafen oder sonstigem genötigt werden. Und überhaupt fand ich sehr schön, dass wir ohne „Bürokratie“ ausgekommen sind.

Ich hätte mir gewünscht, dass die Zusammenarbeit mit der 4WD-Gruppe besser klappen würde. Teilweise hatte man den Eindruck, sie möchten sich von uns am besten komplett Abstand nehmen. Zusammen hätten wir noch viel mehr interessante Sachen realisieren können.

Insgesamt würde ich eine positive Bilanz für die Teilnahme an der PG „Smart Systems“ ziehen. Dass alles so gut klappt, hatte ich zu Anfang der Projektgruppe nicht gedacht. Ich wünsche unseren Nachfolgern, falls es solche geben sollte, noch viel Erfolg und viel Spass beim weiterentwickeln!

7.10 Michael Taphorn

Zu Beginn der Projektgruppe lag mein Schwerpunkt auf der Herstellung des Insekts. Wir trafen und regelmäßig in der Werkstatt der Universität um dort aus Lexan die einzelnen Teile herzustellen. Für Informatiker untypisch, wurde dort gesägt, gebohrt und geschraubt. Eine nette Erfahrung und sehr lehrreich.

Meine Hauptaufgabe war das Programmieren am Host-System. Hier wurde das Sensorfenster und Teile des Fotofensters von mir programmiert. Hier erstellte ich mit Qt eine QListView, in der Infrarot-, Kompass-, Neigungs-, und Ultraschallwerte des In-

sekts angezeigt werden. Außerdem ist es hier möglich sich die Stromaufnahme der Beinservos sowie die aktuelle Akkuspannung anzeigen zu lassen.
Bei dem Fotofenster wurden die drei verschiedenen Arten der Neupositionierung des Insekts von mir programmiert. Bei der Auswahl von Numberplate kann man die Eckpunkte des Nummernschildes eingeben, und es wird die Bewegung ausgeführt um das Nummernschild zentraler in das Bild zu bekommen. Bei der Auswahl von Fine, wird die Mitte des gezeichneten Rechtecks zentriert und dann mittels Infrarotsensor die Entfernung bestimmt. Und schließlich kann man mittels Coarse noch eine grobe Positionsänderung anfordern. Hier wird lediglich das Foto in 16 Teile unterteilt, und dann darauf geachtet welche von dem aufgezogenen Rechteck berührt werden. Die normalen Bewegungsbuttons auf dem Fotofenster wurden ebenfalls von mir umgesetzt.
Als persönliche Erfahrung nehme ich das C++ Programmieren mit. Da meine Programmiererfahrung sich bisher nur auf Java beschränkte, und auch das schon vor 4-5 Jahren war. Anfangs war es mühsam, aber nach dem lesen von 2 C++ Büchern wurde es besser. Außerdem konnte man in Notfällen meistens noch die anderen Grupppenmitglieder um Hilfe fragen.

Um nun endlich ein Fazit zu ziehen, muß ich sagen, das ich mit diesem Jahr absolut zufrieden bin. Ich habe was dazugelernt und die Gruppe hat (hoffentlich) ein gutes Produkt abgeliefert. Die Betreuung war gut und die Stimmung innerhalb der Gruppe ebenfalls.
Als einzigen Kritikpunkt möchte ich das Fehlen von weiteren Linux-Rechnern nennen. 3 Linux-Rechner im Offis für beide Gruppen war (besonders in der Endphase) vielleicht doch zu wenig.

8 Fazit

Wie bereits in den Erfahrungsberichten zu lesen ist, fällt auch das Fazit positiv aus. Wir hatten insgesamt eine gute Zusammenarbeit innerhalb der Gruppe, haben alle etwas dazugelernt, wurden gut betreut und zuguterletzt ist auch noch ein Roboter entstanden der die geforderten Aufgaben erfüllt. Ein wirkliches Problem stellten lediglich die Servos dar, die zum Ärger aller immer zu schnell ihren Geist aufgaben und dadurch ständig gewechselt werden mußten. Ein nicht so schwerwiegendes, aber trotzdem reales Problem war die geringe Reichweite via Bluethooth.
Abschließend bleibt zu sagen, dass wir „Smart-Systems“ als ein lehrreiches und erfolgreiches Projekt betrachten.